UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ROBIN HOOD
Um Ambiente para a Avaliação de Políticas de Balanceamento de Carga

por

MAURO LÚCIO BAIONETA NOGUEIRA

Dissertação submetida à avaliação, como requisito parcial para
a obtenção do grau de Mestre em
Ciência da Computação

Prof. Cláudio Fernando Resin Geyer
Orientador

Prof. Simão Sireneo Toscani
Co-orientador

Porto Alegre, outubro de 1998.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Nogueira, Mauro Lúcio Baioneta

ROBIN HOOD: um ambiente para a avaliação de políticas de balanceamento de carga / por Mauro Lúcio Baioneta Nogueira. - Porto Alegre: CPGCC da UFRGS, 1998.

127f. : il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul. Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, BR-RS, 1998. Orientador: Geyer, Cláudio Fernando Resin. Co-orientador: Toscani, Simão Sirineo.

1. Balanceamento de carga 2. Sistemas distribuídos 3. Análise de desempenho I. Geyer, Cláudio Fernando Resin. II. Toscani, Simão Sirineo. III. Título.

Arquitetura de Computadores_SB
Processamento distribuído
Sistemas distribuídos
Análise: Desempenho
Balanceamento de Carga
CNPq 1.03.03.00-6

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profa. Wrana Panizzi
Pró-Reitor de Pós-Graduação: Prof. José Carlos Ferraz Hennemann
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenadora do CPGCC: Profa. Carla Maria Dal Sasso Freitas
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

DEL RIGOR EN LA CIENCIA

... En aquel Imperio, el Arte de la Cartografía logró tal Perfección que el mapa de una sola Provincia ocupaba toda una Ciudad, y el mapa del Imperio, toda una Provincia. Con el tiempo, esos Mapas Desmesurados no satisficieron y los Colegios de Cartógrafos levantaron un Mapa del Imperio, que tenía el tamaño del Imperio y coincidía puntualmente com él. Menos Adictas al Estudio de la Cartografía, las Generaciones Siguientes entendieron que eso dilatado Mapa era Inútil y no sin Impiedad lo entregaron a las Inclemencias del Sol y de los Inviernos. En los desiertos del Oeste perduran despedazadas Ruinas del Mapa, habitadas por Animales y por Mendigos; en todo el País no hay otra reliquia de las Disciplinas Geográficas.

SUÁREZ MIRANDA: Viajes de varones prudentes,
Libro Cuarto, cap. XLV, Lérida, 1658.

JORGE LUIS BORGES, 'EL HACEDOR'

Agradecimentos

Lembrar é correr o imperdoável risco de esquecer. Que os esquecidos me perdoem do imperdoável pecado de tê-los esquecidos.

Esta lista de agradecimento, de certa forma, se apresenta como uma radiografia do que foi minha vida durante o período de meu mestrado. Troquei os paralelos e resolvi colocar três estados entre mim e meus familiares e amigos. Mas penso que valeu a pena. Ganhei um belo resfriado em meu primeiro inverno, e os conhecidos da primavera tornaram-se bons amigos no final do outono. Selei fortes amizades e descobri pessoas imprescindíveis.

Em um primeiríssimo lugar, agradeço aos meus pais, de infinita paciência com este tão sempre insatisfeito filho. Esta dissertação é fruto da esperança que em mim depositaram e dos inesgotáveis incentivos que deles recebi. A eles dedico esta dissertação. Agradeço também aos meus irmãos Sânzio e Sâmia, pelo afeto, carinho e amor.

Agradeço a minha namorada Cristina Melchiors, cúmplice deste trabalho, pelo apoio e pelo carinho, por compartilhar minhas dúvidas, anseios e medos, pelo ombro amigo.

Duas pessoas espero para sempre guardar, no lado esquerdo do peito, debaixo de sete chaves. A Angela e Sérgio Ribeiro, pais-procuradores, tutores-representantes, pessoas incríveis, de valor ímpar, meus mais sinceros agradecimentos, por tudo que fizeram e ainda fazem por mim.

Agradeço a toda família Melchiors — Djalmar, Leila, Lúcia e Ana Paula — por me acolherem de braços tão abertos e por carinhosamente acompanharem a gestação tão demorada deste trabalho.

Ao meu orientador Cláudio Geyer pela inesgotável paciência, pelos valiosos conselhos, pela constante motivação, pelo companheirismo, meus sinceros agradecimentos. Agradeço também ao meu co-orientador Simão Toscani pela disponibilidade, pela atenção, pelos conselhos.

Aos funcionários do CPGCC, à fantástica equipe da biblioteca (Henrique, Zita, Adriana, Ida, Bia e todos os demais), a Eliane, à equipe da administração da rede, meu muito obrigado.

Por fim, agradeço ao CNPq pelo auxílio financeiro que possibilitou a realização deste trabalho, a UFRGS pela oportunidade de gratuitamente realizar este mestrado e aos professores desta casa pelo prestimoso auxílio e pelos conhecimentos recebidos.

Sumário

Lista de Abreviaturas ou Siglas

CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
CPGCC	Curso de Pós-Graduação em Ciência da Computação
CSL	Control and Simulation Language
ECSL	Extended Control and Simulation Language
GASP	General Activity Simulation Program
GPSS	General Purpose Simulation System
GUI	Graphical User Interface
LAN	Local Area Network
SIMAN	Simulation Analysis
SIMULA	Simulation Language
SLAM	Simulation Language for Alternative Modeling
UFRGS	Universidade Federal do Rio Grande do Sul
WAN	Wide Area Network

Lista de Figuras

Lista de Tabelas

Resumo

É ponto passivo a importância dos sistemas distribuídos no desenvolvimento da computação de alto desempenho nas próximas décadas. No entanto, ainda muito se debate sobre políticas de gerenciamento adequadas para os recursos computacionais espacialmente dispersos disponíveis em tais sistemas. Políticas de balanceamento de carga procuram resolver o problema da ociosidade das máquinas(ou, por outro lado, da super-utilização) em um sistema distribuído. Não são raras situações nas quais somente algumas máquinas da rede estão sendo efetivamente utilizadas, enquanto que várias outras se encontram subutilizadas, ou mesmo completamente ociosas. Aberta a possibilidade de executarmos remotamente uma tarefa, com o intuito de reduzirmos o tempo de resposta da mesma, ainda falta decidirmos "como" fazê-lo. Das decisões envolvidas quanto à execução remota de tarefas tratam as políticas de balanceamento de carga. Tais políticas, muito embora a aparente simplicidade quanto às decisões de controle tomadas ou ao reduzido número de parâmetros envolvidos, não possuem um comportamento fácil de se prever. Sob determinadas condições, tais políticas podem ser tornar excessivamente instáveis, tomando sucessivas decisões equivocadas e, como conseqüência, degradando de forma considerável o desempenho do sistema. Em tais casos, muitas das vezes, melhor seria não tê-las. Este trabalho apresenta um ambiente desenvolvido com o objetivo de auxiliar projetistas de sistema ou analistas de desempenho a construir, simular e compreender mais claramente o impacto causado pelas decisões de balanceamento no desempenho do sistema.

Palavras-Chave: balanceamento de carga; sistemas distribuídos; análise de desempenho.

TITLE: "ROBIN HOOD: AN ENVIRONMENT TO LOAD BALANCING POLICIES EVALUATION"

Abstract

There is no doubts about the importance of distributed systems in the development of high performance computing in the next decades. However, there are so much debates about appropriated management policies to spatially scattered computing resources available in this systems. Load balancing policies intend to resolve the problem of underloaded machines (or, in other hand, overloaded machines) in a distributed system. Moments in which few machines are really being used, meanwhile several others are underused, or even idle, aren't rare. Allowed the remote execution of tasks in order to decrease the response time of theirs, it remains to decide 'how' to do it. Load balancing policies deal with making decisions about remote execution. Such policies, in spite of the supposed simplicity about their control decisions and related parameters, doesn't have a predictable behavior. In some cases, such policies can become excessively unstable, making successive wrong decisions and, as consequence, degrading the system performance. In such cases, it's better no policy at all. This work presents an environment developed whose purpose is to help system designers or performance analysts to build, to simulate and to understand the impact made by balancing decisions over the system performance.

Keywords: load balancing; distributed systems; performance analysis.

1 Introdução

A evolução tecnológica dos últimos anos deu margem à proliferação do que conhecemos por sistemas distribuídos. Hoje em dia, os sistemas distribuídos não mais estão restritos exclusivamente a raros laboratórios de pesquisa. Ao contrário, são de uso comum e difundidos [MIL 94]. Em tais sistemas, naturalmente surge o problema da gerência dos recursos computacionais, espacialmente dispersos entre os vários centros de processamento que compõem o sistema. O gerenciamento dos recursos não se limita aqui à observância simples das demandas ao longo do tempo. Uma nova dimensão é acrescida às decisões de gerência: o espaço. Deve-se decidir agora não somente pelo momento adequado de quando satisfazer uma dada demanda, mas também pelo local onde esta demanda será satisfeita.

Coibir o acesso a recursos remotos, restringindo a execução dos processos necessariamente à máquina que os originou, implicaria sumariamente desconsiderar eventuais ganhos de desempenho que poderiam ser obtidos através do compartilhamento da carga computacional. Não é de todo incomum encontrarmos uma situação em que somente parte das máquinas que compõem o sistema está sendo efetivamente utilizada, enquanto as demais se encontram ou subutilizadas, ou completamente ociosas [THE 85][NIC 87]. Tais circunstâncias — onde, por um lado, há máquinas sobrecarregadas e, por outro, máquinas ociosas — clamam por um gerenciamento eficaz dos recursos, que contemple um universo de decisão mais amplo que o de uma única máquina, que seja capaz de diluir, entre as máquinas ociosas, os piques de processamento que, por vezes, acometem algumas das máquinas do sistema.

O balanceamento da carga computacional entre as máquinas, isto é, a transferência de carga, em momentos apropriados, de uma máquina para outra, se apresenta como solução a este impasse. Todavia, restam as perguntas: (*i*) quais seriam os referidos momentos apropriados? quais situações dariam ensejo a uma ação de redistribuição? (*ii*) qual carga transferir entre as existentes? (*iii*) para qual máquina transferir? As respostas a tais perguntas coletivamente constituem o que chamamos de política de balanceamento. Seria, então, equivalente questionarmos: qual política de balanceamento seria a mais adequada, ampliando ao máximo, dentro dos limites do possível, os ganhos de desempenho do sistema?

Esta pergunta, conquanto pareça conduzir a uma fácil resposta, traz consigo uma série de considerações de difícil análise. O afinamento dos parâmetros da política de balanceamento, o comportamento desta frente a situações limites (tais como carga excessivamente baixa ou, ao contrário, demasiadamente elevada), o controle da instabilidade que permeia o sistema, a gerência das informações que fundamentarão as decisões de balanceamento, a própria escolha dos índices de relevância (sejam para exprimir o desempenho do sistema, sejam para mensurar o estado de carga de uma da máquina), e várias outras questões devem ser examinadas ao escolhermos uma política entre as demais.

Estas adversidades sugerem a criação de uma ferramenta específica para a análise do problema de balanceamento de carga. Ferramenta que permita a descrição facilitada da política desejada (ou das políticas), e a visualização do desempenho do sistema frente à variação de uma vasta gama de parâmetros. É de se observar que, na

maior parte dos trabalhos encontrados sobre balanceamento, não há menção explícita sobre a utilização de uma ferramenta de análise similar.

A construção de uma ferramenta de análise específica ao problema de balanceamento de carga foi, dentro deste contexto, tomada como meta de trabalho. A dissertação que aqui se apresenta procurará descrever os resultados desta tarefa, **o ambiente Robin Hood**. O ambiente que aqui propomos foi concebido com o intuito de permitir, de forma ágil, o desenvolvimento de políticas de balanceamento e a análise das mesmas frente à variação de uma vasta gama de parâmetros.

O próximo capítulo procura aclarar alguns conceitos básicos, imprescindíveis ao entendimento do restante do texto. A vaga noção 'desempenho' será limitada à extração e verificação de alguns índices numéricos do sistema em estudo. A análise do desempenho do sistema será compreendida, portanto, como a análise dos referidos índices numéricos. Inúmeras técnicas podem ser empregadas a fim de extrair os índices que representam o desempenho do sistema. O ambiente Robin Hood optou por construir um modelo de balanceamento e exercitar, através de uma simulação discreta dirigida por eventos, o modelo construído. Adotou, por conseguinte, uma técnica de modelagem, onde os índices de desempenho são extraídos a partir do modelo concebido, e não do próprio sistema real. Caberá ainda ao capítulo 2, explanar sobre as técnicas adotadas pelo ambiente, a saber, sobre as técnicas de modelagem e, dentre estas, especialmente, sobre as técnicas de simulação discreta. A real necessidade do capítulo 2 será reforçada mais à frente. Ao apresentar ao leitor os detalhes internos de implementação do ambiente Robin Hood, o capítulo 5 se servirá largamente dos conceitos introduzidos no capítulo 2.

O capítulo 3 se atém exclusivamente ao problema de balanceamento de carga. Procura compilar os principais trabalhos encontrados na literatura sobre o tema, sistematizando-os. Desta forma, o conteúdo apresentado ao leitor não se limita à simples exposição destes trabalhos. Através de uma leitura cuidadosa dos mesmos, foram identificados os principais tópicos, ou elementos, constituintes do problema de balanceamento de carga. As referências feitas aos trabalhos encontrados sobre o tema se justapõem, portanto, aos elementos constituintes identificados. Ao falarmos, por exemplo, sobre a especificação da demanda, citaremos, neste contexto, somente os posicionamentos divergentes acerca do assunto abordado.

O capítulo 4 discorre sobre as funcionalidades oferecidas pelo ambiente Robin Hood. Cada um dos elementos constituintes do modelo identificado no capítulo anterior receberá uma especial atenção por parte do ambiente Robin Hood. Neste capítulo estão descritos os recursos oferecidos pelo ambiente a seu potencial usuário. O tratamento dado pelo ambiente à questão dos índices de desempenho, dos índices de carga, da especificação da demanda e da topologia, e sobre outros assuntos aludidos em momentos anteriores serão pauta deste capítulo.

Ao passo que o capítulo 4 apresenta o ambiente Robin Hood sob a ótica do usuário, privilegiando as facilidades que este oferece, o capítulo 5 o faz sob a ótica do programador, desnudando os detalhes internos de implementação. O capítulo 5 explana sobre como foi concebido e construído o ambiente Robin Hood. Discorre sobre as várias camadas de software que, juntas, constituem o ambiente. Precisa o funcionamento do simulador de propósito geral que subjaz à base das demais camadas de software que compõem o ambiente e elucida a lógica do algoritmo de simulação que este executa.

Explica a construção de um modelo de balanceamento sobre a camada do simulador de propósito geral e enumera as entidades pertencentes a este modelo de balanceamento. Apresenta uma biblioteca contendo políticas de balanceamento clássicas já implementadas e, por fim, discorre sobre a interface gráfica criada para o ambiente.

Encerrando o texto, uma análise crítica sobre o ambiente é feita no último capítulo. São confrontados os pontos positivos e negativos do ambiente e uma lista de metas é traçada como trabalho futuro.

2 Dos Aspectos de Modelagem

Neste capítulo serão abordadas questões relativas à análise de desempenho de um sistema. A vaga noção evocada pelo termo 'desempenho' será restringida e vinculada à observação de índices numéricos extraídos de um sistema em estudo (índices de desempenho).

Em seguida, serão apresentadas as principais fases de um estudo de desempenho. Das técnicas empregadas para a extração dos índices, o texto ater-se-á somente às de modelagem, sendo omisso quanto às técnicas de medição, por estas serem julgadas prescindíveis à compreensão e completeza desta dissertação. As técnicas de modelagem se dividem em duas categorias: técnicas analíticas e técnicas de simulação. Uma breve explanação será dada sobre cada uma destas técnicas, sendo a atenção dirigida especificamente, no caso das técnicas analíticas, para a Teoria das Redes de Fila, e no caso das técnicas de simulação, para os modelos de simulação discretos. A mesma alegação feita quanto à omissão das técnicas de medição será apresentada para os modelos de simulação contínuos: são prescindíveis à compreensão e à completeza.

A final do capítulo, será fornecida, a título de exemplo, a implementação de um modelo simplificado em uma linguagem de simulação discreta orientada a eventos. As razões para fazê-lo se devem ao fato de o ambiente Robin Hood se estruturar internamente de forma análoga a uma linguagem de simulação discreta orientadas a eventos e de oferecer primitivas similares às encontradas em tais linguagens.

2.1 Avaliação de Desempenho

Os sistemas computacionais, além de cumprirem corretamente as funções de processamento de informação para as quais foram projetados, devem satisfazer critérios mínimos de desempenho a fim de que possam ser utilizados. O funcionamento correto de um sistema computacional não implica que este seja útil, visto que, caso o mesmo possua um desempenho sofrível, dificilmente arrebanhará futuros usuários ou justificará os custos para a sua aquisição. Poder-se-ia, até mesmo, dizer que a satisfação de critérios mínimos de desempenho é, entre outras, uma condição indispensável ao se aferir o funcionamento correto do sistema.

Obedecer certas especificações de desempenho pode ser entendido grosseiramente como quão bem o sistema é capaz de executar suas funções [FER 78]. Assim como podemos formular várias exigências sobre o desempenho de um automóvel — velocidade máxima alcançada, consumo de combustível, estabilidade, facilidade de uso, conforto, espaço interno, *etc.* —, o mesmo podemos fazer para o desempenho esperado de um sistema computacional. Algumas exigências possuem um caráter qualitativo, não sendo mensuráveis, o que dificulta verificarmos a satisfação das mesmas por parte do sistema, fato que não ocorre para várias das demais, que possuem natureza numérica e, portanto, estão sujeitas a uma análise e verificação imediatas.

Embora, de uma maneira ampla, o conceito para desempenho seja subjetivo, podemos, muitas das vezes, traduzi-lo quantitativamente, encontrando expressões numéricas para percepções qualitativas sobre o sistema. A estas exigências de

desempenho de natureza numérica chamaremos doravante de **índices de desempenho** [FER 78].

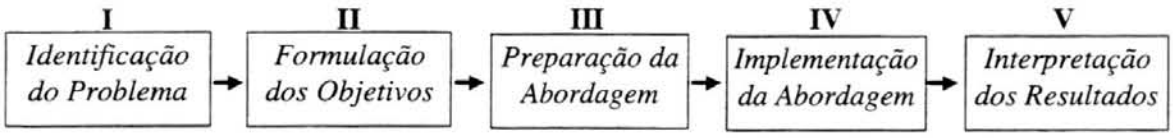
Pelo fato de serem quantificáveis, os índices de desempenho podem ser objetos de uma avaliação. Entendemos por avaliação de desempenho o conjunto de atividades técnicas com o propósito de extrair de um determinado sistema os índices de desempenho considerados relevantes, seja através de medição, seja através de estimativa ou cálculo [FER 83].

2.2 Estudos de Avaliação

Antes de nos aventurarmos a avaliar o desempenho de um sistema, temos que ter em mente, de maneira bem definida, os motivos, os métodos empregados e os propósitos para tal avaliação. A figura 2.1 apresenta as várias fases pelas quais terá de passar o analista na confecção de seu estudo de desempenho.

A diagnose do problema, ou seja, a constatação das razões para se levar a cabo um estudo de avaliação, é o primeiro passo a ser dado (fase I). Entre possíveis razões, algumas bem comuns seriam: a identificação de pontos de elevada contenção no sistema; a necessidade de se comprovar a viabilidade de um determinado projeto de sistema; uma futura alteração na configuração atual dos componentes do sistema; uma futura alteração na forma ou intensidade de uso do sistema.

Aliados às razões, estão os objetivos que se pretende alcançar com a elaboração de um estudo sobre o desempenho (fase II). A especificação minuciosa destes contribui consideravelmente para o sucesso de um estudo de avaliação. Informalmente, pode-se citar entre os objetivos mais encontrados: a comparação entre dois ou mais sistemas; a determinação de valores ótimos para certos parâmetros de um sistema (afinamento do sistema); a eliminação de pontos de contenção; a caracterização da carga de trabalho que é submetida ao sistema quotidianamente; o prognóstico do sistema frente a determinadas cargas de trabalho (previsão de comportamento).



Fase I: Surge a necessidade de se fazer o estudo
Fase II: Os objetivos do estudo são formulados
Fase III: Um plano de estudo é preparado
Fase IV: O plano é implementado
Fase V: Os resultados são interpretados

FIGURA 2.1 - Fases de um estudo de desempenho

Os índices de desempenho relevantes, aqueles que fornecem uma visão do sistema adequada aos propósitos do estudo em questão, devem ser escolhidos ao final da segunda fase, uma vez que já se têm em mãos os objetivos delineados.

É de se observar que, para fazer a análise de desempenho de um sistema, não há sequer a necessidade de o sistema existir realmente. Existem várias técnicas que podem ser empregadas a fim de se coletar as informações de desempenho almejadas. Nas fases seguintes, III e IV, o analista deverá escolher entre obter as informações de desempenho a partir do próprio sistema (técnicas de medição) ou obter tais informações a partir de um modelo (técnicas de modelagem).

Na verdade, apesar da linearidade aparente das fases mostrada na figura 2.1, freqüentemente um estudo de avaliação de desempenho resulta em um ciclo de verificação de conjecturas. Uma hipótese é formulada (fase III), testada (fase IV) e, caso os resultados não a corroborem (fase V), modificada. Da mesma forma, é comum, somente ao final de um estudo, perceber que os índices utilizados, ou mesmo os objetivos formulados, estavam equivocados, obrigando, então, a uma nova iteração pelo ciclo de estudo [FER 78].

Nas subseções seguintes, detalharemos a sistemática referente às técnicas de modelagem. Desconsideraremos neste texto as técnicas de medição, embora ressaltando, aqui e ali, quando necessário, alguns conceitos por elas introduzidos. Fazemo-lo por dois motivos: por brevidade e por julgar que tal decisão não prejudicará, de nenhuma maneira, a compreensão das informações restantes contidas neste texto.

2.3 Modelagem de Sistemas

Um modelo nada mais é que a descrição de um sistema: a especificação tanto dos objetos que compõem estruturalmente o sistema quanto dos relacionamentos existentes entre estes objetos [SOA 90].

Um modelo pode ser encarado como uma abstração do sistema, isto é, um croqui, onde somente linhas mestras são expostas, de forma a se obter uma visão panorâmica que não é obscurecida por minúcias excessivas, não relegando, no entanto, contornos que, ausentes, desfigurariam o conjunto. Detalhes de menor significância à compreensão do sistema são desconsiderados, levando-se em conta, na definição do modelo, somente aspectos de relevância à mecânica de funcionamento. Pelo observado, conclui-se que vários modelos podem ser derivados de um único sistema físico, cada um deles privilegiando um aspecto ou ângulo próprio.

Aos objetos constituintes de um modelo, com funcionalidade e estado associados, damos o nome de **entidades**. Aos elementos do modelo cujos valores caracterizam o estado de uma entidade damos o nome de **atributos**. O estado global do modelo pode, assim, ser definido como o produto cartesiano dos estados de cada uma das entidades. Conceitos similares podem ser encontrados em [COT 92] [SOA 90] [SHA 75] [GOR 69].

Às técnicas que se valem de um modelo para dele extrair as informações de desempenho necessárias à execução dos objetivos formulados, chamaremos de **técnicas de modelagem**. As técnicas de modelagem se dividem em duas categorias: **técnicas analíticas** e **técnicas de simulação**.

Ambas as técnicas procuram, através de mecanismos próprios, representar o funcionamento do sistema ao longo do tempo; descrever como o sistema evolui, como os estados, pelos quais o sistema passa, se sucedem temporalmente. Para isto, exercitam um modelo definido a partir do sistema e observam as respostas fornecidas por este modelo. Caso o modelo seja válido — isto é, exista um grau de confiança aceitável entre os comportamentos gerados pelo exercício do modelo e os comportamentos apresentados na realidade —, os resultados fornecidos são confiáveis e o estudo de desempenho logra êxito. Caso contrário, o modelo terá de ser modificado, acrescentando-lhe detalhes então omissos, ou mesmo reestruturando-o.

As técnicas analíticas almejam, através do estabelecimento de relacionamentos matemáticos entre as entidades do modelo, encontrar expressões analíticas, sejam exatas, sejam aproximadas, para os índices de desempenho selecionados.

As técnicas de simulação — ressalvando que, ao mencioná-las, estamos nos referindo particularmente àquelas passíveis de simulação por um computador digital [SOA 90] [SHA 75] —, trilham outros caminhos. Estas representam as entidades do modelo como uma coleção de atributos (objetos de dados), mimetizando a dinâmica do sistema através da execução ordenada de procedimentos computacionais, responsáveis estes pelas alterações nos atributos das entidades e conseqüente mudança do estado global. O funcionamento do sistema seria, mesmo que de forma simplificada, reproduzido pela execução ordenada de tais procedimentos.

Ambas as técnicas possuem os seus méritos próprios. Uma forma analítica fechada para os principais índices de desempenho seria a ferramenta ideal para analisarmos o comportamento de um sistema. Infelizmente, a fim de manter o problema matematicamente tratável, muitas simplificações devem ser feitas, o que pode conduzir a um modelo, por sua extrema simplicidade, inválido. Entretanto, para casos específicos, existem formas de se contornar tal questão, em que se consegue, a despeito da elevada complexidade do modelo, extrair, para os índices de desempenho, soluções analíticas que constituem aproximações aceitáveis, próximas dos valores esperados [SAL 81] [NAI 85] apud [SOA 90].

Quando isto não se aplica, somos obrigados a lançar mão de técnicas de simulação. As técnicas de simulação conseguem, mesmo para modelos extraordinariamente complexos e detalhistas, fornecer respostas aceitáveis. O ônus desta versatilidade e maleabilidade são intermináveis horas computacionais despendidas. Aqui, estamos preocupados em simplificar o modelo não por tratabilidade matemática, mas sim para, após enxugá-lo de minúcias desnecessárias, reduzir o tempo despendido pelo computador ao simular o sistema.

As técnicas de simulação açambarcam uma faixa maior de modelos, permitindo um grau de descrição e uma riqueza de detalhes bem superior aos permitidos pelas técnicas analíticas. Todavia, exigem certa paciência por parte do analista, pois demandam tempo e esforço para serem implementadas e simuladas [FER 83].

Nas seção seguinte, introduziremos, como exemplo de técnica analítica, a teoria das redes de filas. Isto não é feito em vão porque, em tal apresentação, estarão contidos termos e conceitos usados nas demais partes do texto. Além do mais, a maioria das linguagens e ambientes de simulação e, em especial, o ambiente de simulação descrito

por esta dissertação adotam estruturas de modelos inspiradas na teoria de filas, fornecendo aos usuários conceitos como os de transação, filas, centros de serviço, etc., seja na forma de primitivas da linguagem, seja, indiretamente, através da combinação de primitivas que facilitem a implementação de tais conceitos [KAR 91] [MAK 91] [VAU 91] [RUS 90].

Posteriormente, abordaremos as técnicas de simulação, concentrando nossas atenções na técnica de simulação discreta dirigida por eventos. Esta técnica particularmente nos interessa por servir de fundamento à construção do ambiente Robin Hood, proposto nesta dissertação. Urge, portanto, que a compreendamos.

2.4 Técnicas Analíticas – Teoria das Redes de Filas

A teoria das redes de filas constitui uma poderosa ferramenta matemática amplamente utilizada na análise e modelagem de sistemas reais. Embora tivesse, como origem, a intenção específica de analisar o comportamento de sistemas telefônicos, hoje em dia é aplicada a uma vasta gama de sistemas, e seus conceitos básicos jogam luz sobre diversas áreas do conhecimento [TAN 89].

A teoria das redes de filas enxerga os sistemas a serem modelados como estruturalmente compostos por usuários que chegam, esperam por sua vez em uma fila, demandam por algum tipo de serviço ou recurso, são atendidos e, por fim, ou deixam definitivamente o sistema, ou partem para outro centro de serviço.

Uma vez que um recurso está sendo utilizado por um usuário, os demais usuários, ao requisitarem a utilização desse recurso, deverão esperar pela liberação do mesmo em uma fila, gerando contenção quanto ao uso do recurso. O grau de contenção, como também as quantidades de recursos requisitadas, determinarão categoricamente o comportamento do modelo. Modelos que não sofrem do problema de contenção quanto ao uso dos recursos são usualmente de fácil análise [SOA 90].

Podemos, pois, observar, na teoria das redes de filas, a existência das seguintes espécies de entidades com papéis distintos e complementares:

- **usuários**, são as entidades que recebem algum tipo de serviço ou utilizam os recursos disponibilizados;

- **servidores**, são as entidades que prestam algum tipo de serviço, ou disponibilizam algum recurso;

Assim, clientes de um banco, processos em um sistema operacional e tarefas em uma unidade fabril são possíveis exemplos de entidades usuárias, enquanto que, por outro lado, um caixa de banco, um processador de um computador, ou uma máquina de uma linha de produção são exemplos de entidades servidoras.

Um centro de serviço pode ser composto de um ou mais servidores e pode possuir uma ou mais filas de espera para o atendimento aos usuários. Os caminhos percorridos pelos usuários através dos vários centros de serviços determinam a topologia de interconexão da rede de filas.

A dinâmica do modelo, por sua vez, seria determinada por uma série de fatores, entre os quais podemos citar [KIN 90]:

- a distribuição de tempo entre chegadas apresentada pelos usuários (**padrão de chegada**);
- a distribuição de tempo acerca dos vários serviços requisitados, ou de quantidade acerca do volume de recursos demandados (**padrão de serviço**);
- o número de servidores;
- as políticas de escalonamento adotadas pelos servidores na escolha de qual será o próximo usuário a ser atendido em seu centro de serviço (**disciplinas de fila**);
- o comprimento máximo permitido para as filas de espera;

As distribuições acima mencionadas seriam, para o modelo, a contrapartida dos padrões de funcionamento do sistema real. Como a maioria dos sistemas de interesse não possui um comportamento determinístico, seja para os padrões de serviço, seja para os padrões de chegada, tais distribuições, no modelo, são representadas através do emprego de funções de distribuição de probabilidades. Tenta-se captar, com isto, ao se descrever o modelo, a imprevisibilidade inerente à maioria dos sistema encontrados.

Na figura 2.2, extraídos de [SOA 90], são apresentados alguns exemplos de possíveis centros de serviços.

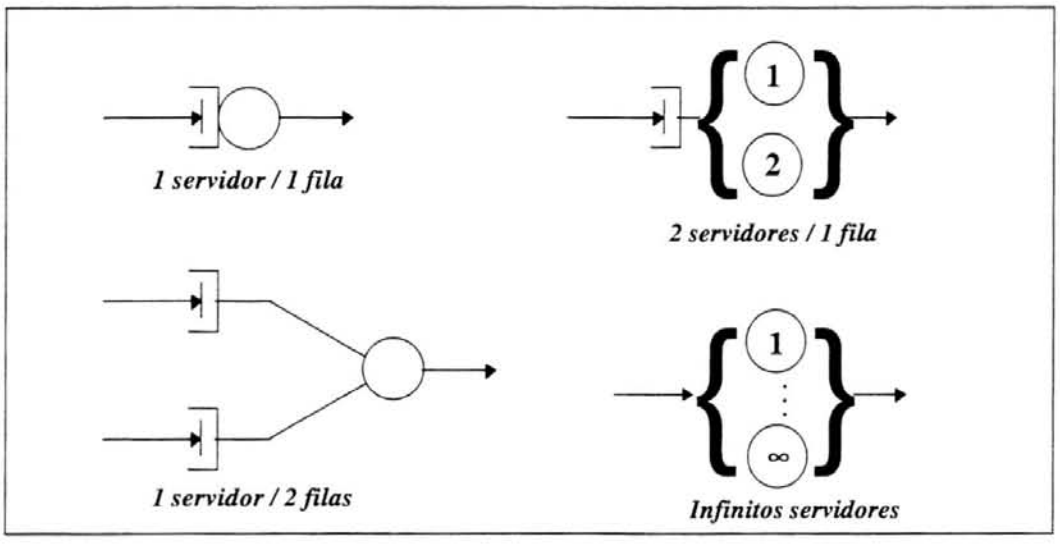


FIGURA 2.2 - Centros de serviço

Uma notação para descrever os modelos de teoria de filas (notação de Kendall) bastante difundida é a mostrada abaixo, que consiste de letras e números separados em campos por uma barra '/' [KIN 90]:

A/B/c/n/p

A primeira letra, ***A***, e a segunda, ***B***, descrevem respectivamente as distribuições de probabilidade associadas ao padrão de chegada (intervalo de tempo entre chegadas) e ao padrão de demanda de serviço (tempo de serviço). Tais letras são extraídas de um reduzido conjunto de descritores:

- ***D***, significando DETERMINÍSTICO. Indica que a variável aleatória da referida distribuição pode assumir um único valor constante, a média;

- ***M***, significando MARKOVIANO. Assinala que o modelo é exponencialmente distribuído;

- ***G***, significando GERAL. Indica que o modelo possui distribuições gerais. Nenhuma restrição é colocada quanto aos tipos de distribuição utilizados.

Os três campos seguintes da notação significam respectivamente o número de servidores, o comprimento máximo permitido a uma fila e o número total de usuários do modelo. O último campo é útil quando temos uma população de usuários finita e pequena, de maneira que a presença de uma parte destes usuários em um centro de serviço possa afetar (reduzir) a taxa de chegadas no sistema. Caso os dois últimos campos não sejam mencionados, implicitamente é assumido o valor ∞.

Assim, ao classificarmos um modelo como M/M/1, estamos, com isto, afirmando que tal modelo possui intervalos de tempo entre chegadas exponencialmente distribuídos, tempos de serviço exponencialmente distribuídos, um único servidor, nenhum limite quanto ao comprimento máximo da fila (tamanho máximo infinito) e um número infinito de clientes.

A complexidade dos possíveis modelos de redes de fila vai, em um crescente, dos modelos D/D/m, de pouco interesse por sua extrema simplicidade, passando pelos modelos M/M/1, bastante utilizados nas modelagem de sistemas reais por apresentarem solução analítica, até chegarmos aos modelos G/G/m, para os quais ainda não são conhecidas soluções analíticas exatas [TAN 89].

2.5 Técnicas de Simulação

Na maior parte dos modelos de simulação, a variável independente encontrada mais importante, em função da qual serão derivados os valores das demais, é o tempo. A variável independente tempo determina, destarte, como evolui e se modifica o estado interno do modelo, pois sua alteração implicará, direta ou indiretamente, em uma atualização nos valores das variáveis restantes, conduzindo o modelo a um novo estado de execução.

Aos modelos de simulação cuja variação do tempo se dá de maneira suave, sem que haja descontinuidade ou atualizações abruptas, damos o nome de **modelos de simulação contínuos**. Já àqueles cuja variação do tempo se dá de maneira discreta, ou seja, descontinuamente, ocorrendo somente em momentos predeterminados, chamamos **modelos de simulação discretos**.

É válido observar, entretanto, que em um computador digital as variáveis componentes de um modelo podem assumir somente valores discretos. Assim, alguns autores preferem classificar como sendo contínuos os modelos cuja atualização do tempo se dá de forma regular, ocorrendo sempre em intervalos espaçados por um quantidade Δt de tempo fixa; estes autores classificam como discretos os modelos cuja atualização se dá de forma irregular, ocorrendo em momentos arbitrariamente espaçados [SHA 75].

O analista, ao transpor um modelo para um computador digital, pode se valer tanto de linguagens de programação de propósito geral (C, C++, FORTRAN, *etc.*) quanto de linguagens projetadas especificamente para suporte à simulação (GPSS, SIMSCRIPT, RESQ, ModSim, DYNAMO, *etc.*).

A princípio, qualquer linguagem pode ser considerada uma **linguagem de simulação**. No entanto, atribuiremos este conceito àquelas que fornecem um conjunto de primitivas que facilitam a transposição do modelo de simulação, não exigindo por parte do analista grande esforço na codificação do modelo, ou mesmo na extração dos resultados.

2.6 Modelos de Simulação Discretos

Os modelos de simulação discretos lançam mão de um engenhoso artifício ao simular o comportamento de um sistema real. Ao conceberem um modelo para um sistema em estudo, os analistas estão preocupados em observar, de um determinado ângulo, o comportamento do sistema. O modelo contempla, pois, somente os aspectos do sistema cobertos pelo ângulo de observação desejado.

Para uma grande parte de sistemas, tais aspectos — que nada mais são, no modelo, que os atributos das entidades em observação — têm o seu estado alterado somente em momentos pontuais e esparsos da linha do tempo. Assim, por exemplo, quando se analisa um sistema composto por um caixa de banco, os únicos momentos de interesse são, a grosso modo, a chegada e a partida de um usuário. Salvo tais eventos, o sistema, segundo a perspectiva do analista, permanece inalterado.

Muitas das vezes, os estados dos aspectos de interesse se mantêm constantes a maior parte do tempo para o sistema em estudo. Seria interessante então, ao se simular tal sistema, considerar somente os instantes pontuais ao longo do tempo onde o estado global é alterado (**tempos de evento**). Deste exato pormenor tiram proveito os modelos de simulação discretos.

Nestes modelos, o funcionamento do sistema é simulado através do histórico dos eventos, como descrito na figura 2.3. O exercício do modelo se resume a um ciclo, cujas etapas são:

- Atualizar o relógio de simulação para o tempo de ocorrência do próximo evento;
- Executar a ação vinculada a este evento;

FIGURA 2.3 - Exercício de um modelo de simulação discreto

Em última instância, descrever o funcionamento de um sistema em um modelo de simulação discreto é saber discernir quais os tipos de eventos envolvidos, quais as ações executadas para estes eventos e quais os tempos de ocorrência dos mesmos.

A figura 2.4 procura exemplificar isto, mostrando como se procede a evolução dos estados do modelo ao longo do tempo. As alterações feitas no estado do modelo devido às ocorrências dos eventos *e1*, *e2 e3* e *e4* acontecem respectivamente nos tempos de evento *t1*, *t2*, *t3* e *t4*. Entre a ocorrência de dois eventos contíguos, o estado global do modelo permanece inalterado.

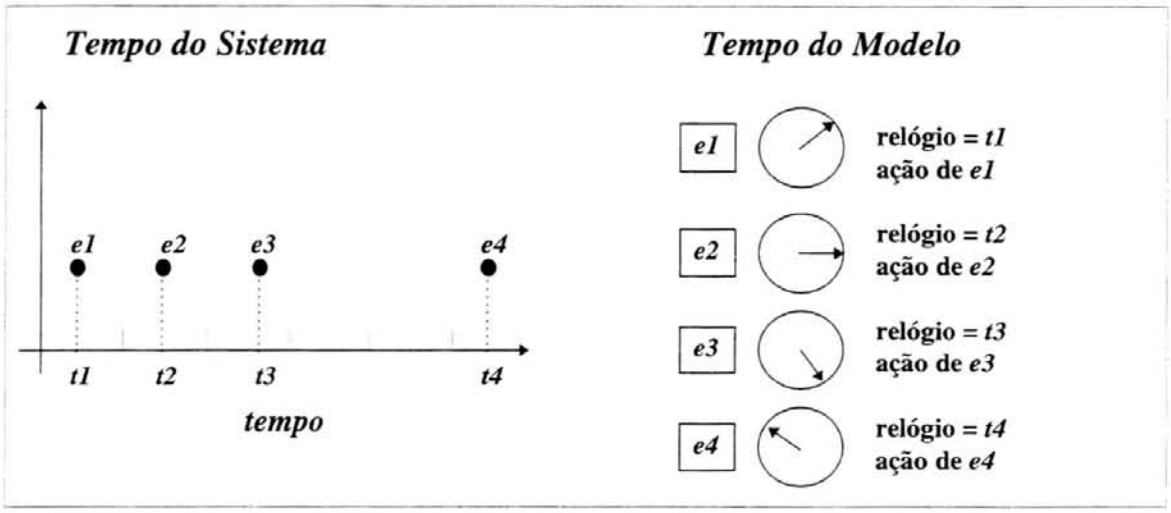


FIGURA 2.4 - Dinâmica de um modelo de simulação discreto

2.6.1 Linguagens de Simulação Discreta

As linguagens de simulação discreta fornecem ao analista uma maneira adequada para descrever o modelo. Geralmente a descrição dos atributos das entidades que compõem o modelo é feita por estruturas de dados apropriadas, não variando muito, em forma, de uma linguagem para outra. A descrição da dinâmica, no entanto, é abordada de forma diferenciada. Descrever a dinâmica de um modelo de simulação discreto é indicar [COP 91]:

- como serão representados os eventos que definem as mudanças de estado do modelo;
- como ocorrerão as comunicações entre as entidades;
- o ponto de vista a partir do qual o comportamento das entidades será descrito;

Por motivos de ordem prática — os de ater-se somente às informações imprescindíveis, detalharemos apenas o primeiro dos itens acima e, para os demais, remetemos o leitor à referência supracitada.

Uma expressão bastante utilizada na literatura de simulação é **visão de mundo**. A visão de mundo oferecida por uma linguagem, como a própria expressão indica, descreve como o projetista da linguagem concebeu os sistemas que por ela seriam modelados [SHA 75]. Ou, colocado de outra forma, como a linguagem facilita a descrição das dinâmicas desses sistemas. Para os modelos de simulação discretos, onde a dinâmica do sistema é mimetizada pela ocorrência ordenada de eventos e execução de procedimentos relacionados a estes eventos, a visão de mundo se reflete em como são representados os eventos que definem as mudanças de estado do modelo.

TABELA 2.1 - Classificação das linguagens segundo estratégia de simulação

Estratégia de Escalonamento de Eventos	*Estratégia de Monitoração de Atividades*	*Estratégia de Interação de Processos*
GASP (II, IV)	AS	GPSS
SIMSCRIPT (I.5; II; II.5)	CSL	Q-GERT
SLAM (SLAM II)	ECSL	SIMSCRIPT II.5
SIMAN	ESP	SLAM (SLAM II)
	SIMON	SIMAN
		SIMULA

Fonte: [HOO 86]

Considerando as linguagens mais conhecidas, podemos classificá-las de acordo com as três seguintes visões de mundo: **dirigida por processos**, **dirigida por atividades** e **dirigida por eventos** [SHA 75]. Cada uma das visões de mundo possui uma estratégia de simulação própria, um algoritmo para nortear o exercício do modelo. A tabela 2.1 apresenta a classificação de algumas linguagens de simulação discreta segundo a estratégia de simulação adotada por elas. Nos parágrafos seguintes, uma ligeira explicação, extraída de [COT 91] e [HOO 86], é fornecida para as visões de mundo mencionadas e suas respectivas estratégias de simulação.

Nas linguagens dirigidas por eventos, o analista especifica quais tipos de eventos o modelo possui e qual o procedimento que será executado no momento da ocorrência de um determinado tipo de evento. O procedimento executado, vinculado à ocorrência de um evento, poderá, por sua vez, escalonar a ocorrência de novos eventos em um tempo futuro; alterar o estado dos atributos das entidades do modelo; ou mesmo, cancelar a ocorrência de um evento que, antes, já fora escalonado. A estratégia de simulação para esta visão de mundo consiste em, repetidamente, selecionar, em uma lista de eventos, aquele com o menor tempo de ocorrência; avançar o relógio de simulação para o tempo em que o evento escolhido foi escalonado para ocorrer; e, executar o procedimento associado àquele tipo de evento.

Nas linguagens dirigidas por atividades, o analista também especifica os tipos de eventos do modelo, com os respectivos procedimentos associados. Entretanto, fica em aberto ao analista a possibilidade de definir **eventos contingenciais**, que ocorrem quando uma certa condição é satisfeita. Tipicamente um evento contingencial marca o começo de uma atividade, escalonando outro evento para pôr fim a esta atividade (**evento determinado**). Os termos contingencial e determinado são devidos a [NAN 81] apud [COT 91]. A estratégia de simulação para esta visão de mundo difere da anterior pelo fato de ser necessário avaliar todas as condições vinculadas aos eventos contingenciais após a execução de cada evento, a fim de determinar se um evento contingencial deve ou não ocorrer.

Nas linguagens dirigidas por processos, ao invés de o analista definir os tipos de eventos do modelo e seus procedimentos, ele especifica um conjunto de processos. Um processo pode ser pensado como uma seqüência de eventos ou de atividades. Assim como as ações associadas à ocorrência de um evento, as etapas de um processo são definidas, geralmente, através de um procedimento computacional. Entretanto, estes procedimentos também incluem instruções para suspender a execução do processo por um intervalo de tempo. O intervalo de tempo em que o processo fica suspenso pode ser explicitamente definido e conhecido — 10 minutos, 11 segundos, *y* segundos —, ou conter uma condição que, uma vez satisfeita, restauraria a execução do processo. Eventualmente o processo será reativado e a execução prosseguirá a partir de uma instrução chamada **ponto de ativação**. Os processos suspensos por um intervalo de tempo conhecido e finito são chamamos **ativos**; aqueles suspensos até o momento em que uma certa condição seja satisfeita são chamados **ociosos**.

A estratégia de simulação para a visão de mundo dirigida por processos utiliza duas listas para os processos suspensos: uma lista para os processos ativos e outra para os ociosos. A estratégia consiste em avançar o relógio para o menor tempo de ocorrência vinculado à reativação de um processo ativo; executar o procedimento

associado a este processo, até que o mesmo seja suspenso; verificar se existe algum processo ocioso cuja condição de reativação tenha se tornado verdadeira após a execução do processo anterior e, se for o caso, executá-lo; caso algum processo ocioso tenha sido reativado, é imprescindível verificar novamente a lista de processos ociosos, à busca de algum outro que, agora, tenha tido a sua condição de reativação satisfeita e, então, executá-lo. Quando não mais houver nenhum processo ocioso em condição de ser executado, o relógio de simulação é novamente avançado para o menor tempo de ocorrência vinculado à reativação de um processo ativo. É digno de nota que o relógio de simulação é avançado somente para o caso de serem reativados processos ativos; os processos ociosos não causam nenhum avanço ao tempo de simulação [KAR 91].

Uma discussão mais completa sobre as visões de mundo e suas respectivas estratégias de simulação pode ser encontrada em [HOO 86]. Neste artigo, são apresentadas, sob forma algorítmica, cada uma das estratégias mencionadas acima, bem como considerações de desempenho sobre cada uma delas.

2.6.2 Linguagens Dirigidas por Eventos

A título de exemplo, será apresentada uma implementação, em uma linguagem de simulação discreta dirigida por eventos, de um modelo M/M/1, tomado de empréstimo da teoria das redes de filas. O exemplo visa demonstrar as facilidades fornecidas por esta categoria de linguagens de simulação, bem como assinalar os passos pelos quais terá de passar o analista na transposição de seu modelo para uma linguagem dirigida por eventos.

Como existem vários pontos de similitude entre a maioria das linguagens dirigidas por eventos e o ambiente aqui proposto, este exemplo também auxilia a compreensão de idéias e conceitos que se encontram presentes no ambiente Robin Hood e que serão explorados em detalhes nos capítulos 4 e 5. A escolha de um modelo M/M/1 se deve à sua extrema simplicidade, pois não se quer, ao escolher um modelo mais complexo, obscurecer o enfoque dado às primitivas da linguagem com aspectos irrelevantes, para o momento, de modelagem ou codificação.

Podemos reconhecer, neste modelo, dois tipos de entidades: o USUÁRIO, e o SERVIDOR; e dois tipos de eventos: a CHEGADA de um usuário ao centro de serviço e a PARTIDA do mesmo após ter sido atendido. O USUÁRIO possuirá um único atributo de interesse: o tempo em que esperou pela posse do recurso; já o SERVIDOR possuirá como atributos: a fila de espera e uma variável lógica, que indica se o servidor, no momento, está ocioso ou ocupado.

A figura 2.5 demonstra como pode ser feita a declaração dos eventos de interesse e das entidades constituintes do modelo em SIMSCRIPT II.5 [RUS 90], a linguagem dirigida por eventos que adotaremos na implementação do exemplo proposto (na figura 2.5, as entidades USUÁRIO e SERVIDOR correspondem, respectivamente, a CLIENT e SERVER; e os eventos CHEGADA e PARTIDA, por sua vez, correspondem, respectivamente, a ARRIVAL e DEPART).

```
Preamble
Event notices include ARRIVAL and DEPART
   Every DEPART has a CLIENT.PTR
   Define CLIENT.PTR as a pointer variable

Temporary entities
   Every CLIENT has a WAIT.TIME
      and belongs to a WAIT.QUEUE
   Define WAIT.TIME as a real variable

Permanent entities
   Every SERVER has a STATE
      and owns a WAIT.QUEUE
   Define STATE as a integer variable
   Define WAIT.QUEUE as a fifo set
End ''Preamble
```

FIGURA 2.5 - Declaração de eventos e entidades do modelo

A ações vinculadas ao eventos CHEGADA e PARTIDA são descritas, de forma coloquial, pelas figuras 2.6 e 2.7.

- Escalone outro evento CHEGADA para ocorrer no futuro, usando uma distribuição exponencial (tempo entre chegadas);
- No momento, o servidor está ocupado?
  - **Sim:** então, coloque o usuário na fila de espera;
  - **Não:** então, escalone o fim do serviço para o usuário, agendando o evento PARTIDA para ocorrer em um tempo futuro, usando uma distribuição exponencial (tempo de serviço);

FIGURA 2.6 - Evento CHEGADA

- Imprima o tempo de espera na fila para a entidade que acaba de ser servida;
- Destrua tal entidade;
- Existem mais usuários na fila de espera?
  - **Sim:** então, escalone o fim do serviço para o próximo usuário, agendando o evento PARTIDA para ocorrer em um tempo futuro, usando uma distribuição exponencial;

FIGURA 2.7 - Evento PARTIDA

As figuras 2.8 e 2.9 traduzem as ações relativas aos eventos CHEGADA e PARTIDA da linguagem coloquial para SIMSCRIPT II.5.

```
Event ARRIVAL
   Schedule an ARRIVAL in exponential.f(MEAN, 1)
   Create a CLIENT
   If STATE(SERVER) = 0
      WAIT.TIME(CLIENT) = 0
      Schedule an DEPART giving CLIENT in exponential.f(MEAN, 1)
      STATE(SERVER) = 1
   Else
      WAIT.TIME(CLIENT) = time.v
      File this CLIENT in the WAIT.QUEUE(SERVER)
   Always
End ''ARRIVAL
```

FIGURA 2.8 - Evento CHEGADA

```
Event DEPART given CLIENT.PTR
   Define CLIENT.PTR as a pointer variable
   Print 1 line with WAIT.TIME(CLIENT.PTR) thus
      Wait Time = *.**
   Destroy CLIENT.PTR
   If STATE(SERVER) = 1
      Remove the first CLIENT.PTR from WAIT.QUEUE(SERVER)
      WAIT.TIME(CLIENT.PTR) = time.v - WAIT.TIME(CLIENT.PTR)
      Schedule an DEPART giving CLIENT.PTR in exponential.f(MEAN, 1)
      If WAIT.QUEUE(SERVER) is empty
         STATE(SERVER) = 0
      Endif
   Always
End ''DEPART
```

FIGURA 2.9 - Evento PARTIDA

O exemplo acima deixa bem claro quais são as tarefas pelas quais o analista terá de passar ao transpor o seu modelo para uma linguagem de simulação dirigida por eventos:

- reconhecer, em conformidade com os objetivos estipulados no estudo de desempenho, as entidades que constituirão o modelo;
- caracterizar adequadamente tais entidades, determinado quais os atributos que a descreverão;
- discernir quais são os tipos de eventos encontrados no sistema, para os quais uma ação apropriada terá de ser definida;

O analista não precisa se preocupar com os problemas referentes ao escalonamento e à ordenação dos eventos. É incumbência da linguagem implementar,

segundo a visão de mundo que adota, a estratégia apropriada de simulação. Resta, ao analista, coletar os dados fornecidos pela execução do programa (modelo) e elaborar, sobre eles, as considerações e análises que julgar necessárias.

2.7 Conclusão

Este capítulo procurou fornecer uma visão dos principais aspectos de análise de desempenho e modelagem relacionados diretamente com os propósitos desta dissertação, a saber, a definição e construção de um ambiente para a avaliação de políticas de balanceamento de carga. Logo, a abordagem aqui empreendida não pretendeu, de forma alguma, ser exaustiva. Ao contrário, as informações aqui contidas são somente aquelas julgadas imprescindíveis, seja por servirem de alicerce à construção do ambiente, seja por constituírem premissas necessárias à compreensão dos conceitos utilizados.

Assim, os capítulos 4 e 5, incumbidos de descreverem o ambiente Robin Hood, proposto nesta dissertação, se servirão largamente das idéias e conceitos aqui introduzidos: em tais capítulos, (*i*) um modelo para o problema de balanceamento será apresentado; (*ii*) a implementação do modelo será feita através de um simulador dirigido por eventos; (*iii*) para a sua construção, o simulador utilizará estruturas análogas às encontradas na teoria das redes de filas; (*iv*) índices de desempenho relevantes, para o caso específico do problema de balanceamento, serão fornecidos como resultado da simulação.

O capítulo seguinte explana sobre os vários modelos encontrados na literatura para o problema de balanceamento. Deste capítulo serão extraídos os subsídios que orientarão as principais decisões de projeto referentes à construção do simulador, tais como a escolha dos índices de desempenho apropriados, a definição da carga de trabalho para o modelo e, principalmente, quais entidades incluir no modelo e quais não. Enquanto que as informações contidas no capítulo que aqui se encerra pertencem a um domínio de conhecimento geral e servem de instrumento à definição e construção de modelos em vários segmentos de pesquisa, não sendo específicas a nenhum, as do próximo estão vinculadas particularmente ao problema de balanceamento e servem para explicitar, em tal segmento, as entidades essenciais à composição de um modelo, bem como aclarar os relacionamentos importantes que existem entre estas entidades.

3 Balanceamento de Carga

Este capítulo discute os principais pontos relativos à construção de um modelo para o problema de balanceamento de carga. Uma ação de balanceamento objetiva, através de transferências de cargas entre máquinas, melhorar o desempenho do sistema. Há uma real necessidade em se adotar ações de balanceamento, haja vista a constante presença, nas redes de computadores, de máquinas ociosas. A motivação ao emprego de técnicas de balanceamento em um sistema distribuído é apresentada na primeira seção deste capítulo.

A seção seguinte, 3.2, precisa o sentido dado a alguns termos utilizados ao longo do capítulo. O desempenho de um sistema distribuído, expresso quantitativamente através de índices numéricos, é considerado na seção 3.3. Nesta seção são perfilados os principais índices de desempenho encontrados na literatura. A seção subseqüente elenca alguns dos principais índices de carga encontrados. Um índice de carga é utilizado pelas decisões de balanceamento ao aferir a situação de carga em que se encontra uma máquina: ociosa, com carga média, sobrecarregada, etc. Através da comparação entre os índices de carga de diversas máquinas é que se optará, em uma ação de balanceamento, por uma delas, seja para lhe enviar, seja para dela receber uma parcela de carga.

Compreendidos os principais índices envolvidos no problema de balanceamento, as seções restantes procuram delimitar com precisão o sistema para o qual são dirigidos os esforços de balanceamento e as funcionalidades associadas aos módulos responsáveis pelas ações de redistribuição de carga. A especificação da topologia e dos padrões de carga — incumbidos de descrever o volume de carga que, ao longo do tempo, será submetido ao sistema — é feita na seção 3.6.

Os módulos de balanceamento são apresentados logo após. Uma vez que não há memória compartilhada entre as máquinas que compõem o sistema, todas as comunicações que envolvam mais de uma máquina têm de ser feitas através de trocas de mensagens. Como as decisões de balanceamento comumente pressupõem um conhecimento do estado do sistema, é necessário que tal conhecimento seja obtido através de mensagens trocadas pela rede. O Módulo de Gerência de Informação de Carga é responsável pela administração do envio e recebimento de mensagens, portadoras de informação sobre o estado do sistema. Quando tais mensagens são trocadas, sobre o que informam e entre quais máquinas esta troca é feita são perguntas respondidas pelo Módulo de Gerência de Informação de Carga. O Módulo de Tomada de Decisões é responsável, como o próprio nome indica, por decidir quando terá início a redistribuição de carga, qual carga será transferida e para qual máquina. Cada uma destas questões é tratada individualmente por subseções.

Finalmente, uma classificação bibliográfica dos trabalhos citados neste capítulo é apresentada, almejando facilitar ao leitor a elaboração de futuras consultas ou pesquisas.

3.1 Motivação

Os grandes avanços tecnológicos conseguidos nas últimas décadas na área de informática possibilitaram não somente a produção em larga escala de equipamentos com

alto grau de elaboração, como também o conseqüente barateamento dos mesmos.

Atrelado a esta maior disponibilidade de recursos computacionais está o ônus da gerência. Tipicamente, em redes de estações pessoais de trabalho, durante grande parte do tempo, muitas máquinas estão ociosas ou subutilizadas. Theimer [THE 85] relata que, em um ambiente como este, cerca de um terço do número total de máquinas fica ocioso durante o dia.

Nichols [NIC 87] estima que de 50 a 70 estações ficam ociosas em um ambiente com um número total de 350 estações. Tais máquinas ociosas, muitas das vezes, representam um poder computacional conjunto maior do que aquele que é disponibilizado, de forma isolada, ao usuário por sua estação de trabalho.

Bricker [BRI 92], além de expor o problema do desperdício de ciclos computacionais, relaciona-o aos padrões de utilização a que são submetidas, pelos usuários, as estações de trabalho em um ambiente típico de pesquisa. Como observado por este autor, a maioria dos usuários, classificados em seu artigo como pertencentes ao tipo 1, utiliza somente uma parcela do poder computacional que é disponibilizado por sua estação pessoal de trabalho, seja para ler correspondência, seja para a edição de documentos ou outras tarefas de cunho administrativo.

Ao lado destes usuários, aparecem aqueles que, apesar de exigirem para si maiores recursos computacionais, estão longe de se encontrarem em uma situação em que somente sua estação de trabalho não consiga satisfazê-los. Estes usuários, classificados como tipo 2, utilizam-se da máquina para fins de desenvolvimento de aplicativos, envolvidos, portanto, no ciclo de editoração do programa fonte, compilação, execução e depuração.

O último padrão de utilização apresentado por Bricker, tipo 3, constitui-se de usuários envolvidos em tarefas que exigem grandes quantidades de recursos computacionais, tais como lotes de simulações ou pesquisas combinatórias. Estes usuários necessitam de poder computacional maior que o disponibilizado por apenas uma estação de trabalho, pois esta dificilmente disporá de tal volume de recursos. Os usuários do tipo 1 e do tipo 2, quando não estão trabalhando, deixam completamente ociosas suas estações de trabalho, enquanto que os do tipo 3 geralmente mantêm suas máquinas ocupadas 24 horas por dia.

A partir do reconhecimento destes padrões de utilização, pode-se vislumbrar um maior aproveitamento dos recursos, que aqui faltam para uns e sobram para outros, através do balanceamento da carga computacional entre as máquinas [CHO 82] [HAC 89].

3.2 Terminologia

Um **sistema distribuído** consiste de múltiplos processadores autônomos que não compartilham memória primária, mas cooperam entre si através do envio de mensagens em uma rede de comunicação. Em concordância com o conceito exposto, citamos [BAL 89] [COU 94].

Por **balanceamento de carga**, entenda-se, doravante, o conjunto de ações que procuram redistribuir a carga computacional entre as máquinas com vistas a melhorar o desempenho de um sistema distribuído. A expressão 'balanceamento de carga' é colocada propositadamente em termos genéricos, pois não se busca aqui delimitar com precisão o conceito, nem restringir as acepções com que a expressão vem sendo usada na literatura. Isto será feito ao longo deste capítulo, quando cada um dos elementos constituintes do conceito apresentado será considerado isoladamente. Desta forma, caberá às seções seguintes tornar explícito o que se entende por: 'ações de balanceamento', 'redistribuição de carga' , 'desempenho' e 'carga de trabalho'.

3.3 Índices de Desempenho

Na literatura há um certo consenso quanto à utilização do tempo de resposta médio como o principal índice de desempenho na análise das políticas de balanceamento [EAG 86][KUN 93][ZHO 88][WAN 85][STA 85][NI 85][DIK 89][EVA 94][GOW 93]. Outros índices, no entanto, são também utilizados [BAT 95][LEM 93][LOH 96].

Entre os principais índices encontrados na literatura temos (figura 3.1):

- **Tempo de Resposta Médio**, índice de desempenho definido como sendo a média aritmética dos tempos de execução real despendidos por cada um dos processos do sistema (equação 1). Em sua contabilidade, o **tempo de execução real** considera tanto a parcela de tempo durante a qual o processo fica ocioso, esperando pela obtenção de algum recurso, quanto a parcela de tempo durante a qual o processo realiza alguma tarefa útil. Uma outra forma de se definir o tempo de execução real, equivalente à definida anteriormente, é considerá-lo como sendo o intervalo de tempo compreendido entre o momento em que o processo ingressa no sistema e o momento em que ele abandona o sistema devido ao término de sua execução.

- ***Slowdown* Médio**, índice de desempenho definido como sendo a média aritmética, para todos os processos do sistema, das razões entre o tempo de execução real e o tempo de execução sem contenção (equação 2). O **tempo de execução sem contenção** é aquele que um processo despenderia para ser executado caso não houvesse outros processos na máquina, caso não tivesse que competir pela posse da CPU.

- **Tempo de Espera Médio**, índice de desempenho definido como sendo a média aritmética dos tempos de espera incorridos por cada um dos processos do sistema (equação 3). O **tempo de espera** corresponde à parcela de tempo, na vida de um processo, durante a qual ele permanece ocioso, esperando pela obtenção da CPU a fim de prosseguir sua computação. De forma equivalente, pode-se definir o tempo de espera para um processo como a diferença entre o tempo de execução real e o tempo de execução sem contenção.

- **Taxa de Utilização Média de CPU**, índice de desempenho definido como sendo a média aritmética, para todas as CPUs do sistema, das razões entre o tempo útil despendido pela CPU e o tempo total (equação 4). O **tempo útil**

de CPU corresponde à parcela de tempo que uma CPU dedica à execução de processos. Desconsidera-se, portanto, em seu cálculo, a parcela de tempo durante a qual a CPU fica completamente ociosa. O **tempo total** corresponde ao intervalo total de mensuração.

$$TR\ Médio = \sum_{x=1}^{X} \frac{TempoExecuçãoReal_x}{X} \quad (1)$$

$$Slowdown\ Médio = \sum_{x=1}^{X} \frac{\frac{TempoExecuçãoReal_x}{TempoExecuçãoSemContenção_x}}{X} \quad (2)$$

$$TEspera\ Médio = \sum_{x=1}^{X} \frac{TempoEspera_x}{X} \quad (3)$$

$$Taxa\ de\ Utilização\ Média\ de\ CPU = \sum_{j=1}^{N} \frac{\frac{TempoÚtilCPU_j}{TempoTotal}}{N} \quad (4)$$

[1.. X]	*processos cujo início e término estejam compreendidos dentro do intervalo de mensuração*
[1.. N]	*máquinas que integram o sistema*

FIGURA 3.1 - Índices de desempenho

Usualmente, a variância relativa ao tempo de resposta médio é também apresentada como forma de validar análises feitas sobre uma política de balanceamento. O mesmo acontece para os demais índices de desempenho apresentados acima [ZHO 88].

3.4 Índices de Carga

Muitos são os índices empregados, na área de balanceamento, para expressar a carga existente, num dado momento, em uma máquina. Alguns exemplos de índices empregados são (figura 3.2):

- **Taxa de Utilização Instantânea de CPU**, índice de carga definido como sendo a razão entre o tempo útil de CPU, calculado para o intervalo compreendido entre os instantes de tempo t_0 e t_1, e o comprimento do intervalo de mensuração considerado (equação 5).

- **Tamanho da fila de processos prontos para execução;**

- **Número de processos em uma máquina**, índice de carga que representa a soma dos tamanhos das filas de todos os centros de serviço da máquina. Este

índice considera, além dos processos prontos para execução, os processos presentes nas filas de E/S e, caso a máquina implemente escalonamento por prioridades, os processos em *background*.

$$Taxa\ de\ Utilização\ Instantânea\ da\ CPU_j(t_0,t_1) = \frac{TempoÚtilCPU_j(t_0,t_1)}{t_1 - t_0} \qquad (5)$$

FIGURA 3.2 - Índices de carga

O período com o qual os índices são disseminados não precisa necessariamente coincidir com o período com o qual é feita a mensuração dos índices. Certos índices, tais como a taxa de utilização de CPU ou o tamanho da fila de processos prontos para execução, possuem, ao longo de um curto intervalo de tempo, uma grande variação. Isto faz com que sejam necessárias mensurações freqüentes, capazes de captar a abrupta variação dos índices. Como não é possível ajustar o período de disseminação ao período de mensuração, pois isto incorreria em uma intensa troca de mensagens na rede, há que se fazer com que o valor do índice disseminado seja representativo do conjunto de valores mensurados. Uma das maneiras seria disseminar a média aritmética dos valores coletados. Valores extremos, no entanto, possuem uma grande influência sobre a média aritmética, o que poderia tornar menos efetiva a representatividade do índice disseminado. Outras medidas estatísticas de tendência central que não sofressem do mesmo problema, tais como a mediana ou a moda, poderiam ser empregadas [FER 86].

De forma contrária, poder-se-ia optar pela simplicidade. Não mais seriam feitas mensurações freqüentes e a variação abrupta dos índices seria ignorada . Uma única medida feita seria representativa de todo o período de disseminação. Os perigos desta abordagem são disseminar um valor extremo, não representativo do período de mensuração, ou mesmo não contemplar a evolução do estado de carga da máquina.

$$TRPrevisto_{y,j} = TempoExecuçãoSemContenção_{y,j} + \sum_{c=1}^{C} D_{y,c} \times Q_{c,j} \qquad (6)$$

FIGURA 3.3 - Tempo de resposta esperado

Alguns autores põem em dúvida a eficácia dos índices apresentados acima na descrição do real estado de carga de uma máquina [FER 85][GOW 93].

Ferrari [FER 85] estipula, como objetivo principal das ações de balanceamento, a redução do tempo de resposta por parte dos processos. Partindo deste objetivo, o autor procura formular um índice de carga que forneça uma boa estimativa do tempo de resposta esperado para os processos. Através de equações de valor médio para redes de filas fechadas, o autor demonstra que a combinação linear dos tamanhos de filas nos centros de serviço de uma máquina é uma boa estimativa para o tempo de resposta que o processo obterá. De uma maneira simplificada, a equação 6 (figura 3.3) representa o índice proposto por Ferrari.

Em tal equação, $TempoExecuçãoSemContenção_{y,j}$ corresponde ao tempo que o processo y despenderia executando sem contenção em uma máquina j, isto é, y sendo o único processo a executar em j; $D_{y,c}$ corresponde ao tempo que processo y despenderia no centro de serviço c; $Q_{c,j}$ corresponde ao tamanho da fila no centro de serviço c, para a máquina j.

Segundo este índice, o tempo de resposta esperado para um processo y se executado em uma máquina j seria igual à soma do tempo que y despenderia executando sem contenção em j com a combinação linear dos tamanhos das filas nos vários centros de serviço de j ($D_{y,c} \times Q_{c,j}$). Os termos da combinação linear teriam como coeficientes os tempos despendidos pelo processo y em cada um dos vários centros de serviços de j ($D_{y,c}$).

É de se observar que, para o cálculo deste índice, é necessário, além dos atuais tamanhos de fila nos centros de serviços ($Q_{c,j}$), um conhecimento prévio sobre as demandas do processo em relação a estes centros de serviços ($D_{y,c}$). Ferrari argumenta que o conhecimento prévio das demandas do processo não chega a constituir um problema potencial. Os coeficientes da combinação linear (isto é, as demandas do processo) podem ser preditos através de uma análise estatística feita sobre o histórico das demandas anteriores.

Aqui, podem ser observados alguns pontos de convergência entre a concepção proposta por Ferrari para o índice de carga e a proposta por Goswami [GOS 93]: *(i)* ambos os autores procuram definir um índice de carga que forneça uma boa estimativa para o tempo de resposta esperado pelos processos; *(ii)* ambos os índices pressupõem um conhecimento prévio sobre as demandas dos processos e o tamanho das filas nos centros de serviços das máquinas.

O trabalho de Kunz [KUN 91], em concordância com [EAG 86], ressalva, no entanto, que a definição de complexos índices de carga não altera significantemente o desempenho da política de balanceamento. Índices multidimensionais, como o proposto por Ferrari, frutos da combinação linear de várias medidas de consumo de recursos, têm um desempenho similar ao dos índices unidimensionais, que consideram somente o consumo de um único recurso da máquina.

3.5 Definição do Modelo

A definição de um modelo para o problema de balanceamento de carga deve obrigatoriamente contemplar os seguintes aspectos:

- a descrição do ambiente computacional para o qual são dirigidos os esforços de balanceamento;
- a especificação do módulo de balanceamento, responsável pelas transferências de carga entre as máquinas;
- a definição do índice de carga;

- a escolha dos índices de desempenho e das medidas de relevância que nortearão a análise do modelo.

3.6 Ambiente Computacional

A descrição do ambiente computacional se inicia com a especificação da topologia com a qual se interconectam os elementos de processamento. Topologias comuns são as que prezam pela simplicidade ou regularidade matemática (figura 3.4): em malha (a, b), em anel (c), hipercúbica (d), borboleta(e), piramidal, etc. [QUI 87].

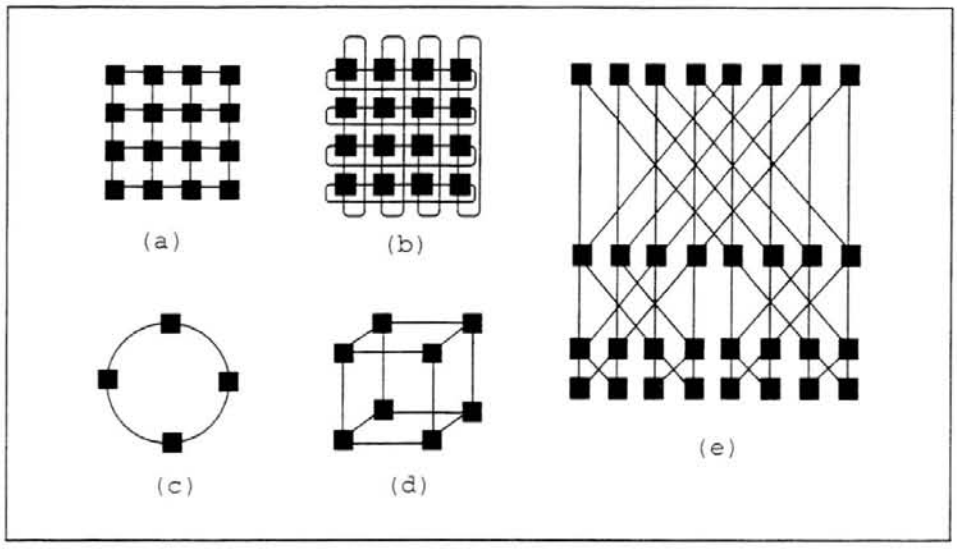


FIGURA 3.4 - Exemplos de topologias

A forma como o balanceamento em um sistema distribuído se efetua é afetada diretamente pela topologia deste. Topologias que possuem conectividade pobre ou elevado valor de diâmetro impedem uma redistribuição imediata de carga, retardando as ações de balanceamento. O reduzido número de vizinhos cerceia as opções de distribuição e obriga as cargas migrantes a um ziguezague pelo sistema à busca de nodos ociosos, degradando, de forma imediata, o tempo de resposta e inserindo, de forma mediata, uma parcela de instabilidade no sistema.

Topologias onde há tanto nodos com elevado grau de conectividade, quanto outros com apenas uma ou duas conexões, favorecem o aparecimento de pontos de contenção nos fluxos de redistribuição de carga. Nelas, os principais pontos de junção, aqueles com os maiores graus de conectividade, provavelmente sofrerão um constante influxo de cargas, atuando, ora como elemento intermediário, repassando a outrem as cargas migrantes, ora como servidor de carga, processando-as localmente quando possível. Geralmente, tais topologias apresentam, ao longo do tempo, uma homogeneidade de carga pobre, possuindo um conjunto específico de nodos sintomaticamente sobrecarregados.

Relações entre o grau de conectividade médio, ou mesmo entre a distância topológica média, e um índice de desempenho escolhido podem ser traçadas, como feito em [LOH 96]. Considerações sobre estabilidade podem ser extraídas de [CAS 88].

Todas as informações que trafegam pela rede de interconexão, sejam mensagens, sejam cargas migrantes, estão sujeitas à latência do meio. A forma como tal latência é compreendida pelo modelo depende, no entanto, do grau de detalhamento dado a esta particularidade. Há modelos que adotam uma visão simplificadora e consideram sempre constante o atraso na entrega das informações, a despeito do conteúdo trafegado. Outros consideram o atraso na entrega como sendo uma variável aleatória [NI 85]. Por fim, há aqueles que calculam o tempo de atraso como uma função do volume de dados [STA 85][BAT 95].

Alguns autores argumentam que, em sistemas reais, há largura de banda excedente nos enlaces e que os atrasos na comunicação são devidos mais aos esforços despendidos no processamento dos protocolos de comunicação que ao uso do enlace. No modelo de balanceamento que apresentam, os atrasos de comunicação são encampados por custos fixos de processamento tanto no lado do remetente, quanto no lado do destinatário dos dados [ZHO 88] [EAG 86] [WAN 85].

Há ainda a possibilidade de se considerar, isoladamente, as propriedades de cada enlace. Este modelo representaria com maior fidelidade sistemas em que os meios de comunicação que constituem a rede de interconexão possuem propriedades discrepantes. A ligação por um enlace de baixa velocidade entre LANs geograficamente afastadas seria um bom exemplo. Fica a cargo do analista julgar quais detalhes são prescindíveis de serem contemplados pelo modelo e quais não o são.

Na descrição do ambiente computacional, a especificação da topologia responde somente pela discriminação dos componentes que integram materialmente o sistema. Resta ainda, para que a descrição esteja completa, determinar como o sistema evoluirá ao longo do tempo.

Para o problema de balanceamento, a evolução temporal, isto é, a seqüência de ações executadas pelos componentes do sistema, é desencadeada por cargas que ingressam no sistema e demandam por recursos. Determinar como o sistema evolui, neste caso, equivale a especificar a **chegada de cargas** e as **demandas** associadas a estas cargas. A descrição completa do ambiente computacional termina com o conhecimento dos padrões de chegada e de demanda apresentados pelas cargas.

Em alguns modelos, a demanda associada à carga se resume ao tempo de processamento requerido [EAG86][SON 94]. Especificar o padrão de demanda para uma carga que ingressa no sistema, em tais modelos, significa atribuir à carga o consumo de uma parcela de processamento. Em outros modelos, a noção de demanda é enriquecida e, além do processamento, o consumo de outros recursos é considerado, tais como operações de E/S [ZHO 88], memória [BAT 95], comunicação entre os processos [HEI 93].

O detalhamento da noção de demanda denuncia a importância dada pelo analista, na construção do modelo, às particularidades do sistema real. Modelos que não consideram a comunicação entre processos como elemento constituinte da noção de demanda supõem, implícita ou explicitamente, que a omissão deste ponto não invalida os resultados obtidos.

Com freqüência, na literatura, os padrões de chegada e de demanda são representados por processos estocásticos [STA 85][WAN 85][NI 85][EAG 86] [CAS 88]. Esta abordagem permite, para os estudos que empregam técnicas analíticas, manter sob controle a tratabilidade matemática do modelo, e para os que empregam técnicas de simulação, coadunar simplicidade com um esperado grau de fidedignidade na descrição de tais padrões. Em ambos os casos, de forma tácita, está a suposição de que as distribuições estatísticas escolhidas na descrição dos padrões não são discrepantes o suficiente do real a ponto de comprometerem os resultados.

Outros autores extraem da observação da dinâmica de um sistema real as informações necessárias à descrição dos padrões de chegada e de demanda [ZHO 88][GOW 93] [BAT 94][BAT 95][LOH 96]. Uma máquina real é submetida a diversas sessões de mensuração, extraindo-se os tempos de chegada e de término dos processos compreendidos no intervalo de observação, e armazenando tais registros em um **arquivo de rastro**. O comprimento da sessão de mensuração é de suma importância a fim de reduzir os **efeitos de borda**, causados por processos que tiveram seu início antes do começo da sessão, ou tiveram seu fim após o término da mesma. Intervalos maiores reduzem tal interferência, mas envolvem a extração de um grande volume de dados.

Em [ZHO 88], foram elaboradas 49 sessões de mensuração com a duração de 4hs cada, totalizando 196hs dedicadas à observação de uma máquina VAX-11/780 rodando UNIX 4.3BSD. Cada sessão de mensuração resulta em um arquivo de rastro composto de registros iguais ao abaixo:

< *tempo de chegada*, *demanda de CPU*, *número de operações de E/S* >

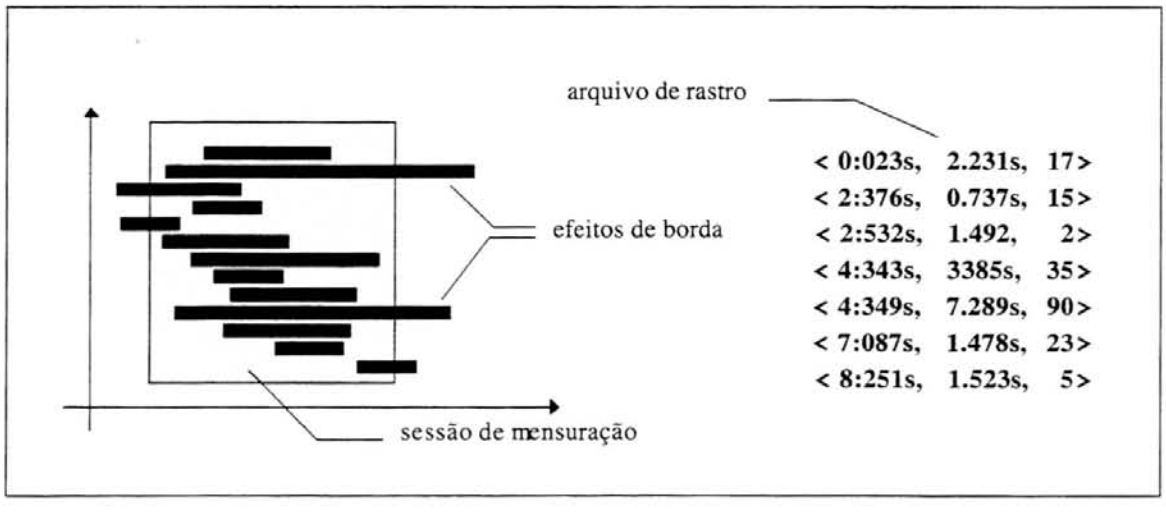


FIGURA 3.5 - Sessão de mensuração e confecção de arquivo de rastro

A figura 3.5 exemplifica a construção de um arquivo de rastro. Durante um intervalo de mensuração determinado, são armazenados, para os processos cujo início e término estão compreendidos dentro dos limites do intervalo, registros com o formato similar ao apresentado acima.

Este arquivo de rastro alimentará os padrões de futuras simulações, emprestando a elas a dinâmica roubada do sistema real. A utilização de arquivos de rastro permite a elaboração de simulações que mimetizam o comportamento de um sistema real com um grau de fidedignidade bem superior ao obtido com a utilização de processos estocásticos.

Em [BAT 95], Harchol-Balter advoga a importância do conhecimento prévio da distribuição de freqüências dos padrões de demanda. Segundo ela, as decisões tomadas pela maior parte das políticas de balanceamento apresentadas na literatura se deixam guiar por uma presunção implícita de que os **tempos de vida** dos processos são exponencialmente distribuídos. Se a premissa se torna inválida, e temos os tempos de vida dos processos descritos com mais apuro por uma outra distribuição de freqüência, muito provavelmente as decisões de balanceamento perderão parte de sua eficácia.

A partir de dados extraídos de um estudo feito sobre os tempos de vida de processos em máquinas UNIX, Harchol-Balter conclui que os tempos de vida não seguem uma distribuição exponencial. A autora, então, apresenta novos critérios de escolha para as ações de balanceamento. Estes critérios tiram proveito do conhecimento prévio da nova distribuição de freqüência encontrada para a demanda.

O trabalho de Harchol-Balter vem realçar a importância que dever ser dada a uma apurada descrição dos padrões de carga do sistema, e colocar sob suspeição trabalhos desatentos a esta questão. Se, num primeiro momento, é suficiente que se adote distribuições estatísticas conhecidas na validação de políticas de balanceamento, fica ainda, para uma análise necessária e posterior, saber se tal distribuição realmente se adequa, e, na eventual substituição forçosa da distribuição adotada por uma outra mais apropriada, o que aconteceria à eficácia das políticas?

3.7 Módulo de Balanceamento

O módulo de balanceamento é, no modelo, a entidade responsável tanto pelas decisões quanto pelas ações de balanceamento. Funcionalmente, o módulo de balanceamento pode ser dividido em duas partes [MIL 94]:

- **Módulo de Gerenciamento de Informação**
- **Módulo de Tomada de Decisões**

3.7.1 Módulo de Gerência de Informação

As decisões de balanceamento geralmente tiram partido de informações coletadas sobre o estado das máquinas do sistema. Pelo fato de não se ter uma memória compartilhada comum, é necessário que as informações sejam trocadas através de mensagens enviadas pela rede de interconexão. É papel do Módulo de Gerência de Informação coletar e disseminar as informações necessárias ao Módulo de Tomada de Decisões. Na literatura, o Módulo de Gerenciamento de Informação é referido também como **Política de Informação** [EVA 94][ZHO 88].

Ao desempenhar o seu papel, o Módulo de Gerência de Informação deve procurar responder satisfatoriamente as seguintes questões [MIL 94]: (*i*) qual informação gerenciar; (*ii*) como essa informação é representada e possivelmente medida; (*iii*) a quem transmitir esta informação; (*iv*) quando o Módulo de Gerência de Informação deve ser ativado; (*v*) quando iniciar a disseminação da informação. As subseções seguintes abordarão individualmente cada uma destas questões.

3.7.1.1 Qual informação gerenciar?

Usualmente, as mensagens trocadas levam consigo informações sobre o poder de processamento, mas poderiam conter também informações adicionais sobre dispositivos de E/S, comunicação ou outros recursos.

3.7.1.2 Como essa informação é representada e possivelmente medida?

A carga de processamento é representada por um **índice de carga**. Como visto na seção 3.4, várias propostas foram feitas procurando se definir um índice de carga adequado. Entre os propostos se têm: a taxa de utilização da CPU, o comprimento da fila de processos prontos para execução, o número total de processos, o número de processos em *background*, o volume de paginação e interrupções.

3.7.1.3 A quem transmitir as informações?

A resposta a esta pergunta determina como será organizado o fluxo de informação pelo sistema. Há várias formas de organização possíveis. Em uma organização **local**, a uma máquina é dado saber somente sobre o estado de seus vizinhos imediatos, ou apenas de uma parcela destes. Em uma organização **global**, cada máquina possui a visão de todo o sistema. Neste caso, para topologias onde a difusão (*broadcast*) de dados não pode ser feita de maneira direta, é necessário que se implemente algum esquema de repasse de informação que garanta a todas as máquinas a manutenção da visão global. Vale frisar que nem sempre o controle das informações é distribuído uniformemente entre as máquinas. Organizações há em que o controle da informação fica restrito a apenas algumas máquinas do sistema, especialmente designadas.

O controle da informação por parte das máquinas está intimamente relacionado com a noção de **centro de decisão** de balanceamento. Nem sempre é facultada a todas as máquinas do sistema autonomia suficiente para, de per si, tomarem decisões de balanceamento. A autonomia para tomar decisões surge de forma concomitante com a responsabilidade pelo controle de informações acerca de um domínio de balanceamento. Somente máquinas detentoras de informação, salvo raras propostas tais como, por exemplo, a escolha aleatória da fonte ou destino das cargas [ZHO 87][EAG 86], possuem subsídios para elaborar uma decisão de balanceamento.

A disseminação de informação pode se restringir a um âmbito local, onde a uma máquina é dado saber somente sobre o estado das máquinas vizinhas. Esta restrição possui a virtude de reduzir a latência em que a entrega das informações incorre, afinal as trocas são feitas somente entre máquinas topologicamente adjacentes, mas, em contrapartida, deteriora a qualidade das decisões, pois limita o universo de máquinas

candidatas a participarem das ações de balanceamento. Ocasiões em que as informações utilizadas por uma decisão de balanceamento são inconsistentes, porque as atualizações enviadas ainda trafegam pela rede de interconexão, são evitadas quando se reduz a latência na entrega. Valores reduzidos de latência, conforme detalhado no parágrafo seguinte, obstam o aparecimento de situações de instabilidade.

De forma contrária, poder-se-ia optar por manter uma visão global, onde a cada uma das máquinas seria dado saber sobre o estado de todas as demais. Conforme dito acima, para topologias onde a difusão de dados não pode ser feita de maneira direta, mecanismos de repasse de informação teriam de ser implementados. Problemas de instabilidade podem surgir devido à lentidão com que o repasse é feito. A evolução, de ociosa para sobrecarregada, do estado de uma máquina pode não ser constatada por uma outra, topologicamente distante, que a toma, então, como alvo, migrando-lhe uma parcela de carga não desejada, o que viria lhe acentuar ainda mais o estado de sobrecarga. Se tal máquina decide por rejeitar a parcela de carga que lhe foi cedida, passando-a adiante para uma terceira supostamente ociosa, abre-se a possibilidade de, por motivos de inconsistência de informação, se ter carga vagantes indefinidamente rejeitadas. Em situações limites, referidas na literatura como **sucateamento de processamento** [EAG 86][NI 85], o ônus de receber, processar e passar as cargas vagantes a frente pode chegar a consumir por completo o processamento das máquinas, levando a uma extrema degradação do desempenho, onde aos processos não é permitido evoluir em sua computação, pois todo o poder de processamento está votado às decisões de balanceamento.

A forma como se efetua a distribuição de informação não precisa necessariamente ficar atada à topologia física do sistema. Organizações há onde, sobre a topologia física do sistema, é definida uma nova estrutura de distribuição de informação [EVA 94][LEM 93][ZHO 88][MUN 95]. A topologia física é retalhada em **domínios de balanceamento**, partições que limitam a abrangência da distribuição de informação. Relações de hierarquia podem ser estabelecidas entre os domínios e nada impede que estes venham a se sobrepor.

A noção de domínio é usualmente utilizada por organizações onde o controle das informações não é distribuído uniformemente entre as máquinas, mas se concentra em apenas algumas [LEM 93]. Domínios de balanceamento são estabelecidos, ficando a cargo de uma das máquinas do domínio, conhecida como **gerente**, o controle das informações das demais máquinas pertencentes ao domínio. As informações referentes às máquinas de um domínio estão concentradas e disponíveis somente na máquina gerente, responsável por aquele domínio. Todas as demais devem prestar contas ao gerente de seu domínio, sendo-lhes vedado, no entanto, o acesso à informação das demais máquinas integrantes do domínio.

Na figura 3.6, extraída de [LEM 93], a topologia hipercúbica é particionada em vários domínios de balanceamento, hierarquicamente relacionados. Os fluxos de distribuição de informação tomam a forma de árvore binária. Partindo das folhas, as informações passam por nodos intermediários até atingir a raiz. Nesta organização, os nodos {6,7} prestam contas ao gerente de seu domínio, o nodo 6. Este, por sua vez, por pertencer também ao domínio {6,4}, cujo gerente é o nodo 4, deve a ele enviar ou as informações que recebera, ou um resumo destas. Os centros de decisão, representados

aqui pelos gerentes, efetuam o balanceamento dos nodos pertencentes a domínios localizados em níveis inferiores da árvore. Maiores detalhes podem se encontrados na referência indicada.

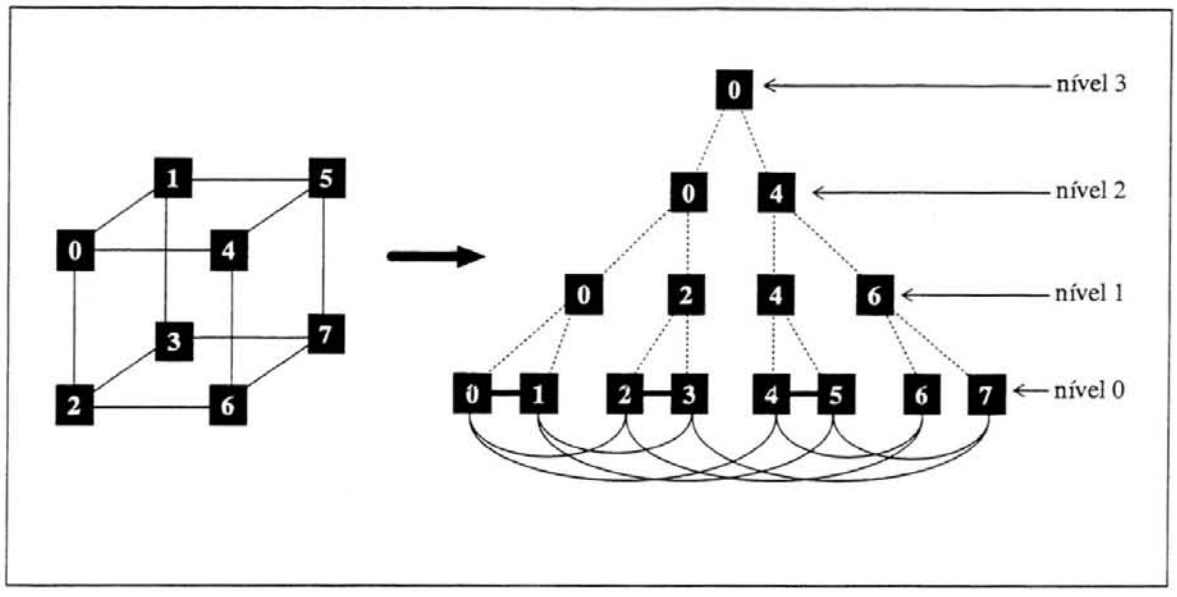


FIGURA 3.6 - Balanceamento hierárquico em topologia hipercúbica

Em topologias que possuem um grande número de nodos, é praticamente impossível coadunar a manutenção de uma visão global e baixos valores para a latência na entrega da informação. Isto decorre do fato de ser a latência máxima proporcional ao diâmetro topológico. Valores elevados de latência favorecem a tomada de decisões equivocadas. A constatação disto depõe a favor das organizações que, ao particionar em domínios a topologia física, reduzem o caminho a ser percorrido pela informação, e evitam que a latência atinja valores intoleráveis [EVA 94]. Políticas de balanceamento que se valem de tal organização de distribuição de informação são ditas escaláveis, pois suportam mais facilmente o acréscimo de nodos ao sistema.

Um caso extremo, e por isso digno de nota, seria o controle da informação de todo o sistema por parte de uma única máquina. Esta organização, referida doravante como **centralizada**, possui o inconveniente de atribuir a uma única máquina todo o ônus de gerência de informação do sistema. Para certas topologias, torna-se desencorajador, ou mesmo proibitivo, utilizar a organização centralizada, visto possuírem um grande diâmetro topológico e não disporem de uma forma facilitada de difusão da informação.

Isto não acontece para topologias onde a comunicação entre os nodos se dá por um meio de comunicação compartilhado, e a difusão de informação para o sistema, como um todo, pode ser feita de maneira direta. A conexão entre nodos feita por um barramento compartilhado ou por um anel compartilhado são bons exemplos desta categoria.

Considere uma topologia constituída por um único barramento compartilhado. Seja M_{ENV} e M_{REC} os custos de tempo incorridos pela CPU para, respectivamente, enviar e receber atualizações relativas ao estado das máquinas; N o número de máquinas do sistema; e P o período de tempo empregado na disseminação das atualizações. Para

tal topologia, a organização centralizada imputará, em termos de percentagem de tempo de CPU gasto com trocas de informação, o custo C_{ESP} (equação 7) à máquina especialmente designada para receber, de todas as outras, as atualizações, e o custo C_{NORMAL} (equação 8) às demais. Exceto para a máquina incumbida de centralizar as informações, os custos relativos às atualizações **independem** do tamanho N do sistema.

$$C_{ESP} = \frac{(N-1) \times M_{REC}}{P} \qquad (7)$$

$$C_{NORMAL} = \frac{M_{ENV}}{P} \qquad (8)$$

$$C_{DIST} = \frac{(N-1) \times M_{REC} + M_{ENV}}{P} \qquad (9)$$

FIGURA 3.7 - Custos de processamento incorridos na disseminação de informação

Em contraste, para organizações puramente distribuídas, o custo para todas as máquinas é igual a C_{DIST} (equação 9), pois, a cada intervalo P, cada máquina terá de processar as atualizações difundidas por cada uma das demais.

O trabalho de [ZHO 88] matiza e explora esta questão. Além da organização centralizada, é apresentada uma variante desta, chamada pelo autor, no trabalho, de global. Nesta variante, todos as máquinas enviam sua atualização para uma única máquina, mas, ao contrário da organização centralizada, dela recebem, de volta, o agregado de todas as informações do sistema. A cada nodo é permitido ter acesso às informações de todos os demais. A visão global que possuem não resulta, no entanto, de trocas feitas diretamente entre os nodos, mas sim intermediadas por uma máquina central. O autor conclui, através do exercício de várias simulações, que, para uma topologia em barra: (*i*) apesar dos enormes esforços despendidos por uma única máquina, seja como elemento intermediário no repasse de informação (global) , seja como elemento final de todas as informações do sistema (centralizada), tais organizações possuem um desempenho satisfatório, muitas das vezes, superior aos apresentados por organizações distribuídas; (*ii*) disseminações periódicas impõem um custo menor às organizações centralizadas do que às organizações distribuídas.

3.7.1.4 Quando o Módulo de Gerência de Informação deve ser ativado?

A ativação pode acontecer periodicamente, ou ser vinculada à ocorrência de algum evento. Para o primeiro caso é necessário se definir um período P de ativação do módulo; para o segundo, os eventos que levam à ativação são geralmente: a chegada, partida ou transferência de carga.

3.7.1.5 Quando iniciar a disseminação de informação? Qual a freqüência de disseminação adequada?

A disseminação **periódica** de informação possui o inconveniente de, se a freqüência de distribuição da informação for muito elevada, saturar os canais de comunicação ou provocar custos de processamento inaceitáveis. Se, por outro lado, se dilata o período com o qual as informações são trocadas, corre-se o risco de se tomarem decisões com informações desatualizadas. Outro problema surge quando os estados das máquinas não variam muito ao longo do tempo, não havendo, então, porque enviar informação de carga, pois não há nada a se atualizar, e todo o esforço de trocas periódicas de informação acaba por se configurar um desperdício.

Uma forma de contornar os transtornos causados pela disseminação periódica de informação seria a definição de **estados de carga** para as máquinas. Assim, dependendo do consumo interno de seus recursos, uma máquina se configuraria como BAIXA-CARGA, MÉDIA-CARGA ou ALTA-CARGA. A disseminação de informação ocorreria agora somente quando da alteração do estado de carga da máquina. Com isto, evitar-se-iam trocas desnecessárias de informação e, ainda assim, manter-se-ia uma visão consistente do sistema por parte das máquinas.

No entanto, políticas que procuram constantemente manter o índice de carga das máquinas dentro de um estreito limite em torno do índice médio do sistema (ou dos vizinhos) não são compatíveis com a definição de estados elementares de carga. Procuram explorar ganhos marginais de paralelismo e, para isto, exigem um maior refinamento na definição dos estados de carga. A fina acurácia exigida por tais decisões de balanceamento implica que pequenas variações nos índices de carga de uma máquina devem ser notificadas às demais.

Um intervalo constante de carga, ΔL, pode ser, então, estipulado. Caso a variação no índice de carga de uma máquina exceda a ΔL, o processo de disseminação de informação terá início. Esta estratégia faz com que a acurácia das decisões esteja limitada por uma margem fixa de erro (erro absoluto) de $\pm\Delta L$. Se se reduz o valor de ΔL, têm-se informações mais acuradas, mas às custas de um aumento na freqüência de disseminação. O erro relativo, a que está sujeita a visão que uma máquina possui do índice de outra, entretanto, não é fixo, e depende do valor real do índice. Se o valor real do índice é L_p, a visão estará sujeita a um erro relativo de $\Delta L/L_p$. Enquanto L_p for grande, a percentagem de erro a que está sujeita a visão é pequena, mas, quando se diminui L_p, a percentagem de erro aumenta.

O intervalo de carga que determinará o início do processo de disseminação não precisa ser necessariamente constante [LEM 93]. Este intervalo pode ser calculado como uma função do índice de carga. Seja definida a variável u, conhecida como **fator de atualização de carga**, com valores situados na faixa **]0..1[**, de tal forma que, uma máquina enviará sua atualização sempre que seu índice de carga se elevar a $(1/u)L$, ou se reduzir a uL. Se $u=1/2$, uma máquina enviará sua atualização sempre que seu índice de carga, em relação à última disseminação de informação que tomara parte, cair pela metade, ou tiver o seu valor dobrado. Caso se tenha $u=4/5$, bastará que a carga ultrapasse os limites superior ou inferior, cujos valores respectivamente representam $(4/5)L$ e $(5/4)L$. Quanto mais próximo de 1 for o fator de atualização, mais estreita será

a faixa compreendida entre os limites inferior e superior e, por conseguinte, maior será a freqüência com que as disseminações serão feitas. Considere, a título de exemplo, $u=1-\delta$. Os limites superior e inferior ficarão respectivamente cotados a $(1-\delta)L$ e $L/(1-\delta)$, e a faixa que se estende de um a outro terá a extensão aproximada, para valores de δ próximos a zero, de $2\delta L$. À medida que u se aproxima de 1 ($\delta \rightarrow 0$), os limites inferior e superior convergem para L, e o espaço que os separa tende a zero ($2\delta L \rightarrow 0$).

Uma conseqüência da adoção de tal estratégia é que a **percentagem de erro** (erro relativo) na informação de carga manter-se-á constante. A margem de erro máxima a que está sujeita a informação de carga (erro absoluto) será, no entanto, variável e igual a $(1/u)L_p$. A freqüência com que são enviadas as atualizações crescerá à medida que o índice de carga diminuir e a possibilidade de a máquina se tornar ociosa aumentar. Isto faz com que os custos relativos à disseminação da informação sejam diluídos pela capacidade computacional ociosa. A escolha de um valor adequado para o fator de atualização de carga representa uma compromisso entre a qualidade de informação de carga e os custos para atingi-la.

Uma opção alternativa às apresentadas acima seria efetuar a disseminação de informação **sob demanda**. De forma simétrica, a requisição de informação poderia partir tanto de uma máquina sobrecarregada, à procura de outra que acolhesse parte de sua carga, quanto de uma máquina subcarregada, ansiosa por encontrar uma outra que algo lhe possa lhe fornecer, ou mesmo de ambas. A disseminação de informação dar-se-ia, neste caso, somente quando estritamente necessária, reduzindo o número de trocas a um patamar mínimo. As decisões de balanceamento, no entanto, teriam de aguardar pela informação, injetando um atraso imediato nos tempos de resposta dos processos. Isto não ocorre para os casos anteriores pois, devido a disseminações intermediárias, periódicas ou não, a informação utilizada na decisão de balanceamento já se encontra disponível.

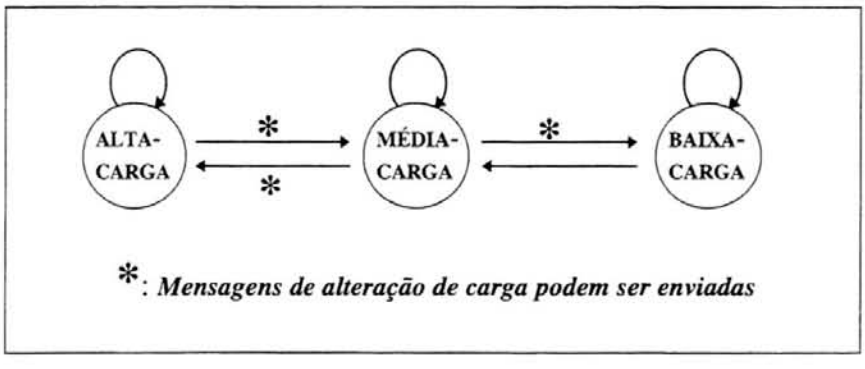


FIGURA 3.8 - Diagrama de transição de estados de carga [NI 85]

Requintes podem ser adicionados às opções sugeridas acima, como feito por [NI 85]. Este trabalho adota a disseminação de informações quando da mudança de estado de carga em uma máquina. No entanto, nem todas as transições de estado de carga disparam a disseminação de informação. Somente transições imprescindíveis, segundo a lógica interna da política de balanceamento definida no trabalho, são capazes de iniciar a disseminação de informação. Na figura 3.8 é mostrado o grafo de transições de estado usado.

3.7.2 Módulo de Tomada de Decisões

O Módulo de Tomada de Decisões constitui a essência do processo de redistribuição de carga. As virtudes mais nobres ou os piores defeitos do balanceamento de carga, terão nele, provavelmente, raiz. Norteado, geralmente, pelas informações fornecidas pelo Módulo de Gerência de Informação, ele é responsável por todas as decisões de balanceamento. A figura 3.9 explicita o relacionamento existente entre o Módulo de Tomada de Decisões e o Módulo de Gerência de Informação.

As decisões de balanceamento objetivam a melhoria do desempenho do sistema. Nem sempre, no entanto, é deixado claro pelos autores qual índice de desempenho se procura melhorar. Ferrari [FER 87] aborda esta questão, ressaltando a importância de se ter sempre em mente objetivos claros, pois será dos objetivos que os analistas, inicialmente, partirão na definição dos critérios a serem adotados pelas decisões de balanceamento, e também, até mesmo, na definição de um índice de carga adequado.

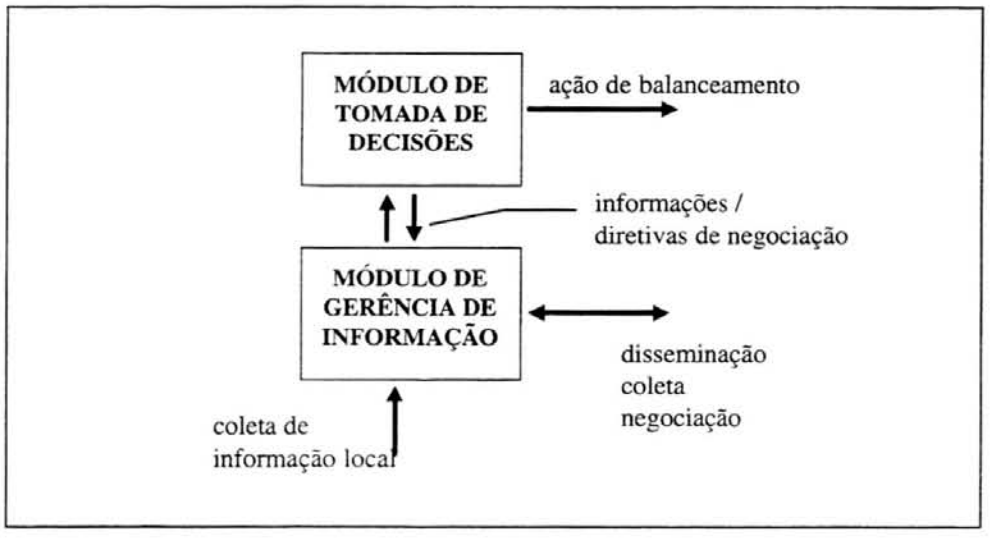


FIGURA 3.9 - Relacionamento entre Módulo de Tomada de Decisões e Módulo de Gerência de Informação

Uma decisão de balanceamento é a resposta dada pelo Módulo às seguintes questões: **quando** iniciar uma ação de balanceamento, **qual** carga transferir e **para onde**. Habitualmente, as respostas individuais a cada uma destas questões são referidas, com pequenas discordâncias, na literatura como **política de ativação**, **política de transferência** e **política de localização**, respectivamente [MIL 94][EAG 86][ZHO 87] [ZHO 88][SON 94].

3.7.2.1 Política de Ativação

O processo de balanceamento pode ser desencadeado por uma máquina que se julga ociosa (**iniciada pelo receptor**), com vistas a tomar para si uma parcela de carga, ou, de forma contrária, por uma máquina que julga sobrecarregada (**iniciada pelo emissor**) e procura, portanto, ver-se livre de parte de sua carga, ou até mesmo por ambas [LEM 93] [LOH 96][DIK 89]. Outros autores preferem a expressão 'iniciada pela servidor' à 'iniciada pelo receptor' e a expressão 'iniciada pela fonte' à 'iniciada pelo

emissor' [ZHO 88][WAN 85]. Para políticas que centralizam todas as informações em uma única máquina, a divisão precedente não faz sentido, sendo desencadeado o processo de balanceamento somente pela máquina especialmente designada.

TABELA 3.1 - Condição de disparo para o balanceamento da carga (critério de limiar)

	Condição de disparo
iniciada pelo emissor	$L > V_{max}$
iniciada pelo receptor	$L < V_{min}$

Dois são os critérios apresentados pela literatura para se iniciar o balanceamento, aos quais no referiremos como **critério de limiar** e **critério de diferença**. De acordo com o primeiro deles, o balanceamento tem início quando uma máquina percebe que sua carga local L, representada por um índice de carga, ultrapassou um determinado valor de limiar máximo V_{max} (iniciada pelo emissor). O mesmo pode ocorrer, por analogia, quando uma máquina percebe que seu índice de carga está abaixo de um valor de limiar mínimo V_{min} (iniciada pelo receptor). Ambas as situações são contempladas pela tabela 3.1. Aqui, a condição de disparo da ação do processo de balanceamento, como pode ser notado, não depende de informações outras que aquelas disponíveis localmente para a máquina.

TABELA 3.2 - Condição de disparo para o balanceamento da carga (critério de diferença)

	Condição de disparo
iniciada pelo emissor	$L - L_j > \Delta L$ $L - f(L, L_j, ..., L_k) > \Delta L$
iniciada pelo receptor	$L_j - L > \Delta L$ $f(L, L_j, ..., L_k) - L > \Delta L$

De acordo com o segundo critério, o balanceamento tem início quando uma máquina detecta uma diferença de carga ΔL existente entre si e uma outra máquina suficientemente grande para dar início ao processo de redistribuição de carga. Alternativamente, a diferença pode também ser constatada ao se contrapor a carga local com o resultado de uma função aplicada às cargas das máquinas do domínio de balanceamento. Um bom exemplo disto seria a contraposição do valor local da carga com a média das cargas dos vizinhos imediatos, média esta que considera em seu cálculo inclusive o valor local da carga [LEM 93].

A diferença de carga observada pode ser tanto **positiva** — significando que a máquina possui carga excedente e as ações de redistribuição partem, portanto, do emissor —, quanto **negativa** — indicando que a máquina possui pouca carga, sendo o receptor o responsável por desencadear o processo de balanceamento. A tabela 3.2 apresenta ambas as situações. Ao contrário da condição de disparo vista no parágrafo precedente, esta exige, para sua detecção, a posse de informações sobre o estado do sistema.

Alguns autores apontam diferenças entre os conceitos de balanceamento de carga e **compartilhamento de carga**. Segundo eles, as políticas de balanceamento têm por objetivo manter as cargas das máquinas dentro de uma estreita faixa em torno da média instantânea de cargas do sistema, ao passo que as políticas de compartilhamento se restringem a evitar a utilização excessiva ou a ociosidade das máquinas [EAG 86][SON 94][FER 95]. Cumpre observar que, na verdade, as diferenças entre os conceitos residem nos critérios adotados para o disparo das ações de redistribuição de carga: políticas de balanceamento empregam o critério de diferença, enquanto que políticas de compartilhamento empregam o critério de limiar. Neste texto, tal distinção é desconsiderada, sendo empregada a expressão 'balanceamento de carga' para ambas as acepções. Outros autores agem de forma similar ([ZHO 88]).

Satisfeita a condição de disparo, o processo de balanceamento segue seu curso normal. Os próximos passos a serem dados são: determinar quais cargas serão recebidas (ou enviadas) e encontrar uma máquina fonte (ou alvo) disposta.

3.7.2.2 Política de Transferência

Muitas estudos passam por cima da questão de se determinar quais cargas serão efetivamente transferidas por uma ação de balanceamento. Tais estudos ou consideram os tempos de processamento de todas as cargas iguais [LEM 93][SON 94], não havendo, então, porque optar por uma carga específica dentre as demais, ou partem da premissa de que não é permitido transferir cargas que já tenham iniciado sua execução [ZHO 87][EAG 86], sendo forçosa e necessária, então, a escolha, para transferência, de uma carga que acabara de chegar e cuja execução não tenha ainda iniciado.

Os autores se referem como **políticas de posicionamento** àquelas que não permitem a transferência de cargas cujo início já tenha ocorrido, isto é, cargas que já tenham recebido da CPU uma parcela (ou *quantum*) de tempo. De forma contrária, os autores se referem às políticas que não impõem restrições aos critérios de escolha, sendo permitido migrar uma carga qualquer dentre as locais, mesmo as que já tenham sido executadas, como **políticas de migração**. Em concordância com tal nomenclatura citamos [ZHO 87][MIL 94].

Outros estudos, uma vez percebida a necessidade de se efetuar uma ação de balanceamento, escolhem, criteriosamente, dentre as cargas locais, uma específica para ser transferida. A fim de exemplificar os critérios adotados na escolha da carga a ser transferida, exporemos os usados por [BAT 95].

A heurística utilizada, por este trabalho, na escolha da carga é de selecionar, entre as cargas locais, a que possuir a maior probabilidade de viver uma parcela de

tempo superior à que foi gasta para migrá-la. As razões fornecidas pelos autores são duas. Do ponto de vista da máquina, uma considerável parcela de tempo foi gasta na migração da carga, sendo sensato, portanto, escolher, para migrar, uma carga cujos custos de execução figurassem ser provavelmente maiores do que os custos de migração. Do ponto de vista da carga, o tempo despendido na migração possui uma grande impacto no tempo de resposta, sendo justificável migrar somente cargas cujo tempo de migração pudesse ser amortizado por um longo tempo de vida.

$$\Pr\{\ tempo\ de\ vida > \mathbf{T}\ segs.\ |\ idade > 1\ seg.\} = 1/\mathbf{T} \qquad (10)$$

$$\Pr\{\ tempo\ de\ vida > \mathbf{a}\ segs.\ |\ idade = \mathbf{b}\ segs. > 1\ seg.\} = \mathbf{b}/\mathbf{a} \qquad (11)$$

$$IdadeMínimaMigração_{y,j} = \frac{CustosDeMigração_{y}}{L - L_j} \qquad (12)$$

FIGURA 3.10 - Probabilidade condicional de tempo de vida e idade mínima de migração [BAT 94]

Um estudo realizado pelo mesmo autor [BAT 94] constatou que, ao contrário do que se pensava, os tempos de vida dos processos, em um sistema operacional UNIX, não são exponencialmente distribuídos. Segundo este estudo, uma função de distribuição de probabilidade igual à definida pela equação 10 descreve melhor o tempo de vida dos processos. Manipulações algébricas feitas na função acima conduzem à função de distribuição condicional definida pelo equação 11.

Dado que um processo tenha vivido ***b*** segundos, a probabilidade de que viva, pelo menos, ***a*** segundos é de ***b/a*** (equação 11). Ou seja, processos velhos, aqueles que possuem uma idade ***b*** avançada, têm uma probabilidade maior de viverem mais. Isto sugere que sejam escolhidos, para migrar, processos com idades de execução avançadas, e, portanto, com altas chances de terem uma longa vida que custeie os gastos de migração despendidos com eles. Há no entanto, dois problemas potenciais com este critério de migração: primeiro, a avassaladora maioria dos processos possui uma vida curta, podendo não haver suficientes processos idosos, o que reduz bastante os efeitos do balanceamento da carga; segundo, o custo adicional, relativo à transferência de memória, incorrido na migração de um processo idoso pode ser superior aos benefícios oriundos de sua migração.

A insuficiência de processos idosos é compensada pelo fato de que tais processos representam uma larga fração da carga total da CPU. De acordo com as medições feitas em [BAT 94], menos de 3,5% dos processos vivem mais de 2 segs., no entanto, eles contribuem com mais de 60% da carga total da CPU. Ainda que se esteja limitado a migrar uma reduzida parcela de processos, composta somente pelos que possuem uma idade avançada, tais migrações terão um grande impacto sobre o desempenho sistema, haja vista que um único processo extenso impõe, para o caso de uma máquina sobrecarregada, atrasos consideráveis a processos pequenos, se comparados ao tempo de vida destes.

Quanto aos custos de transferência de memória (custos de migração), resta fazer com que o critério de escolha os contemple. Não bastará a um processo, para que seja escolhido para migrar, ser o mais idoso de sua máquina. É necessário que as vantagens de transferi-lo sejam confrontadas com os eventuais gastos decorrentes de sua transferência. Os autores, objetivando reduzir o *slowdown* (ver seção 3.3), chegaram a um critério de idade mínima para a transferência, expresso pela equação 12 (figura 3.10).

Nesta equação, $\boldsymbol{IdadeMínimaMigração_{y,j}}$ é a idade mínima necessária para que o processo y seja migrado para a máquina vizinha j, L é o número total de processos locais, L_j é o número total de processos na máquina j e $\boldsymbol{CustosdeMigração_y}$ o tempo dedicado à transferência do processo y. Tal fórmula descreve uma situação de equilíbrio, onde a vantagem de se migrar um processo idoso y — a maior probabilidade de que este venha a viver mais (equação 11, figura 3.10) — é contrabalançada pelo tempo despendido em sua migração. Este tempo varia de processo para processo e é, geralmente, função do volume da memória alocada pelo processo. A diferença entre a carga local e a da máquina vizinha j entra como denominador da fórmula, o que é intuitivamente aceitável. A dedução formal da equação 11 pode ser encontrada em [BAT 95].

A manutenção de uma lista com os nomes dos processos que tendem a ter uma longa vida e a verificação se o processo pertence ou não à lista, é outro critério que pode ser adotado na escolha da carga.

3.7.2.3 Política de Localização

Existe uma infinidade de políticas de localização propostas na literatura. Por brevidade, nos ateremos às mais significantes, seja por serem bastante conhecidas, seja pela simplicidade extrema que apresentam em sua concepção, ou, de forma contrária, pelo grau de requinte.

Como havíamos dito anteriormente, há uma profunda ligação entre a disseminação de informação e a política de localização. As decisões sobre para qual máquina enviar as cargas excedentes dependem fortemente do detalhamento das informações possuídas e das dimensões do domínio de balanceamento em que uma máquina está inserida. Por tal razão, em breves linhas, apresentamos, quando necessário, juntamente à política de localização, um sumário da política de disseminação de informação.

As três primeiras políticas que veremos prezam pela simplicidade. Tais políticas foram expostas e analisadas no célebre artigo [EAG 86]. Este trabalho procurou, entre outras coisas, demonstrar que nem sempre sofisticação e desempenho andam juntos. Os gastos incorridos na administração da política de balanceamento — coleta, processamento e disseminação das informações — e o efeito maléfico que inevitavelmente informações desatualizadas causarão sobre a qualidade das decisões inibem, segundo os autores, os benefícios oriundos de políticas de localização que levam em consideração, em suas decisões, minuciosas informações sobre o estado das máquinas.

A política de localização **aleatória**, como diz o próprio nome, escolhe, a esmo, qual será a máquina destino das cargas. Ela não necessita, para sua decisão, de nenhuma informação sobre o estado do sistema.

A política **limiar** consulta, aleatoriamente, um número máximo M_p de máquinas, procurando por uma cujo índice de carga, mesmo após a transferência de mais uma carga para tal máquina, seja inferior ao valor de limiar V_{max}. A máquina destino será a primeira das máquinas consultadas que satisfaça a condição, sendo interrompido, neste momento, o processo de consultas. Caso não se encontre nenhuma, a carga é processada localmente. Aqui, as informações são obtidas somente quando delas se precisa (sob demanda) e a única informação exigida pelo processo decisório é a carga local das máquinas consultadas.

A política de **menor-valor** consulta um conjunto composto de exatamente M_p máquinas, aleatoriamente escolhidas. Entre as máquinas deste conjunto, será selecionada aquela que possuir o menor valor para o índice de carga. O processo de consultas é interrompido somente se uma máquina vazia é encontrada. Novamente, as informações são obtidas sob demanda, mas agora o processo decisório exige um volume maior de informação.

Outro artigo basilar na área de balanceamento de carga é [ZHO 88]. Neste artigo, além das políticas precedentes, são analisadas outras quatro: distribuída, global, central, por reserva.

Na política **por reserva**, em situações de carga baixa, uma máquina consulta outras R máquinas, a fim de determinar quais delas são detentoras de um índice de carga superior a V_{min}. Para aquelas que o são, a máquina lhes envia uma requisição de reserva. As reservas pendentes são armazenadas em uma pilha de tal forma que, ao chegar uma carga candidata a ser transferida, tal carga é enviada à maquina que fez a reserva mais recente. Caso a carga da máquina caia abaixo do valor V_{min}, todas as reservas são canceladas.

Na política **distribuída**, uma máquina, periodicamente, dissemina seu índice de carga a todas as máquinas de seu domínio de balanceamento. Somente aquelas que possuírem, em relação ao índice de carga local, uma diferença de carga superior a ΔL, são consideradas candidatas no processo de balanceamento. Entre as candidatas, é escolhida aquela com o menor índice de carga.

As políticas **global** e **central** possuem, ambas, o mesmo critério de seleção que o adotado pela política **distribuída.** Diferem somente quanto à política de disseminação de informação utilizada. Remetemos o leitor à seção 3.7.1.3, onde as políticas de disseminação de informação utilizadas por global e central são detalhadas.

Finalizando esta seção, como exemplo de políticas que se valem de um alto grau de requinte na tomada de suas decisões, apresentaremos uma proposta extraída de [GOS 93].

A maioria dos trabalhos na área de balanceamento de carga considera que nenhuma informação sobre as demandas das cargas de trabalho é conhecida, ou está

disponível, *a priori*. Tal presunção se mantém válida, pois é impossível determinar, **com exatidão**, *a priori*, o consumo dos recursos por parte das cargas. Entretanto um estudo conduzido por [DEV 89], mostrou que as requisições de CPU, memória ou E/S de uma carga podem ser previstas, com um satisfatório grau de precisão, antes mesmo de sua execução, através de um método estatístico de reconhecimento de padrões. Os autores reportam um coeficiente de correlação de 0,84 entre as requisições de CPU (tempo de processamento) previstas e as reais.

O trabalho proposto por [GOS 93] tira partido disto ao elaborar as decisões de balanceamento. Vale aqui observar uma tênue semelhança com o trabalho proposto por [BAT 95], onde as decisões tomadas partem de um conhecimento *a priori* sobre a distribuição de freqüência dos tempos de vida dos processos

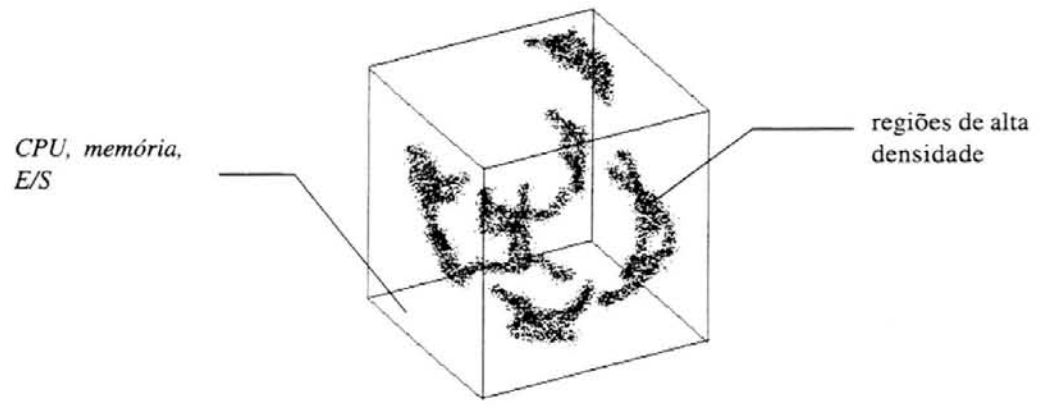


FIGURA 3.11 - Determinação dos estados de utilização de recursos

Em [GOS 93], o esquema de predição consiste de duas parte. Na primeira parte, são definidos os **estados de utilização de recursos** para a execução de programas em uma máquina UNIX. Medidas sobre a utilização, por parte dos processos, dos recursos são coletadas, em um sistema real, durante alguns dias. Cada processo é representado por um ponto em um espaço tridimensional cujas dimensões correspondem ao recursos do sistema, a saber, CPU, memória e E/S. Um algoritmo de agrupamento estatístico é utilizado para identificar as regiões de alta densidade do espaço tridimensional, isto é, determinar o número de tais regiões e as médias de seus centróides. Por definição a maior parte das execuções dos programas ocorre próximo a estas regiões, que, portanto, são referidas como estados de utilização dos recursos. A figura 3.11 apresenta uma possível configuração de demandas, representadas por pontos no espaço formado pelas dimensões *CPU x memória x E/S*.

Na segunda parte, a predição é realmente feita. O esquema de predição constrói e mantém um **modelo de transição de estados** para cada programa. Os estados do modelo são os estados de utilização de recursos, definidos acima. Suponha que um programa tenha sido executado várias vezes, fornecendo uma seqüência de instâncias de execução. Primeiramente, a seqüência de instâncias de execução é convertida em uma seqüência de estados de utilização de recursos, atribuindo a cada instância de execução o estado de utilização de recursos mais próximo. As probabilidades de transição de estado são, então, calculadas, a partir desta nova seqüência, para se construir um modelo de transição de estados para o programa. A predição da demanda é um cálculo de média

ponderada que se vale do modelo corrente de transição de estados para o programa e da utilização real dos recursos feita pelo programa em sua mais recente execução.

Em [GOS 93], são apresentadas quatro políticas de balanceamento distintas: MINQ, MINRT, DMINQ, FDMINQ. As políticas DMINQ e FDMINQ são variantes de MINQ que possuem uma organização de disseminação de informação distribuída. Omitiremos a exposição destas duas políticas por razões de brevidade e, também, por julgarmos que, no que concerne às contribuições feitas pelo trabalho à área, bastaria expor MINQ e MINRT.

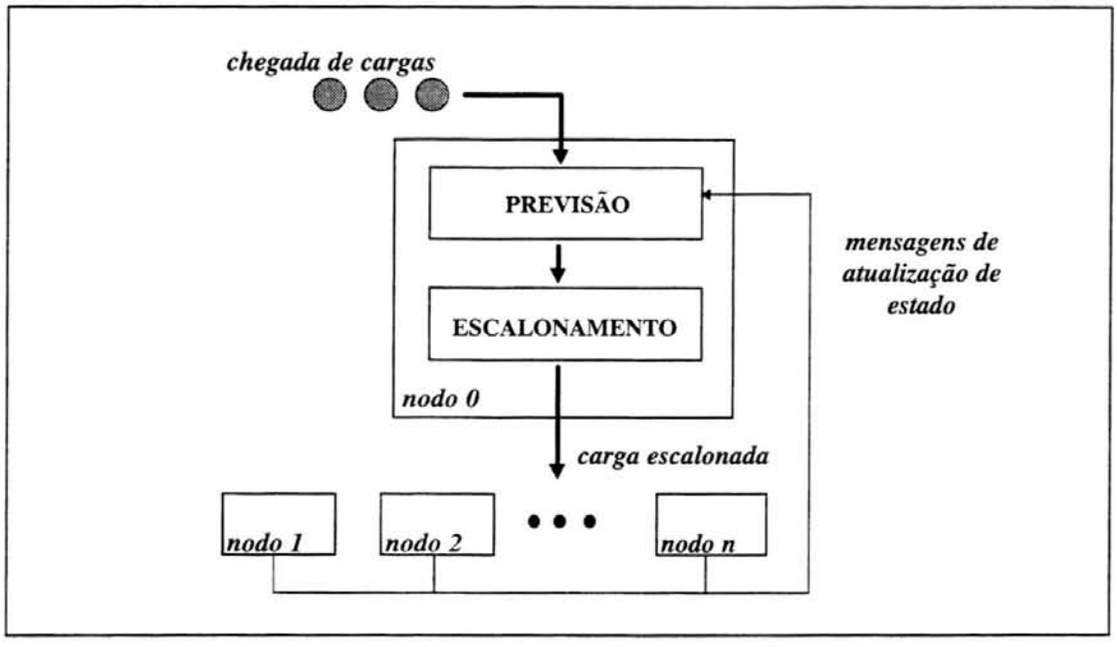


FIGURA 3.12 - Política de localização MINRT/MINQ

Tanto **MINQ** quanto **MINRT** possuem uma organização de disseminação de informação centralizada. A um único nodo é facultado saber sobre o estado dos demais, e decisões de balanceamento só podem ser tomadas por este nodo. A diferença aqui, em relação à organização centralizada exposta na seção 3.7.1.3, é que a chegada de cargas não ocorre em todos os nodos do sistema. A entrada de cargas no sistema somente se dá no nodo designado para ser o gerente de toda a informação do sistema. Este nodo, além de ser o responsável por todas as decisões de balanceamento, age aqui como um *frontend* para os demais.

O funcionamento de MINQ e MINRT é bastante similar. Em ambos os casos, quando um novo processo chega no escalonador central, a identificação do programa sendo executado é enviada ao previsor, a fim de que este preveja os recursos demandados pelo processo. Os valores previstos alimentam o escalonador que, de acordo com as políticas de localização específicas, determina em qual nodo será executado o processo. Quando um processo completa sua execução, sua demanda real é armazenada. Periodicamente, cada nodo envia ao escalonador central uma lista contendo as demandas reais dos processos que expiraram neste entretempo. Esta informação é utilizada pelo previsor na atualização dos modelos de transição de estados. A figura 3.12 esquematiza o funcionamento de MINRT e MINQ.

Empregaremos nos parágrafos seguintes a nomenclatura própria à proposta, adotada em [GOS 93]. Ressalvamos, no entanto, que conceitos como 'carga de CPU' ou 'carga de um nodo' não são objetos de consenso na literatura. Para eles, diversas conotações podem ser encontradas, tornando-se necessário, antes de usá-los, defini-los com precisão. A seção 3.4 perfila algumas das principais definições encontradas na literatura.

A **carga de CPU de um processo** é definida como sendo a razão entre o tempo de CPU e o tempo de execução — tempo de CPU acrescido do tempo gasto com operações de E/S — previstos para o processo (equação 13). Por sua vez, a **carga de CPU de um nodo** é definida como sendo a soma das cargas de CPU de todos os processos rodando naquele nodo (equação 14).

$$CargaCPUProcesso_y = \frac{TempoCpu_y}{TempoCpu_y + TempoES_y} \qquad (13)$$

$$CargaCPUNodo_j = \sum_{x=1}^{X_j} CargaCPUProcesso_x \qquad (14)$$

FIGURA 3.13 - Carga de CPU

A política de localização de MINQ concerne em atribuir o novo processo ao nodo com a menor carga de CPU, presumindo que, em tal nodo, o processo obterá o melhor tempo de resposta.

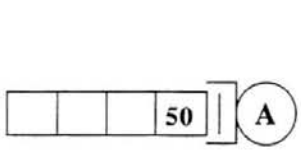
Para cada nodo ***j***, o escalonador mantém uma lista de processos ativos e seus tempos de CPU e de E/S. A chegada de um novo processo ***y*** acarreta a seguinte seqüência de passos, dados pelo escalonador: *(i)* calcula a carga de CPU para cada um dos nodos; *(ii)* fornece ao previsor a identificação do programa sendo executado e obtém, como resposta, a previsão da demanda para este; *(iii)* envia o processo ***y*** ao nodo com o menor valor de carga de CPU; *(iv)* acrescenta uma nova entrada contendo os tempos de CPU e E/S previstos para ***y*** à lista de processos ativos do nodo escolhido.

Ao receber de um nodo uma lista contendo os processos que expiraram e os recursos reais que tais processos consumiram, o escalonador toma as seguintes atitudes: *(i)* alimenta o previsor com a identificação dos programas e as demandas reais — utilização real de CPU e de E/S, de forma que este possa atualizar as tabelas de transição de estados para os programas; *(ii)* remove as entradas relativas aos processos que expiraram.

Assim como MINQ, MINRT mantém uma lista de processos ativos para cada um dos nodos. A diferença fundamental entre MINQ e MINRT reside no índice que adotam ao escolherem o nodo que receberá o novo processo.

MINQ seleciona o nodo com a menor carga de CPU. Entretanto, quando processadores utilizam escalonamento por *quantum*, a escolha do nodo com a menor carga de CPU não garante o melhor tempo de resposta para o processo. O seguinte exemplo ilustra esta questão. Suponha dois nodos ***A*** e ***B***, cada qual com sua respectiva fila de processos. Estes processos são escalonados para executarem intercaladamente a

intervalos fixos de 100ms. (*quantum* de execução). As demandas dos processos, em ***A*** e ***B***, se restringem à utilização da CPU, sendo nulas as exigências quanto às operações de E/S. Tais demandas, expressas em números de *quantum*, podem ser vistas na figura 3.14. O cálculo feito por MINQ para a carga de CPU dos nodos também é apresentado.



	Processo que chega	*Nodo A*	*Nodo B*
MINQ Carga de CPU	—	1	3
MINRT Tempo de Resposta	1	2	4
	25	50	28

FIGURA 3.14 - Comparação entre Carga de CPU e Tempo de Resposta

Se chega um novo processo requisitando 1 *quantum* de CPU, MINQ o escalonará para o nodo ***A***, onde o seu tempo de resposta será igual a 2 *quanta*. Caso fosse escalonado para o nodo ***B***, seu tempo de resposta seria igual a 4 *quanta*. Neste caso, a decisão tomada por MINQ é a mais acertada.

$$TREsperado_{y,j} = \sum_{x=1}^{X_j} I \times TempoCpu_x + (1-I) \times TempoCpu_y \quad (15)$$

$$I = \begin{cases} 1 & se\ TempoCpu_x < TempoCpu_y \\ 0 & c.\ c. \end{cases}$$

$$\left| [1..X_j] \quad processos\ na\ maquina\ j \right|$$

FIGURA 3.15 - Tempo de Resposta Previsto

Se, por outro lado, o processo exige para si 25 *quanta* de CPU, seu tempo de resposta no nodo ***A*** é igual a 50 *quanta*, e somente 28 *quanta* no nodo ***B***. MINQ, no entanto, atribui novamente o processo ao nodo ***A***, pois o critério de escolha que adota não leva em consideração a demanda do processo, tomando aqui uma decisão equivocada.

Isto sugere que pelo menos três fatores — o número de processos na fila, a demanda de CPU destes processos e a demanda de CPU do processo sendo escalonado — sejam levados em consideração pelo escalonador ao selecionar um nodo.

MINRT leva todos os três fatores em consideração ao calcular o **tempo de resposta esperado** que um processo ***y*** obterá quando for escalonado em um nodo ***j***. O tempo de resposta esperado para um novo processo ***y*** escalonado para executar em um nodo ***j*** é a soma dos seguintes termos: *(i)* as demandas de CPU dos processos no nodo,

caso sejam menores que a demanda de CPU de ***y***; *(ii)* uma quantia igual à demanda de ***y*** para cada um dos processos restantes (equação 15, figura 3.15).

Os passos detalhados seguidos pelo escalonador são: *(i)* as demandas do processo ***y*** (CPU e E/S) são calculadas utilizando os mecanismos de previsão; *(ii)* supondo que o escalonamento nos processadores seja feito por atribuição de *quanta* aos processos, o tempo de resposta estimado, $\boldsymbol{TREsperado_{y,j}}$, que o processo ***y*** obterá executando no nodo ***j***, é calculado segundo a equação 15 (figura 3.15); *(iii)* o nodo com o menor valor de $\boldsymbol{TREsperado_{y,j}}$ é escolhido para executar o processo ***y***; *(iv)* o processo ***y*** é acrescentado à lista de processos apropriada.

Ao estimar o tempo de resposta esperado, MINRT não leva em consideração a demanda de E/S. Para fazer isto corretamente, são necessárias informações adicionais sobre os tempos em que as operações de E/S são executadas, relativas ao início do processo. O esquema de predição não fornece esta informação. Ao não considerar as operações de E/S, o escalonador superestima os tempos de resposta. Isto não constitui um problema, visto que tempos de respostas absolutos não são necessários. O que realmente se necessita em uma política de localização é de medidas comparativas consistentes, que sejam capazes de distinguir, entre os nodos, os mais adequados a receber um novo processo.

3.8 Classificação Bibliográfica

Revisões bibliográficas completas, relativas à época de sua publicação, podem ser encontradas em [CAS 88b] e [WAN 85]. A tabela 3.3 tem como objetivo não uma minuciosa revisão bibliográfica, mas sim a organização das citações presentes neste capítulo. Vale notar que um mesmo artigo pode aparecer em duas ou mais linhas da tabela. A área de abrangência coberta por certos trabalhos impossibilita classificá-los sob um único ângulo. A escolha das classes foi arbitrária, e visou somente uma adequação ao conteúdo exposto e à forma como este conteúdo foi organizado.

TABELA 3.3 - Classificação bibliográfica

ÊNFASE DADA PELO TRABALHO	*BIBLIOGRAFIA*
Taxonomias	**[CAS 88b] [WAN 85]**
Índices de Carga, Índices de Desempenho e Medidas de Relevância	**[FER 85] [KUN 91] [CAS 88] [STA 85] [GOS 93] [LOH 96][FER 86]**
Política de Informação	**[LEM 93] [ZHO 88] [WAN 85] [MUN 95]**
Política de Ativação	**[WAN 85] [LEM 93] [DIK 89]**
Política de Transferência	**[BAT 94] [BAT 95] [DOW 95] [SON 94] [ZHO 87]**

Política de Localização	**[EAG 86] [NI 85] [LEM 93] [ZHO 88] [ZHO 87] [WAN 85] [HEI 92] [GOS 93] [LOH 96] [DIK 89] [BAR 85] [MUN 95]**

3.9 Conclusão

Este capítulo procurou discorrer sobre cada um dos elementos que integram a composição de um modelo para o problema de balanceamento de carga. Desta forma, falou-se aqui sobre a especificação da carga de trabalho, sobre a topologia do sistema, sobre índices de carga e de desempenho, sobre os módulos incumbidos de promoverem tanto a difusão de informação quanto a efetiva redistribuição de carga, enfim, acerca de cada um dos tópicos sobre os quais se detiveram a maioria dos autores na construção de seus modelos de balanceamento. O texto procurou demonstrar, para cada um dos tópicos abordados, as considerações feitas pelos principais autores. Nem sempre estes autores concordam quanto aos detalhes que devem ser considerados na construção do modelo e quanto à forma pela qual tais detalhes são efetivamente introduzidos no modelo. Sobre tais divergências, foram tecidos paralelos entre as decisões dos principais autores.

A maioria das decisões de projeto tomadas na construção do ambiente Robin Hood, abordadas nos capítulos seguintes, se norteou pelo conteúdo deste capítulo. O esforço empreendido em compreender as diferentes importâncias dadas pelos autores a particularidades do problema de balanceamento se refletirá nos capítulos seguintes. Desta forma, o ambiente procura, sempre que possível, flexibilizar as opções oferecidas ao analista, fazendo com que ele, e não o ambiente, decida pela relevância dada a tais particularidades. Os índices de desempenho, a descrição da demanda, a especificação dos padrões de chegada e de demanda, a topologia, a organização com a qual serão feitas as disseminações de informação, os instantes em que o Módulo de Gerência será ativado, todas estas opções são deixadas a cargo do analista.

Todavia, por constituírem um certo consenso na literatura, certas opções serão, de antemão, disponibilizadas ao analista na forma de primitivas. Pelo que foi apresentado, é de se esperar, portanto, que o tempo de resposta médio esteja presente entre os índices de desempenho. O mesmo vale, no caso dos índices de carga, para o número de processos prontos para execução. Distribuições estatísticas comumente empregadas são também fornecidas, a saber, exponencial, uniforme, hiperexponencial, normal, erlang.

Os capítulos seguintes procurarão descrever o ambiente Robin Hood apresentando, no caso do capítulo 4, os principais aspectos relativos à funcionalidade apresentada pelo ambiente, e, no caso do capítulo 5, os detalhes relativos à construção do mesmo. Cada um dos elementos constitutivos do modelo de balanceamento considerados no capítulo que aqui se encerra serão retomados pelos capítulos subseqüentes (principalmente pelo 4), ao serem mencionadas as decisões que, acerca destes elementos, o ambiente Robin Hood teve de tomar.

4 Ambiente Robin Hood — Funcionalidade

Neste capítulo serão apresentadas as decisões tomadas no processo de definição do ambiente Robin Hood sobre cada um dos elementos constitutivos de um modelo de balanceamento. As seções aqui contidas procuram fornecer a resposta dada às questões colocadas no capítulo anterior.

A primeira seção perfila os motivos que deram ensejo à construção do ambiente Robin Hood. As duas próximas (4.2 e 4.3) descrevem o modelo de balanceamento adotado e o relacionam com o modelo apresentado na seção 3.7.2 do capítulo precedente. A caracterização da carga de trabalho, isto é, a escolha dos recursos cujo consumo deverá compor a especificação da demanda, é tema da seção 4.4. A seção seguinte discorre sobre os índices de carga oferecidos pelo ambiente Robin Hood. O número de processos prontos para execução é apontado como índice de carga preestabelecido pelo ambiente, mas é deixado em aberto a possibilidade de redefini-lo.

A seção 4.6 se encarrega de discriminar os índices de desempenho fornecidos pelo ambiente como resultado do exercício do modelo. A massa de resultados gerada pelo exercício do modelo constitui um conjunto completo de dados sobre o comportamento do sistema ao longo da simulação. Além dos índices finais — tais como o tempo de resposta médio ou o *slowdown* médio —, que representam um resumo do comportamento do sistema, o ambiente Robin Hood fornece dados referentes à monitoração periódica do sistema — tais como a variação instantânea da carga para cada uma das máquinas, a variação instantânea do desvio padrão calculado sobre o índice de carga das máquinas, *etc*.

Os tópicos relativos à confecção do Módulo de Informação e Balanceamento são detalhados na seção 4.7. Esta seção se incumbe ainda de apresentar o paradigma de comunicação adotado pelo ambiente Robin Hood, a saber, a entrega e o recebimento assíncronos de mensagens. Através deste paradigma serão efetuadas todas as eventuais trocas de mensagens feitas entre os nodos. A próxima seção descreve os custos considerados pelo ambiente e a forma de especificá-los. Por fim, a seção 4.9 enumera e comenta os vários passos pelos quais terá de passar o analista na confecção e simulação de sua política de balanceamento.

4.1 Motivação

O problema de balanceamento exposto em detalhes no capítulo anterior, muito embora não seja recente, pois objeto de estudo há mais de uma década [BOK 79][CHW 79] [EFE 82][STA 85][FER 85][EAG 86], ainda atrai o interesse dos pesquisadores e levanta questões em aberto. Vários trabalhos recentes confirmam isto [MIL 94] [BAT 94][BAL95][DOW 95] [MUN 95][FRI 96][LOH96].

Definir políticas satisfatórias está longe de ser, ao contrário do que afirmavam Eager et al. [EAG 86], uma tarefa fácil. Se políticas de balanceamento simples são suficientes para se obter um considerável ganho de desempenho [EAG 86][ZHO 88], não se conclui, com isto, que o limite para tal ganho tenha sido atingido e que políticas complexas são desnecessárias. Trabalhos recentes [GOS 93][BAT 94][DOW 95], cujas

políticas se valem de um conjunto de informações mais rico que o utilizado pelas políticas em [EAG 86], ampliaram os ganhos de desempenho até então alcançados.

A multiplicidade de fatores que afetam o comportamento de uma política de balanceamento e, por conseqüência, do sistema onde ela atua, impõe sérios obstáculos ao desenvolvimento e análise de novas políticas de balanceamento. A descrição adequada da carga submetida ao sistema [BAT 94][GOS 93], a topologia com a qual se interconectam as máquinas do sistema [LOH 96], os atrasos de entrega sofridos pelas mensagens nos enlaces, os custos associados ao posicionamento ou à migração das cargas [EAG 86][BAT 95], os índices utilizados para descrever o estado de carga das máquinas [KUN 91][FER 85][FER 87], e os parâmetros exigidos pelos módulos de balanceamento e gerenciamento de informação [EAG 86][ZOH 87][ZOH 88] [GOW 93] são alguns dos diversos fatores que devem ser considerados ao se analisar uma política de balanceamento.

A observação da evolução dos índices de desempenho convencionais — por exemplo, o tempo de resposta médio dos processos ou a taxa de utilização média das CPUs — como função da variação dos fatores acima mencionados é uma das etapas indispensáveis a uma análise de desempenho feita sobre políticas de balanceamento. Uma análise de desempenho extensiva deve ser capaz, ainda, de fornecer respostas às questões colocadas sobre o comportamento da política frente a situações limites, quando, para determinados parâmetros em observação, são atribuídos valores extremos. Deve, por fim, ser capaz de perceber e determinar a importância das correlações existentes entre os parâmetros de entradas e os vários índices de saída, muitas das vezes extremamente sutis. Comportamentos anômalos, dos quais a instabilidade excessiva é um exemplo [STA 85][EAG 86], devem ser detectados. É necessário, no entanto, certa cautela quanto ao rigor das restrições impostas à instabilidade presente no sistema, pois nem sempre ocorre que tal instabilidade seja prejudicial ao desempenho, ou que se configure um comportamento anômalo [CAS 88].

Implantar uma política de balanceamento em um sistema real para, em seguida, estudar-lhe o comportamento e as possíveis benfeitorias não é uma idéia sensata. As ações e decisões de balanceamento exigem um contato íntimo com as estruturas do sistema operacional [NOG 95][MIL94][DOU 91] e, a menos que as facilidades de balanceamento tomem a forma de uma ferramenta disposta sobre o sistema operacional, a exemplo de [DIK 89], serão despendidos grandes esforços para implementá-las, testá-las e depurá-las.

De qualquer forma, a utilização de **facilidades reais de balanceamento**, isto é, facilidades implantadas em um sistema real e cujas cargas de trabalho são modeladas como processos reais, exigirá forçosamente todo o sistema exclusivamente para si, sejam tais facilidades construídas sobre o sistema operacional, ou como parte integrante deste, pois, dificilmente, se poderá rotineiramente utilizar alguma das máquinas sem sofrer com as perturbações oriundas das decisões de balanceamento. Infelizmente, nem sempre se pode dispor de uma coleção de máquina alocadas exclusivamente para a análise de políticas de balanceamento.

Outro inconveniente surge, para as facilidades reais de balanceamento, da impossibilidade de se repetir um experimento, por não se conseguir restabelecer condições idênticas às originais. Isto prejudica, em muito, a confecção de análises

comparativas entre políticas, pois não é de todo impossível que perturbações externas ao experimento conduzam a resultados equivocados.

A vasta gama de fatores que influenciam o desempenho de uma política de balanceamento e a dificuldade, disto decorrente, de analisar até certo grau de extensão o comportamento de uma política são argumentos que, aliados a outros, relativos à utilização de facilidades reais de balanceamento — o considerável volume de esforços despendidos na implementação, a exigência quanto ao uso exclusivo de um conjunto de máquinas e a impossibilidade de repetir experimentos —, apontam o caminho e servem de motivação à construção de uma ferramenta de simulação. Tanto assim, que a grande maioria dos estudos encontrados na literatura emprega técnicas de simulação [BAT 94][BAT 95][CAS 88][GOS 93] [HEI 92] [LOH 96] [NI 85] [SON 94] [STA 85] [ZHO 88]. Poucos são os estudos que utilizam facilidades reais, sejam estas construídas sobre o sistema operacional [LEM 93][ZHO 87][KUN 91] [DIK 89], ou como parte integrante deste [BAR 85][MIL 94].

Não há, no entanto, na maior parte dos trabalhos sobre balanceamento, uma referência explícita sobre a utilização de uma ferramenta de análise específica. Construir uma ferramenta de análise, para posteriormente, através dela, obter os resultados desejados, seria exigir do analista um dispêndio desnecessário de esforços, trocando, equivocadamente, os meios pelos fins.

O suporte fornecido por linguagens de simulação convencionais, tais como GPSS [KAR 91], SIMSCRIPT [RUS 90], ModSim [HER 90] ou RESQ [SOA 90], atenua, mas não elimina o problema. Se por um lado, a transposição do modelo é facilitada por primitivas de simulação fornecidas, em contrapartida, exige-se, do analista, o conhecimento da linguagem de simulação. Além do que, por objetivarem ser de propósito geral, as linguagens de simulação não contemplam nenhum dos aspectos imediatos do problema de balanceamento. Facilitam a transposição, mas esta não é, de forma alguma, direta. Alterações no modelo exigem modificações substanciais no código fonte e as facilidades de análise apresentadas por estas linguagens não levam em conta detalhes do problema modelado.

Muitas das vezes, o analista sequer tem acesso a uma linguagem de simulação conhecida, ou por trabalhar em uma plataforma para a qual a linguagem ainda não foi portada, ou por não dispor de recursos para adquiri-la, haja vista que a maioria das linguagens de simulação são produtos comerciais e, portanto, exigem, para sua aquisição, o pagamento de uma taxa. O desempenho apresentado pelas linguagens de simulação também depõe contra sua utilização, uma vez que a simulação de um modelo, em tais linguagens, possui um desempenho sofrível se comparada à simulação do mesmo modelo em uma linguagem de programação convencional [DOY 90].

Deste contexto, emergem fortes argumentos a favor da concepção e desenvolvimento de uma ferramenta específica para a análise de políticas de balanceamento de carga.

4.2 Abordagens à Confecção de um Ambiente para a Avaliação de Políticas de Balanceamento

Levando em conta o que foi dito no capítulo anterior sobre o problema, o que se poderia esperar de um ambiente para avaliação de políticas de balanceamento de carga? Conforme exposto, a literatura apresenta, com fartura, concepções diversas para vários aspectos do problema. Desde a descrição adequada da demanda [STA 85][EAG 86] [ZHO 88][HEI 92][GOW 93][BAT 95], passando pela atribuição dos custos [NI 85] [EAG 86][BAT 95], até a escolha dos índices de carga e de desempenho [FER 85] [STA 85][KUN 91][LOH 96], sobre todos estes aspectos não há, na literatura, uma posição consensual. Como, então, construir tal ambiente?

Uma resposta imediatista propugnaria por escolher, entre as opções oferecidas pelos autores, aquelas que se julgarem mais adequadas, descartando sumariamente as demais, para, então, construir, com as escolhas feitas, o ambiente. Essa linha de raciocínio possui o revés de fazer com que o ambiente tome para si a tarefa de atribuir ou denegar importância aos aspectos do modelo, retirando-a do analista. O analista ficaria sujeito às escolhas feitas, e eventualmente, caso não concordasse com elas, seria forçado a desistir de utilizar o ambiente.

De forma contrária, poder-se-ia postergar, na construção do ambiente, tais escolhas, atribuindo futuramente ao analista este papel. Com isto, caberia ao analista decidir a ênfase dada às particularidades do modelo que deseja simular. Todavia, se for demasiado o esforço exigido ao analista para a confecção de seu modelo, já que o ambiente propositadamente se omite de contemplar vários detalhes, a razão para utilizar o ambiente também desaparecerá.

Um meio-termo entre estes extremos tem de ser alcançado, onde se contrabalancem decisões deixadas a cargo do analista com decisões feitas pelo ambiente de avaliação. Uma solução para este empasse, e que foi adotada pelo ambiente Robin Hood, consiste em construir um ambiente de avaliação onde as principais decisões referentes aos detalhes do modelo de balanceamento estão preestabelecidas, mas fica em aberto a possibilidade de estendê-las, ou mesmo redefini-las. Nesta solução, idealmente, o ambiente deve fornecer, ainda, suporte à redefinição ou extensão dos elementos do modelo dos quais o analista discorde, ou mesmo elementos alternativos já implementados.

A subseção seguinte fornecerá uma visão geral de como foi concebido o modelo de balanceamento adotado pelo ambiente Robin Hood, e as principais entidades que o integram. As próximas abordarão individualmente cada um dos detalhes do modelo para os quais o ambiente Robin Hood procurou encontrar uma solução adequada. Entre estes detalhes se encontram: a caracterização da carga de trabalho e a descrição dos padrões de chegada e de serviço (para os diversos recursos demandados), os índices de carga e de desempenho, os momentos de ativação dos módulos de balanceamento, a definição da política de balanceamento e, por fim, os custos de transferência, de ativação dos módulos e de envio de mensagens.

4.3 Modelo de Balanceamento Adotado

O modelo de balanceamento adotado pelo ambiente Robin Hood tem como fonte de inspiração vários dos modelos propostos na literatura [EAG 86][LEM 93][LOH 96] [BAT 94]. Como demonstra a figura 4.1, duas entidades principais compõem o modelo adotado: o nodo e a fonte. A **fonte** é encarregada de gerar, segundo uma determinada distribuição de tempo entre chegadas, a carga de trabalho que será processada pelas CPUs dos nodos. A carga de trabalho é caracterizada pela **demanda** de recursos que carrega consigo.

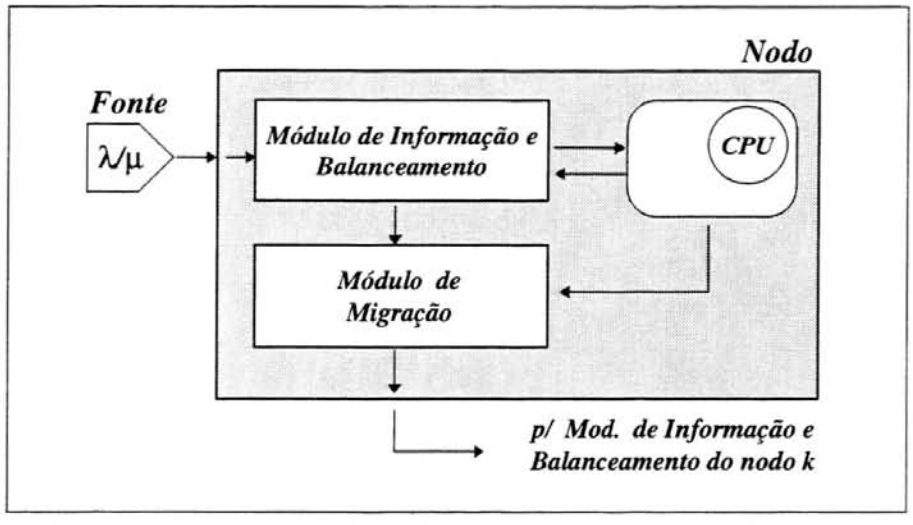


FIGURA 4.1 - Modelo de balanceamento adotado

Cabe ao **nodo**, por sua vez, determinar o destino que será dado às cargas recebidas — sejam estas provenientes diretamente da fonte associada ao nodo, sejam oriundas dos demais nodos que compõem o sistema — processando-as localmente, ou então, repassando-as a outros nodos. São elementos constituintes do nodo, como demonstra a figura 4.1, o Módulo de Informação e Balanceamento, a unidade de processamento (CPU), e o Módulo de Migração.

O Módulo de Gerência de Informação e o Módulo de Tomada de Decisões, apresentados na seção 3.7, se fundem em um único. Os primeiros esboços para o modelo de balanceamento adotado possuíam, como sugerido pela seção 3.7, dois módulos distintos e autônomos: um responsável somente pela gerência de informação e o outro pela tomada de decisões. Contudo, ao longo do desenvolvimento do modelo e do ambiente Robin Hood, foi constatado que existiam fortes relacionamentos entre estes dois módulos e que os acessos feitos por ambos visavam quase sempre os mesmos dados. Não se justificava, portanto, dividir no modelo o que na prática era uno. Conquanto se concorde que exista uma nítida linha divisória entre as funcionalidades exercidas pelos módulos (Gerência de Informação e Tomada de Decisões), as restrições encontradas na implementação do modelo fizeram com que ele forçosamente se modificasse e incorporasse a concepção de um módulo único. Isto não empobrece, de forma alguma, o modelo, nem prejudica sua flexibilidade em comportar a definição de diversas políticas de balanceamento.

Como a figura 4.1 retrata, o destino da carga de trabalho, escolhido pelo Módulo de Informação e Balanceamento, pode ser dois: (*i*) a carga de trabalho é enviada para a CPU, a fim de que seja executada; (*ii*) a carga de trabalho é imediatamente transferida para outra máquina, pelo intermédio do Módulo de Migração.

Os custos de transferência são determinados pelo Módulo de Migração. O termo 'Migração', com que o módulo é cunhado, não exclui a possibilidade de o modelo comportar a definição de **políticas de posicionamento** (seção 3.7.2.2). Estas políticas, aqui, são consideradas um subconjunto das políticas de migração, onde o universo de cargas candidatas à transferência se limita àquelas cujo início ainda não tenha se dado (recebidas da fonte, mas não enviadas ainda à CPU). O termo 'Migração', como aqui empregado, significa, pois, a liberdade em se escolher a carga a ser transferida, e não está vinculado especificamente, como seria de se supor, a nenhuma categoria de políticas.

TABELA 4.1 - Elementos exigidos pelo ambiente Robin Hood

Fonte	***Nodo***	***Sistema***
distribuição de tempo entre chegadas	*algoritmo que implementa o Mod. de Informação e Balanceamento*	
distribuição de tempo de serviço	*custos de transferência (Mod. de Migração)*	*topologia*
*demanda**	*custos de transmissão de mensagens (atraso)*	
*distribuição de demanda**	*CPU **	

** especificação opcional*

Os nodos são interligados por enlaces de comunicação e não compartilham memória primária. Qualquer forma de comunicação que envolva mais de um nodo tem de ser feita através de trocas de mensagens. As mensagens transmitidas, devido à latência natural do meio, sofrem atrasos. O atraso associado à transmissão não é explicitado no modelo. Cumpre ao analista determinar os valores que os atrasos terão. Excepcionalmente, caso deseje, o analista poderá, inclusive, eliminar qualquer espécie de atraso, fazendo com que as mensagens sejam entregues instantaneamente.

A topologia também não é contemplada pelo modelo. A especificação de como os nodos serão interligados é tarefa do analista. O modelo aqui proposto funciona como um gabarito de modelos: vários elementos essenciais não são especificados, tais como qual distribuição empregar para o intervalo de tempo entre chegadas, qual configuração topológica utilizar, de que maneira os custos de transferência de cargas e transmissão de mensagens serão definidos, etc. A especificação destes elementos conduzirá à efetiva construção de um modelo. O ambiente Robin Hood exige, portanto, que o analista especifique uma série de elementos ausentes, e componha com eles o modelo que lhe aprouver. Os elementos que devem ser especificados são apresentados na tabela 4.1.

Mais adiante, nas seções 5.3 e 5.4 do capítulo seguinte, a especificação dos vários elementos do modelo será retomada sob um novo ângulo. Sob a perspectiva da implementação do ambiente, serão repassadas cada uma das etapas de especificação aqui descritas.

4.4 Caracterização da Carga de Trabalho e dos Padrões de Chegada e Serviço

A caracterização da carga de trabalho estipula quais os recursos são objetos de demanda por parte de uma carga que ingressa no sistema. Os autores caracterizam tradicionalmente a carga de trabalho, em seus modelos de balanceamento, exclusivamente pelo consumo do tempo de processamento [WAN 85][EAG 86] [BAT 95]. Não há dúvida de que, para o problema de balanceamento, a CPU seja o principal recurso a ser considerado, mas resta saber se é realmente o único importante o suficiente a ponto de relegarmos os demais.

Em políticas que permitem a transferência de cargas que já tenham iniciado sua computação (políticas de migração), grande parte do tempo despendido na transferência de uma carga se deve ao envio, de uma máquina para outra, do conteúdo de memória e do estado associados à carga. Eis porque tais políticas habitualmente incluem na caracterização da carga de trabalho o volume de memória demandado [STA 85] [BAT 95]. Outros autores, ao tempo de processamento, adicionalmente incluem na caracterização da carga: o tempo despendido em operações de E/S [ZHO 88][GOW 93] ou a comunicação entre processos [HEI 92].

O ambiente Robin Hood optou por incluir unicamente o **tempo de processamento** na caracterização da carga de trabalho. Todavia, ao contrário do que se poderia pensar, esta decisão não impede que outros recursos sejam considerados. A carga de trabalho definida pelo ambiente almeja ser somente o ponto de partida, com o qual poderão ser construídas extensões específicas, que incluam os recursos imprescindíveis para o modelo em questão. Funciona, portanto, como denominador comum às caracterizações encontradas na literatura.

Esta caracterização bastará a uma grande parcela de políticas (referenciadas no primeiro parágrafo desta seção) e nenhuma intervenção deverá ser feita pelo analista ao utilizar o ambiente. Para as demais, onde outros recursos são considerados, cumpre ao analista estender a caraterização da carga, incluindo os recursos necessários. Esta questão é observada pela tabela 4.1, que marca como opcionais, na coluna referente aos elementos da fonte, a especificação da demanda e da distribuição incumbida de descrever a demanda.

Caso o analista se decida por estender a caracterização da carga de trabalho, além de fornecer o rol dos recursos cujas demandas deverão ser contempladas e as distribuições incumbidas de descrevê-las, ele deverá, ainda, prover o ambiente de uma CPU capaz de atender a tais novas demandas. Se optar, por exemplo, em incluir o consumo de memória, deve, então, prover uma distribuição que determine o consumo deste recurso. Além disto, uma extensão à CPU deve ser fornecida pelo analista. O funcionamento da CPU está intimamente relacionado com a caracterização da carga de trabalho e, de antemão, assume, nesta, a existência somente do consumo do tempo de processamento. A extensão de uma implica a extensão de outra, onde, para cada novo

recurso adicionado, cujo consumo é levado em conta, deve haver um acréscimo de funcionalidade correspondente por parte da CPU, que objetive tratar o consumo do novo recurso.

Uma questão, afim à caracterização da carga, e igualmente relevante, é determinar qual será a distribuição empregada para descrever o consumo dos recursos considerados (padrão de serviço), ou mesmo, o intervalo entre chegadas de cargas (padrão de chegada). O impasse fica entre escolher uma distribuição estatística convencional (exponencial, uniforme, etc.) [STA 85][WAN 85][NI 85][EAG 86] [CAS 88], implementada através da geração de números pseudo aleatórios, ou optar por utilizar um arquivo de rastro que norteie a evolução do sistema [ZHO 88][GOW 93][BAT 94][BAT 95][LOH 96]. O ambiente Robin Hood fornece ambas as alternativas para o analista: (*i*) disponibiliza, como primitivas, algumas das principais distribuições estatísticas conhecidas, a saber: **erlang**, **normal**, **exponencial**, **hiperexponencial**, **poisson** e **determinística**; (*ii*) ou, caso se deseje, permite que a simulação seja norteada pela utilização de um arquivo de rastro. A escolha do que empregar fica a cargo do analista.

Se entre as distribuições estatísticas oferecidas pelo ambiente não figurar a que o analista procura, é deixada, em aberto, a possibilidade de o analista implementá-la e integrá-la às demais do ambiente. Obviamente, isto demandará esforço e tempo extras. Em compensação, não se restringe o universo de escolhas do analista às opções oferecidas pelo ambiente.

Informações mais precisas podem ser encontradas na seção 5.3 do capítulo seguinte. Esta seção enumera as várias distribuições fornecidas pelo ambiente Robin Hood, apresenta o paradigma de orientação por objetos através do qual tais distribuições foram implementadas, descreve em detalhes o processo de implementação e explica como novas extensões (novas distribuições) podem facilmente ser agregadas ao ambiente, graças às virtudes de tal paradigma.

4.5 Índices de Carga

Os índices de carga quantitativamente expressam o estado de carga de uma máquina — se esta se encontra sobrecarregada, ociosa, ou em um estado intermediário de carga —, e são, portanto, empregados pelas decisões de balanceamento ao se decidir quais serão as máquinas origem e destino das transferências de carga. O ambiente Robin Hood tem como índice de carga preestabelecido **o número de processos prontos para execução**. Vários autores agem de maneira similar [EAG 86][ZHO 88].

Recorremos aqui ao que antes se dissera para a caracterização da carga de trabalho: o índice fornecido pelo ambiente Robin Hood almeja ser simplesmente o ponto de partida para a definição de outros índices que se conformem com o interesse do analista. Caso o analista se incline para o índice preestabelecido pelo ambiente, nenhuma intervenção adicional lhe será exigida. Caso opte por um índice diverso, lhe é demandado que integre ao ambiente a concepção que pretende para o índice de carga.

4.6 Índices de Desempenho

Após o exercício do modelo, o ambiente Robin Hood fornece, de forma automática, para o analista, um conjunto de dados sobre a evolução do sistema. Fazem parte deste conjunto de dados, em relação aos processos cujos términos tenham ocorrido durante o intervalo de simulação:

- o **tempo de resposta** médio; a variância relativa ao cálculo do tempo de resposta médio; o histograma de freqüência para os tempos de resposta obtidos pelos processos;

- o ***slowdown*** médio; a variância relativa ao cálculo do *slowdown* médio; o histograma de freqüência para os *slowdowns* obtidos pelos processos;

- o **tempo de espera** médio; a variância relativa ao cálculo do tempo de espera médio; o histograma de freqüência para os tempos de espera obtidos pelos processos.

Em relação às CPUs que compõem o sistema:

- a **taxa de utilização** média das CPUs; a taxa de utilização individual, para cada uma das CPUs.

Em relação à cada uma das fontes:

- o **número de cargas** que a fonte originou; a **taxa de chegada** das cargas na fonte.

O número de partições em que se subdividem os histogramas de freqüências é um parâmetro variável, definido segundo a vontade do analista. Os histogramas possibilitam a elaboração de uma análise mais fina sobre um determinado índice de desempenho que a extraída pela simples observação do valor médio e de seu desvio. A título de exemplo, a figura 4.2 apresenta os resultados fornecidos, de forma automática, pelo ambiente Robin Hood ao analista. Estes resultados se referem à simulação de uma política de localização aleatória (seção 3.7.2.3) em um anel composto por quatro nodos.

Os índices de desempenho apresentados pelo ambiente foram selecionados entre os de emprego mais comum na literatura. A utilização do tempo de resposta médio é praticamente consensual [STA 85] [WAN 85] [NI 85] [EAG 86] [FER 87] [ZHO 87] [ZHO 88][DIK 89] [KUN 91][HEI 92][GOW93][EVA 94], e sua presença aqui se faz obrigatória. Os demais índices, conquanto não sejam também objeto de consenso, encontram espaço entre os autores [ZHO 88][HEI 92][BAT 95] e auxiliam à compreensão sobre o comportamento do sistema.

A descrição do sistema através da coleta e análise de um índice — por exemplo, o tempo de resposta médio — assinala somente o resultado final das ações de uma determinada política; sintetiza, ao máximo, o sucesso ou insucesso decorrentes de tais ações. Todavia, esconde a evolução do sistema ao longo do tempo, os erros e acertos cometidos pelas decisões de balanceamento, as situações limites ou excepcionais. Não nos permite que saibamos a seqüência de estados pelos quais as máquinas do sistema

passaram, nem traz consigo quaisquer indícios sobre a instabilidade provocada pela política ou sobre o histórico do desbalanceamento entre as máquinas.

```
[Simulator]
# of Events   = 269163
# of Objects  = 796443
Final SimTime = 8000.0000

[SysSink]
Mean Response Time =  33.7905
Variance = 848.2199
Mean Wait Time     =  16.4411
Variance = 317.3150
Mean Slowndown     =   2.4882
Variance =   1.6639

---  Response Time Frequency Table ---
   0.3328 |---    20.4371      555
  20.4371 |---    40.5414      461
  40.5414 |---    60.6457      232
  60.6457 |---    80.7500      119
  80.7500 |---   100.8542       47
 100.8542 |---   120.9585       22
 120.9585 |---   141.0628        8
 141.0628 |---   161.1671        4
 161.1671 |---   181.2714        3
 181.2714 |---|  201.3757        9

[SysCpuRoundRobin] NODE 4
idle  time =  1626.297
total time =  8000.000
utilization=     0.797

[SysSource] NODE 4
arrival rate =     0.046
# of arrivals= 367

[SysCpuRoundRobin] NODE 3
idle  time =  1655.900
total time =  8000.000
utilization=     0.793

[SysSource] NODE 3
arrival rate =     0.046
# of arrivals= 367
```

```
---  Wait   Time   Frequency Table ---
   0.0000 |---    12.3863       800
  12.3863 |---    24.7726       341
  24.7726 |---    37.1589       162
  37.1589 |---    49.5451        83
  49.5451 |---    61.9314        29
  61.9314 |---    74.3177        19
  74.3177 |---    86.7040        11
  86.7040 |---    99.0903         3
  99.0903 |---   111.4766         7
 111.4766 |---|  123.8629         5

---  Slowndown   Frequency   Table ---
 1.0 |---   2.0         0
 2.0 |---   3.0       618
 3.0 |---   4.0       418
 4.0 |---   5.0       251
 5.0 |---   6.0       111
 6.0 |---   7.0        26
 7.0 |---   8.0        20
 8.0 |---   9.0         6
 9.0 |---  10.0         4
 9.0 |---  oo           6

[SysCpuRoundRobin] NODE 2
idle  time =  1640.874
total time =  8000.000
utilization=     0.795

[SysSource] NODE 2
arrival rate =     0.046
# of arrivals= 367

[SysCpuRoundRobin] NODE 1
idle  time =  1642.495
total time =  8000.000
utilization=     0.795

[SysSource] NODE 1
arrival rate =     0.046
# of arrivals= 367
```

FIGURA 4.2 - Índices de Desempenho finais fornecidos pelo ambiente Robin Hood

Com vistas a suprir tal lacuna, o ambiente Robin Hood provê adicionalmente ao analista a possibilidade de observar, ao longo do tempo, a evolução **da carga das máquinas, da média de cargas do sistema** (cumulativa e instantânea) e **do desbalanceamento** sofrido. Estes dados são obtidos graças a uma monitoração periódica do sistema: um agente de monitoração, a intervalos regularmente espaçados de tempo, é ativado, cabendo a ele coletar os índices de carga das várias máquinas que integram o sistema, calcular e registrar a média e o desvio padrão 'instantâneos' de tais índices, e compor o cálculo da média cumulativa com os valores então coletados.

Dois são os índices de carga coletados pelo agente de monitoração: (*i*) um fixo, e que corresponde sempre ao **número de processos prontos para execução**; (*ii*) o outro variável e *a priori* desconhecido, que corresponde à redefinição personalizada dada pelo

analista ao índice de carga (seção 4.5). Caso o analista opte pelo índice de carga preestabelecido, e não haja uma redefinição ou extensão ao índice fornecido pelo ambiente, os índices (*i*) e (*ii*) se confundirão. Se, pelo contrário, o índice preestabelecido for preterido por um outro, personalizado, de interesse do analista, os índices (*i*) e (*ii*) serão distintos e referir-se-ão a coisas diversas. Em ambos os casos, a presença do número de processos prontos para execução é certa.

A escolha deste índice se justifica pela simplicidade com que sua mensuração é feita e pela clareza com a qual expõe a evolução do estado das máquinas. A maioria dos estudos, onde são apresentados gráficos que indicam a evolução temporal do estado das máquinas, concordam quanto à escolha do número de processos prontos para execução como índice de carga a ser observado [CAS 88][ZHO 88].

Os dados apresentados pelas figuras 4.3 e 4.4 dizem respeito à mesma simulação da qual foram extraídos os dados que compõem a figura 4.2. Sob certo aspecto, as figuras 4.3 e 4.4 são complementares à figura 4.2: esta mostra os índices de desempenho finais — tempo de resposta médio, taxa de utilização média das CPUs, etc. —, cujos valores representam um resumo para o desempenho do sistema durante o intervalo de mensuração considerado, enquanto aquelas traçam a evolução do estado de carga ao longo do tempo, seja de forma individual, restringindo-se à observação de uma única máquina (figura 4.3), seja de forma coletiva, considerando-se, agora, as médias instantânea e cumulativa de carga para o sistema (figura 4.4).

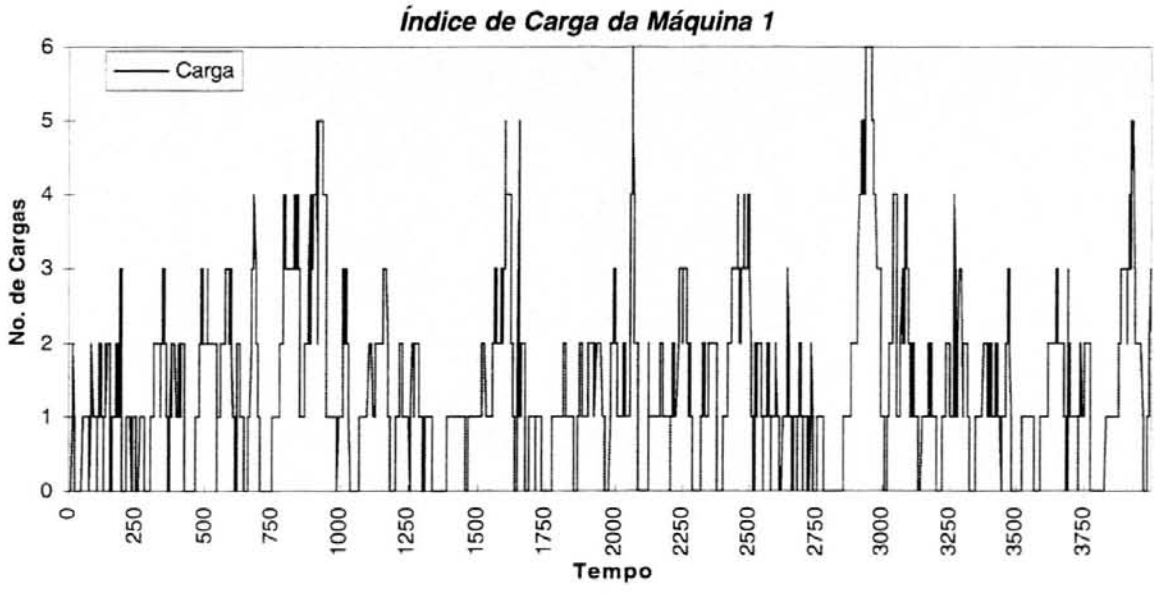


FIGURA 4.3 - Evolução do Índice de Carga ao longo do tempo

Pela análise da evolução dos estados de carga das máquinas (figura 4.3), se flagrarão decisões incorretas de balanceamento, picos de carga momentâneos, situações excepcionais, para a máquina, de sobrecarga, de ociosidade, de instabilidade, ou outras anomalias inesperadas. Através dos dados que apresenta é que ensejaremos compreender o porquê dos índices finais apresentados pela figura 4.2.

A figura 4.4 se imbui do mesmo propósito — apresentar a evolução da carga, mas, ao contrário da figura anterior, não se limita a uma observação individual das máquinas, optando por retratar a evolução da carga de acordo com a ótica do conjunto. Nesta figura, a média instantânea de cargas, isto é, a média aritmética dos índices de carga das máquinas em um dado momento do tempo, é confrontada com a média cumulativa, onde não só o instante presente é levado em consideração para o cálculo do índice, mas também todas as médias anteriores, coletadas em tempos passados. Se comparada à média instantânea, a média cumulativa possui uma gradual e suave variação, pois ao passo que o momento presente define o valor de uma, implicando, via de regra, abruptas alterações, para a outra representa apenas um termo dentro de uma longa série que, mais e mais, ao avançar do tempo, tem sua influência reduzida.

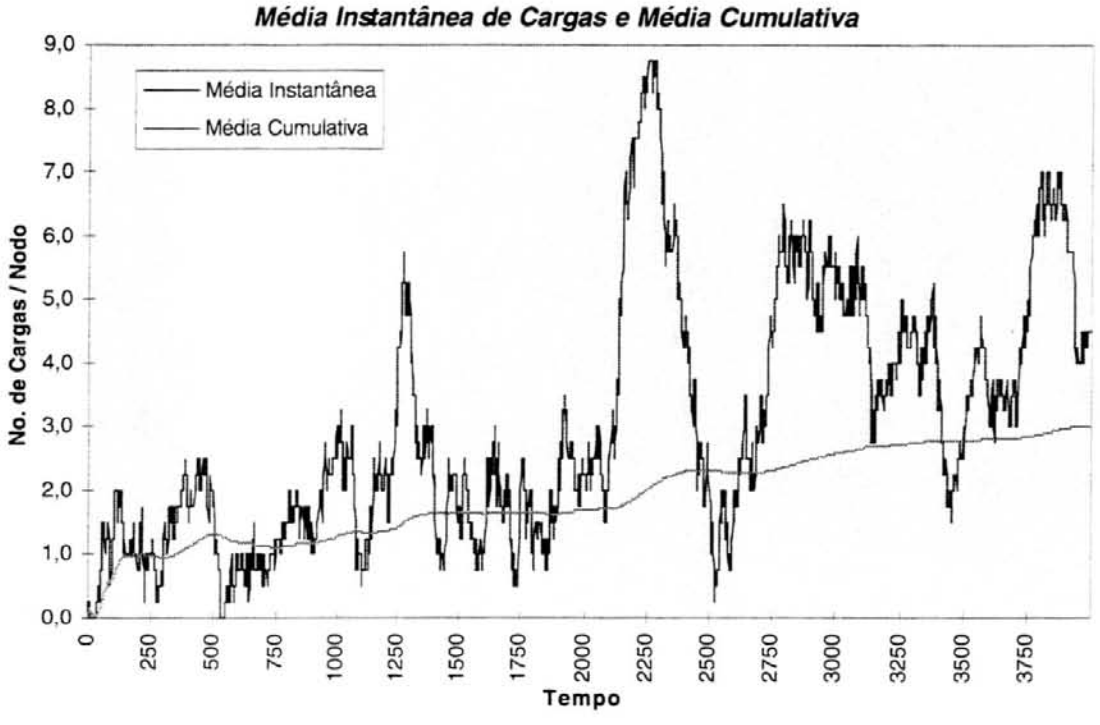


FIGURA 4.4 - Média de Carga Instantânea e Cumulativa ao longo do tempo

Contraposta à figura 4.3, a figura 4.4 nos permite distinguir de quanto, em relação à média do sistema, difere a carga de uma máquina. No entanto, quando analisada isoladamente, as informações contidas na figura 4.4 perdem um pouco de seu significado, pois é indiferente, para o cálculo da média, o desbalanceamento existente entre as máquinas. Assim, se, em um dado instante, as máquinas ***1***, ***2***, ***3***, e ***4***, que compõem o sistema, possuem, cada uma delas, 4 cargas, ou se possuem, respectivamente, 14, 0, 0 e 2 cargas, a média será para ambos os casos, de modo indistinto, 4 cargas por máquina.

Se a figura 4.4 não se presta a aclarar situações de desbalanceamento — e nem é este seu propósito, não se deve, com isto, concluir que não contribui significativamente para a análise de políticas. As informações contidas neste tipo de gráfico permitem determinar o **volume de carga** submetido ao sistema, parâmetro de extrema relevância, uma vez que a ele está intimamente vinculado o comportamento de uma política de balanceamento. Vale lembrar aqui as colocações feitas por [ZHO 88], quando, a fim de

estabelecer comparações válidas entre os desempenhos de uma política frente a sistemas de diferentes tamanhos, ressalta a importância de não descuidarmos de manter, para os diferentes tamanhos em avaliação, o mesmo nível de carga para o sistema.

Os gráficos abaixo, relacionados às figuras 4.5, 4.6 e 4.7, realçam a ocorrência de um fenômeno intrínseco ao problema de balanceamento: a **instabilidade** [STA 85][CAS 88]. A instabilidade surge da impossibilidade de uma máquina obter informações estritamente atualizadas sobre o estado das outras. Devido à latência do meio — salvo o caso excepcional, circunscrito ao campo teórico, onde as mensagens são entregues em tempo zero — as informações que uma máquina possui sobre o estado das demais estarão sempre defasadas em relação ao seu valor real. Destarte, logradas por informações inconsistentes, as decisões de balanceamento podem não surtir efeito algum, ou até mesmo pior, podem agravar o desbalanceamento do sistema ou o estado de sobrecarga de uma máquina.

Outra eventual causa para os problemas de instabilidade se deve à natureza distribuída com que as decisões de balanceamento, em algumas políticas, são tomadas. Pelo fato de não haver um esforço global de balanceamento — isto é, uma decisão única e válida para todo o sistema, onde sejam considerados os estados de todas as máquinas — e, por conseguinte, as decisões serem tomadas individualmente, os interesses que as norteiam podem entrar em conflito, e a soma das decisões não resultar no esperado, acentuando o estado de sobrecarga de algumas máquinas. Ao tentarmos balancear, desbalanceamos. Com isto, máquinas trocam cargas entre si, mas o desbalanceamento do sistema não se altera: máquinas ociosas tornam-se sobrecarregadas e máquinas sobrecarregadas ociosas. Os gráficos exemplos, apresentados pelas figuras abaixo (figura 4.5 e figura 4.6), padecem exatamente deste mal.

Políticas de balanceamento (***UDI1*** e ***UDI2***) foram extraídas de [CAS 88], implementadas através do ambiente Robin Hood, e os gráficos que se seguem delineiam o comportamento destas. As figuras 4.5 e 4.7 representam a evolução do estado de carga de uma máquina sob, respectivamente, as políticas ***UDI1*** e ***UDI2*** (*Unidirectional Incremental Transfers*). O gráfico da figura 4.6 constitui apenas uma ampliação de escala feita para o gráfico da figura 4.5, com o intuito de revelar detalhes escondidos e tornar patente o fenômeno da instabilidade. A presença destes gráficos aqui se justifica unicamente pela contundência com que expõem o problema da instabilidade. O interesse devotado às políticas UDI1 e UDI2 restringe-se às qualidades que possuem em tornar explícita, de forma tão marcante, a instabilidade que permeia o sistema. Em sistemas reais, a manifestação da instabilidade não se dá de forma tão óbvia e detectável e, muitas das vezes, passa desapercebida.

Os gráficos dispostos abaixo foram obtidos sob condições especialíssimas. Aqui, ao contrário do que naturalmente ocorre nos sistemas reais, onde as cargas chegam aos diversos nodos em tempos aleatoriamente espaçados, acarretando infindas e sucessivas perturbações, uma única perturbação é aplicada ao sistema: um pulso de cargas — a chegada quase simultânea de um grande número de cargas — é aplicado a um dos nodos do sistema. Feito isto, nenhuma outra carga ingressa no sistema, e as que ingressaram possuem, excepcionalmente, tempo de processamento infinito, não o abandonando jamais. Sob tais condições, o número de cargas presentes no sistema permanece constante, a despeito da passagem do tempo, pois vedadas ficam a entrada e a saída de

cargas. O processo de redistribuição de carga têm, então, início, quando se procurará balancear a distorção provocada pela inserção do pulso.

Idealmente, após um período limitado de tempo, quando houver logrado êxito o processo de redistribuição, estando o desbalanceamento do sistema reduzido a um nível considerado tolerável, findarão as transferências de cargas entre os nodos. Casavant [CAS 88] chama de **resposta** a este intervalo, que se estende da aplicação do pulso até o término (ou estabilização) das ações de redistribuição. Políticas, cujas ações de redistribuição não conduzem à redução efetiva do desbalanceamento entre as máquinas, onde, apesar das consecutivas transferências de cargas, o nível de tolerância para o desbalanceamento do sistema nunca é atingido, são referidas pelo autor como **instáveis**. Em situações de instabilidade, cargas são trocadas entre as máquinas, sem que se aperfeiçoe a almejada homogeneização de carga do sistema.

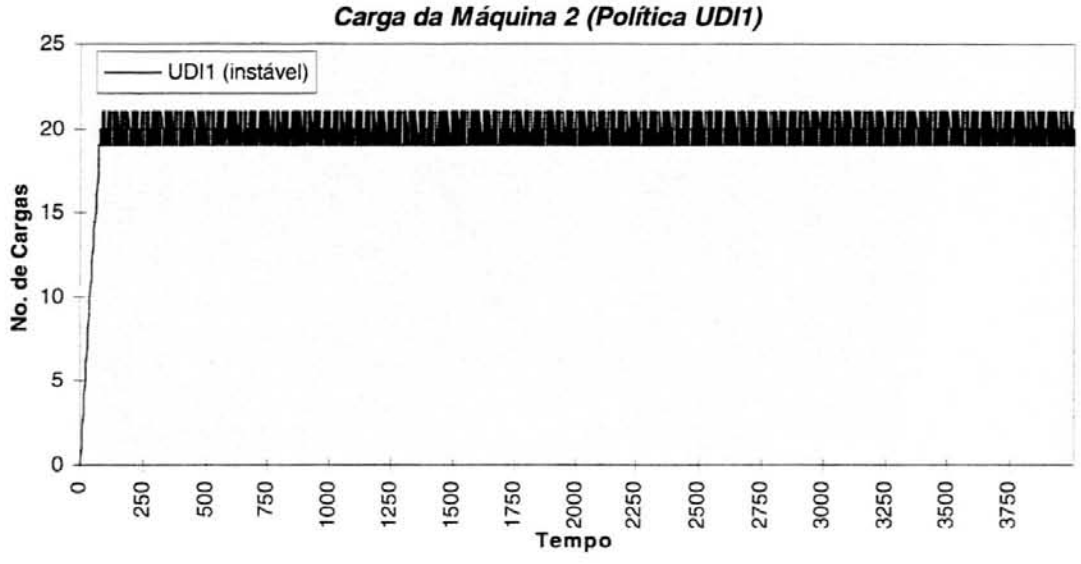


FIGURA 4.5 - Sistema com elevado grau de instabilidade

A análise do comportamento de um sistema frente a este conjunto excepcional de condições — a inserção de uma única perturbação, o número constante de cargas presente no sistema a despeito da passagem do tempo e as ações de balanceamento tomadas objetivando contornar a distorção provocada pela inserção do pulso — é chamada, pelo autor [CAS 88], de **estática**, por oposição à análise convencional, chamada **dinâmica**, na qual, para o sistema em observação, durante todo o intervalo de simulação, cargas ingressam e abandonam aleatoriamente o sistema.

O sistema em questão, seguindo as especificações do artigo do qual foi extraído, é composto por cinco máquinas interligadas em anel, numeradas seqüencialmente. Um pulso de 100 cargas é aplicado, inicialmente, à máquina 1. Após o fato, são observadas as conseqüências resultantes das ações de redistribuição de carga. As ações referentes à política UDI1 (figuras 4.5 e 4.6) não conseguem contornar o desbalanceamento subitamente introduzido pelo pulso, e acabam por provocar um eterno vaivém de cargas, sem atingir, no entanto, o estágio de balanceamento almejado. O número de cargas nas máquina ziguezagueiam (os gráficos contemplam apenas a máquina 2, mas todas as demais se portam de forma similar) devido somente às tentativas de redistribuição, afinal

não há saída ou ingresso de cargas no sistema, mas o nível de homogeneidade obtido nunca é considerado satisfatório.

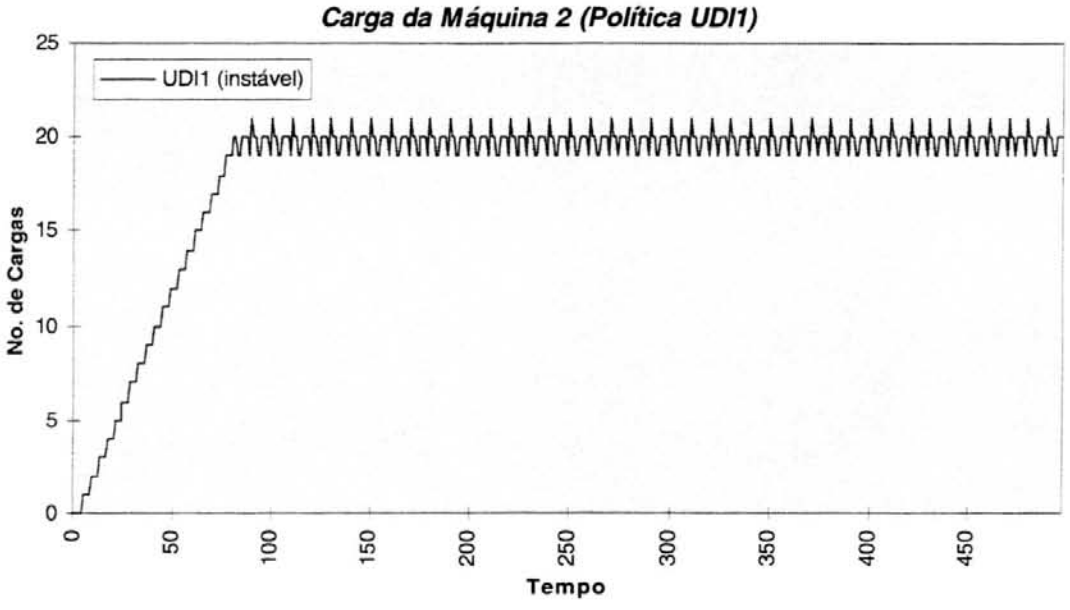


FIGURA 4.6 - Sistema com elevado grau de instabilidade (ampliação)

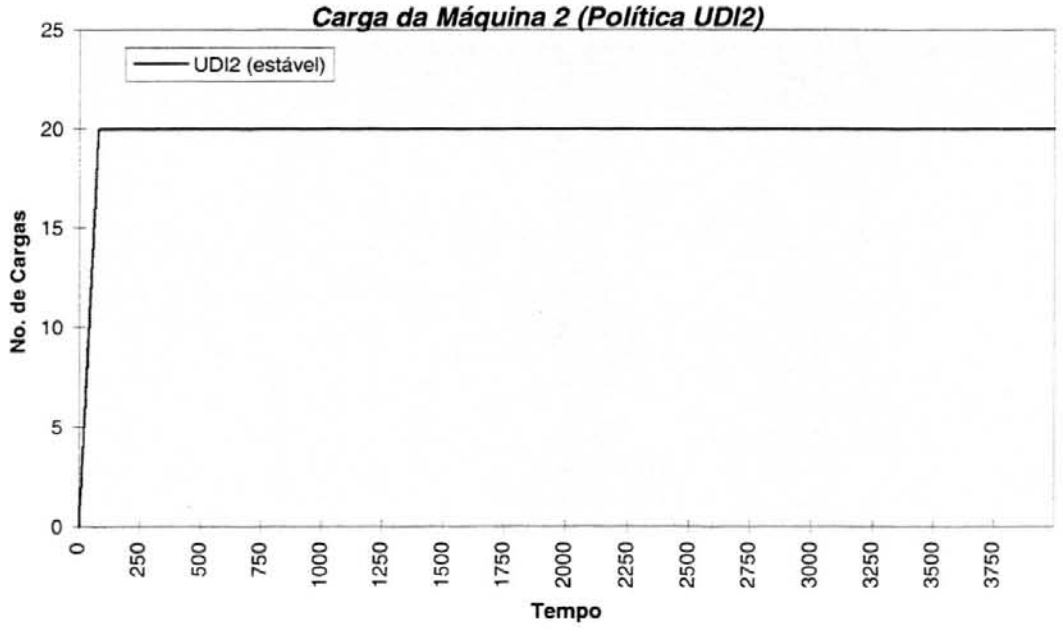


FIGURA 4.7 - Sistema estável

O mesmo não acontece para a política UDI2 (figura 4.7), onde, após um intervalo de tempo, cessam as ações de redistribuição de carga e o sistema converge para o nível de balanceamento almejado.

Para os estudiosos de balanceamento de carga, outro gráfico de sumo interesse é o que indica o desbalanceamento existente entre as máquinas do sistema ao longo do

tempo. Sua análise permite determinar a eficácia de uma política, quando esta se refere à subutilização dos recursos. A figura 4.8 exemplifica bem a questão. Nela são confrontados os desbalanceamentos obtidos por um sistema onde não há ações de redistribuição de carga e por um onde tais ações são tomadas segundo uma política de localização aleatória (seção 3.7.2.3). O sistema do qual foram extraídos os dados é constituído de quatro máquinas interligadas em anel.

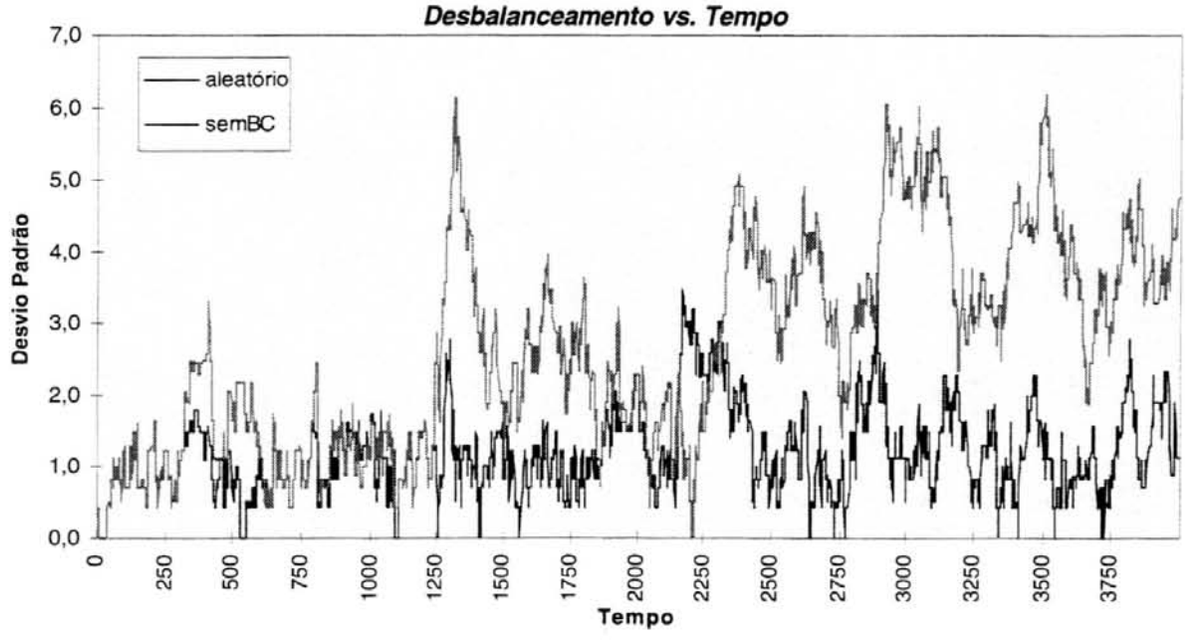


FIGURA 4.8 - Desbalanceamento do sistema ao longo do tempo

Como bem mostra a figura 4.8, o desvio padrão entre as máquinas é bem superior quando não há ações que intentem aperfeiçoar a redistribuição de cargas. A política aleatória, apesar de sua simplicidade, consegue sobremaneira reduzir a subutilização dos recursos, mantendo os índices de carga das máquinas dentro de uma estreita faixa de valores.

Ao longo desta seção foram perfilados alguns dos principais resultados fornecidos pelo ambiente Robin Hood. Procurou-se, em suma, com isto, demonstrar que tais resultados constituem um conjunto completo de informações sobre a evolução e desempenho do sistema, e permitem uma análise pormenorizada das políticas que através dele venham a ser simuladas. A análise estática do sistema, importante técnica disponibilizada pelo ambiente, possibilita a obtenção de valiosas informações relativas ao processo de estabilização do sistema. Através dos gráficos, valores e histogramas, presentes na massa de resultados, pode-se, com clareza, vislumbrar o comportamento de uma política e o impacto de suas ações sobre o sistema.

4.7 Definição do Módulo de Informação e Balanceamento

Existe, de antemão, presente no ambiente Robin Hood, um Módulo de Informação e Balanceamento preestabelecido. As atitudes tomadas por este módulo se limitam a receber, da fonte, a carga de trabalho e repassá-la diretamente à CPU.

Nenhuma tentativa de balancear o sistema é, desta forma, empreendida. As ações de balanceamento que porventura venham a ter lugar no sistema devem ser, então, implementadas, pelo analista, através da redefinição ou extensão do módulo. Todo o capítulo seguinte se esmerará em demonstrar como as funcionalidades aqui descritas são implementadas e como poderiam ser empregadas pelo analista na confecção e análise das políticas de suas políticas de balanceamento, como também as maneiras através das quais o ambiente poderia ser eventualmente estendido.

É tarefa do analista codificar a lógica que pretende dar para a política de disseminação e coleta de informação e para as decisões de balanceamento. Definir o Módulo de Informação e Balanceamento é, em verdade, no ambiente Robin Hood, especificar o comportamento do módulo frente à ocorrência de cada um dos eventos que venha a ativá-lo:

> *— O que fazer quando da chegada de uma carga? Avaliar o índice local de carga e, com base nele, decidir pela transferência da carga para um nodo aleatoriamente escolhido? Ou consultar a carga dos vizinhos imediatos, procurando uma decisão mais acertada?*

A ocorrência de determinados eventos exige uma resposta por parte do Módulo de Informação e Balanceamento. Essencialmente, a literatura enumera, como tais, os seguintes eventos:

- a chegada de uma carga ao nodo;
- o término da execução (partida) de uma carga;
- o término do período de ativação, quando esta ativação periodicamente ocorre.

Estes eventos abarcam todas as circunstâncias nas quais seria de interesse a ativação do Módulo de Informação e Balanceamento. Todavia, nem sempre a ocorrência de todos os eventos precisa ser considerada pela política de balanceamento, mas apenas de alguns deles. Políticas há, em que é de relevância somente a chegada de cargas — a transferência da carga que chega para um nodo aleatoriamente escolhido, condicionada a um estado de sobrecarga local é um bom exemplo. Outras desconsideram a chegada e a partida de cargas, e procuram somente manter, dentro de um patamar tolerável, a diferença entre a carga local e a dos vizinhos imediatos, periodicamente efetuando, se necessário, o balanceamento das cargas.

Quanto a este aspecto, o ambiente Robin Hood é extremamente flexível, ficando o analista livre para escolher os eventos que julgar adequado. A ativação do módulo pode ser vinculada, conforme sua preferência, à ocorrência de qualquer um dos eventos supracitados. Case opte por tal, a ativação do módulo pode se restringir somente à chegada de carga no nodo; ou então, exclusivamente, de forma periódica, a intervalos de tempo regularmente espaçados, cujo comprimento lhe cabe definir; ou mesmo ainda, vinculado à ocorrência de ambos os eventos. Selecionados os momentos de ativação apropriados, o analista deve agora definir o comportamento do módulo frente a tais eventos.

Outra importante questão se refere às primitivas de comunicação disponibilizadas pelo ambiente para a troca de informação entre os módulos dispostos nos vários nodos do sistema.

Existem vários paradigmas de comunicação propostos na literatura, entre os quais, os mais importantes são: troca assíncrona de mensagens, troca síncrona de mensagens, RPC (*Remote Procedure Call*), *rendezvous*, criação dinâmica de processos [AND 91][AND 93]. A maioria dos paradigmas, se presta não só à comunicação entre os processos, transportando informação de um para outro, mas funciona também como ponto de sincronização entre eles. Desta forma, em alguns dos paradigmas, a assertiva, presente no código e responsável pela comunicação, tem o papel auxiliar de, quando necessário, bloquear a execução do processo até que uma determinada condição se verifique.

A troca de mensagens é um bom exemplo disto. Para trocas **síncronas**, o processo, ao enviar a mensagem, deve esperar bloqueado até que o destinatário a tenha recebido. O envio funciona aqui como um ponto de sincronização entre os processos. Já para as trocas **assíncronas**, não há necessidade de o processo que envia a mensagem esperar pelo recebimento da mesma; assim que concluir a assertiva para o envio da mensagem, pode imediatamente prosseguir sua execução, a despeito do recebimento, ou não, desta pelo destinatário. Todavia, para ambos os casos, sejam as trocas síncronas ou assíncronas, a assertiva relativa ao recebimento da mensagem impõe uma sincronização condicional: o processo destinatário que, ao executar a assertiva de recebimento, não encontrar nenhuma mensagem ainda disponível, ficará bloqueado até que uma lhe seja enviada. Cabe observar que nem sempre os termos 'troca síncrona' e 'troca assíncrona' são empregados com o sentido que aqui lhes atribuímos. Tanenbaum, por exemplo, os relaciona com a utilização, ou não, de buffers para o envio das mensagens [TAN 95].

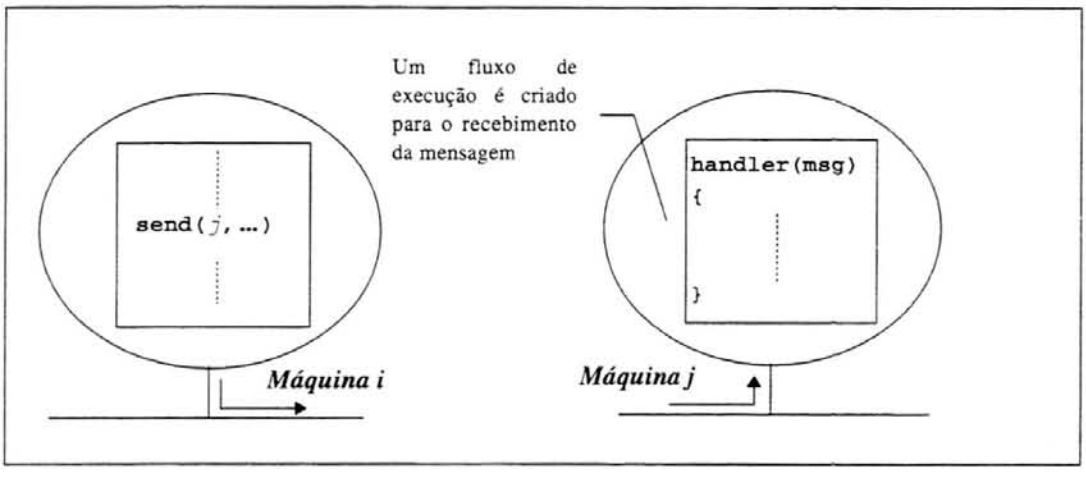


FIGURA 4.9 - Paradigma de comunicação do ambiente Robin Hood

O ambiente Robin Hood fornece apenas um paradigma de comunicação, a saber, o **envio** e **recebimento assíncronos** de mensagens. Aqui, ao contrário do exposto no parágrafo precedente, não há uma assertiva de recebimento que, sob determinadas condições — a inexistência de mensagens a serem recebidas — bloqueie a execução do processo. O envio assíncrono de uma mensagem, isto é, de forma imediata, sem que o processo emissor fique suspenso, provocará, no lado do destinatário, a criação de um

fluxo de execução responsável pelo recebimento da mensagem. Não há, no código do destinatário, assertivas de recebimento, mas sim, a definição de uma função que incumbir-se-á de tratar as mensagens recebidas. A figura 4.9 ilustra a seqüência de ações que decorre do envio de uma mensagem da máquina ***i*** para a máquina ***j***. Este paradigma de comunicação é bastante similar à criação dinâmica de processos, como proposto em [AND 93].

TABELA 4.2 - Primitivas de comunicação

	Há sincronização (condicional) ?	
	Envio	***Recebimento***
trocas síncronas	**sim**	**sim**
trocas assíncronas	**não**	**sim**
ambiente Robin Hood	**não**	**não**

A tabela 4.2 estabelece um paralelo entre as várias semânticas possíveis para a troca de mensagens. As duas primeiras linhas da tabela apresentam as trocas síncronas e assíncronas. A terceira, por sua vez, apresenta a semântica do paradigma de comunicação utilizado pelo ambiente Robin Hood.

4.8 Custos de Comunicação

A transferência de uma carga de uma máquina para outra, bem como, de forma similar, o envio de uma mensagem, incorrem em **custos de processamento** e de **utilização do enlace**. Os custos de processamento dizem respeito principalmente ao empacotamento dos dados e execução dos protocolos de comunicação, e acometem tanto a máquina que envia os dados quanto a que os recebe. O inevitável retardo, introduzido pela latência da rede de comunicação, acrescido do tempo necessário à efetiva transferência dos dados respondem pelos custos de utilização do enlace.

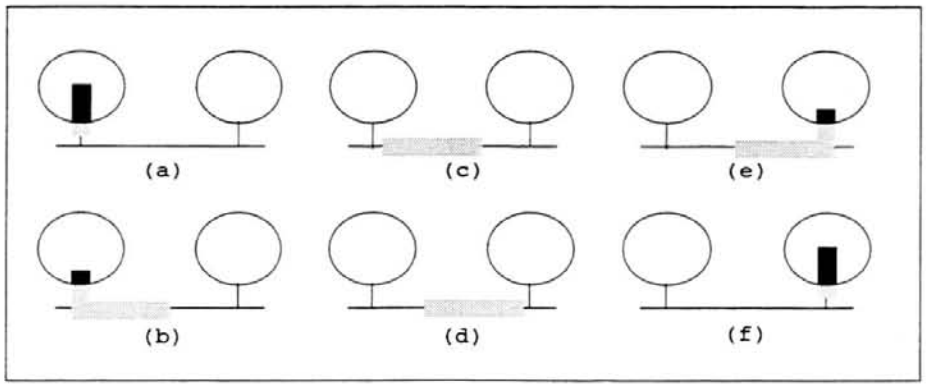


FIGURA 4.10 - Custos de comunicação

A figura 4.10 retrata esta situação. Nesta figura são apresentadas as várias etapas decorrentes do envio de uma mensagem (ou da transferência de uma carga) de uma máquina para outra. Podemos claramente observar os custos de processamento em que incorrem tanto a máquina emissora (a), quanto a receptora (f), e os custos associados à utilização do enlace (b, c, d, e) — o retardo inserido pela latência e o tempo despendido com a efetiva transferência dos dados.

Somente os custos referentes à utilização do enlace são considerados pelo ambiente Robin Hood. De acordo com o modelo adotado pelo ambiente, a transferência de uma carga ou o envio de uma mensagem não imputam à máquina emissora ou à receptora nenhum ônus de processamento, sendo levado em conta apenas o atraso inerente à utilização do enlace. O valor real atribuído aos custos de utilização do enlace é um parâmetro cuja especificação é exigida ao analista. Estabelecer custos fixos de comunicação, ou descrevê-los segundo alguma distribuição estatística apropriada, ou, ainda, defini-los como função do volume de dados transmitidos são decisões delegadas pelo ambiente ao analista. Conforme mencionado anteriormente, quando falávamos sobre a caracterização da carga de trabalho, o ambiente Robin Hood disponibiliza, como primitivas, um conjunto de distribuições estatísticas das quais o analista pode novamente se valer para a descrição dos custos de comunicação.

4.9 A Simulação do Modelo

As seções precedentes se ativeram detidamente sobre os principais aspectos relativos à confecção de um modelo de balanceamento, retratando de que forma o ambiente Robin Hood os aborda. Esta seção procurará integrá-los. Abaixo, pode-se vislumbrar a seqüência de passos exigidas pelo ambiente para a confecção e simulação de um modelo:

1. ***Especificação da topologia***
2. ***Caracterização da carga de trabalho***
3. ***Definição do índice de carga***
4. ***Descrição de cada uma das fontes do sistema (padrão de serviço/padrão de chegada)***
5. ***Descrição de cada um dos Módulos de Informação e Balanceamento do sistema***
6. ***Especificação dos custos de comunicação***
7. ***Especificação do comprimento do intervalo de simulação***
8. ***Especificação dos parâmetros referentes aos dados de saída fornecidos pelo simulador***

O primeiro passo a ser dado é especificar a topologia segundo a qual se interconectam os nodos do sistema. Não há restrições quanto às topologias aceitas pelo ambiente. Em seguida, o analista deve especificar o que entende pela demanda associada às cargas de trabalho. Este passo é facultativo, e se ignorado, implicará que somente o tempo de processamento seja contemplado pela demanda. Opcionalmente, o analista pode, a seu gosto, enriquecer a noção de demanda, acrescendo-lhe o consumo de outros

recursos. No próximo passo, o analista terá de descrever as fontes que compõem o sistema, determinando qual o padrão de chegadas e qual o padrão de demanda que as rege. Cada fonte pode ser individual e autonomamente especificada, e, em um caso limite, para um sistema de N nodos, há a possibilidade de haver N diferentes especificações. Na especificação dos padrões de serviço, o analista deverá considerar a caracterização feita no passo anterior para a carga de trabalho.

No passo 5, o analista atribui a cada um dos nodos do sistema, a definição que lhe aprouver para o Módulo de Informação e Balanceamento, incluindo, obviamente, as que ele próprio tenha desenvolvido. Aqui, vale o que antes foi dito para as fontes: a atribuição dos módulos aos nodos é feita de forma autônoma e individual. Isto permite a definição de políticas de balanceamento **hierárquicas**, onde os papéis desempenhados pelos vários módulos de balanceamento não são equivalentes. Com isto, o ambiente permite que o analista atribua a determinados nodos do sistema uma participação maior (ou menor) no esforço de balanceamento.

O próximo passo requer a especificação dos custos de comunicação (transferência de carga e envio de mensagens). Os custos atribuídos podem variar de enlace para enlace, e cumpre ao analista determinar como tais custos são avaliados. Os dois últimos passos se referem ao simulador: um solicita o comprimento do intervalo de mensuração; o outro, opcional, requer os parâmetros que nortearão a produção dos resultados (número de classes dos histogramas de freqüência, período de monitoração do sistema).

Um vez percorrida a seqüência de passos, o modelo de balanceamento se encontra completo e pronto para a simulação. O ambiente inicia então a compilação dos fontes e ligação das bibliotecas necessárias, bem como a simulação do modelo anteriormente definido.

4.10 Conclusão

Ao longo deste capítulo foram apresentadas as várias decisões tomadas pelo ambiente Robin Hood em relação a cada um dos elementos constitutivos de um modelo de balanceamento tal como exposto no capítulo 3. O capítulo 4, de certa forma, pode ser encarado como um 'Guia do Usuário'. Um texto onde são elencadas as principais características e qualidades do ambiente no que concerne ao auxílio que propicia quanto à definição e simulação de uma política de balanceamento. Com isto, finda a leitura do mesmo, já se pode vislumbrar as prováveis virtudes e falhas presentes no ambiente, determinar a distância porventura existente entre o que se propôs fazer e o que aqui é apresentado.

O próximo capítulo ater-se-á aos detalhes referentes à implementação do ambiente. Procurará descrever como foram implementadas as funcionalidades definidas pelo capítulo 4. Apresentará o ambiente, não mais do ponto de vista de um provável usuário, mas sim a partir da ótica do implementador.

5 O Ambiente Robin Hood — Implementação

Este capítulo procura, com os subsídios fornecidos pelos capítulos anteriores, apresentar a arquitetura interna do ambiente Robin Hood. Se coube ao capítulo 3 apresentar uma visão panorâmica dos principais trabalhos na área de balanceamento e ao capítulo 4 transpor as contribuições de tais trabalhos para o projeto funcional do ambiente Robin Hood, caberá a este capítulo discorrer sobre como, utilizando as técnicas de simulação expostas no capítulo 2, foi efetivamente implementado o ambiente Robin Hood.

A primeira seção retratará, em linhas gerais, como foi concebido o ambiente Robin Hood. Conforme será visto, a arquitetura do ambiente Robin Hood é constituída de camadas.

Cada camada se vincula a uma tarefa específica e funcionalmente distinta das tarefas das demais camadas. Segundo esta organização, uma camada utiliza os serviços prestados pelas camadas inferiores e disponibiliza às superiores os serviços que efetivamente implementa. As seções seguintes se ocuparão de cada uma das camadas que compõem a versão atual do ambiente Robin Hood.

A primeira camada, descrita na seção 5.2, implementa um simulador orientado por objetos e dirigido por eventos de propósito geral. A exemplo do que ocorre com as linguagens de simulação discreta orientada por eventos do capítulo 2, esta camada constitui uma plataforma para a transposição de modelos genéricos. Nesta camada não se encontram indícios do modelo que será simulado. As camadas seguintes se apoiarão sobre as primitivas por ela oferecidas a fim de construir um ambiente para a simulação de políticas de balanceamento de carga.

Na próxima camada, descrita na seção 5.3, estão presentes os elementos relacionados ao problema de balanceamento. Valendo-se das primitivas de simulação fornecidas pela camada precedente, esta camada é a que efetivamente implementa as funcionalidades descritas pelo capítulo 4.

As camadas seguintes simplesmente agregam funcionalidades extras que, conquanto sejam prescindíveis, facilitam e agilizam a tarefa de confecção de um estudo de desempenho. Uma, descrita na seção 5.4, oferece ao analista um conjunto de políticas de balanceamento clássicas já implementadas, o que facilita a elaboração de estudos comparativos. A outra, descrita na seção 5.5, implementa uma interface gráfica para a interação com o ambiente. Por fim, a última seção enumera alguns dos porquês referentes às decisões de implementação tomadas.

5.1 A Arquitetura do Ambiente Robin Hood

O ambiente Robin Hood foi implementado em C++, inicialmente sobre a plataforma computacional Linux/Intel, sendo, em seguida, portado para a plataforma SunOS/Sparc. Estruturalmente, o ambiente foi concebido em camadas. De acordo com esta forma de organização, a camada superior agrega funcionalidades e serviços adicionais aos serviços prestados pela camada inferior, oferecendo-os, agora, por sua

vez, à camada subseqüente. Tal estrutura coaduna perfeitamente com os preceitos propalados pela orientação por objetos, principalmente no que se refere aos mecanismos de extensão por herança e de redefinição de funcionalidades através de métodos virtuais, uma vez que cada camada pode ser encarada como um repertório de classes, que são utilizadas, estendidas ou têm, até mesmo, parte de sua funcionalidade redefinida pelas camadas superiores.

A longo de sua elaboração, o ambiente Robin Hood foi sendo alterado de forma a comportar um número crescente de facilidades para o analista. Inicialmente, em sua primeira versão, o ambiente possuía apenas duas camadas, configuração mínima para a descrição e análise de políticas de balanceamento. Versões ulteriores acrescentaram a estas camadas outras que, muito embora fossem prescindíveis, ampliavam as facilidades oferecidas ao analista, seja através de uma interface mais amigável, seja através da disponibilização de políticas de balanceamento previamente implementadas. Os parágrafos abaixo apresentarão uma sucinta visão de cada uma das versões pelas quais o ambiente passou. As seções seguintes procurarão detalhar cada uma das camadas que compõem a versão final do ambiente.

A primeira versão do ambiente Robin Hood era composta somente pela sobreposição de duas camadas. Na primeira camada, situada na base, doravante referida como **camada SIM**, estão inclusos **o simulador** e as demais classes a ele correlatas. A camada SIM, ou seja, as classes que a integram, constituem um simulador orientado por objetos e dirigido por eventos de propósito geral. Não há nesta camada nenhuma indicativa sobre o modelo específico que será simulado. A camada SIM se preocupa somente em oferecer um *framework*, com o qual podem ser construídos e simulados modelos reais.

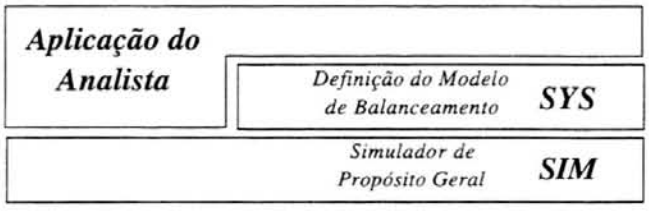


FIGURA 5.1 - Primeira versão do ambiente Robin Hood

Na camada superior, doravante referida como **SYS**, estão inclusos os elementos específicos à definição do modelo a ser simulado. Integram esta camada as classes concernentes à definição de um modelo apropriado para a descrição de **políticas de balanceamento de carga**. O modelo delineado pela camada SYS se estrutura sobre as primitivas genéricas de simulação fornecidas pela camada inferior, SIM. Em verdade, como veremos adiante, a camada SYS estende as noções de unidade de simulação (`SimUnit`) e modelo de simulação (`SimModel`) herdadas de classes da camada SIM, bem como estabelece os inter-relacionamentos existentes entre as unidades de simulação que, porventura, haja estendido.

A aplicação do analista funciona como uma camada adicional, que, empregando as abstrações de balanceamento disponibilizadas pela camada inferior, SYS, dá concretude às noções que foram, por esta camada, propositadamente omitidas ou desconsideradas. Compõe a aplicação do analista um conjunto mínimo de extensões às classes disponibilizadas pela camada SYS. Caso lhe aprouve, é facultado ao analista a

possibilidade de redefinir as entidades de balanceamento que lhe são oferecidas: (*i*) a **fonte**, responsável pela geração da demanda; (*ii*) e o **nodo**, incumbido de processar a carga de trabalho, como também de controlar o balanceamento do sistema. Via de regra, as primitivas fornecidas pela camada SYS são suficientes ao analista, não havendo porque sua aplicação interagir diretamente com a camada SIM. Todavia, embora infreqüente, caso julgue necessário, o analista pode recorrer a entidades que inexistem na camada SYS, sendo-lhe possível empregar, para tal propósito, primitivas de mais baixo nível, oferecidas pela camada SIM. A figura 5.1 retrata a configuração da primeira versão do ambiente.

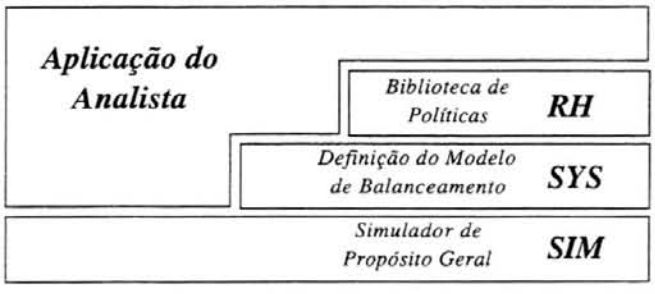


FIGURA 5.2 - Segunda versão do ambiente Robin Hood

A segunda versão, às duas camadas anteriores, SIM e SYS, acrescentou uma terceira, **RH**. Esta camada oferece ao analista uma biblioteca com várias políticas de balanceamento clássicas já implementadas. Aqui, não há diferenças entre uma política de balanceamento concebida pelo analista e uma implementada pela camada RH, exceto o fato de que esta já se encontra, de antemão, disponível. A existência da camada RH possibilita ao analista, sem que despenda esforços extras, dispor de políticas prontas para estudos comparativos. As políticas criadas pelo analista eventualmente constituiriam uma camada similar à RH, compiladas em uma biblioteca personalizada de políticas prontas. A figura 5.2 apresenta a segunda versão do ambiente, onde se pode notar a presença da nova camada.

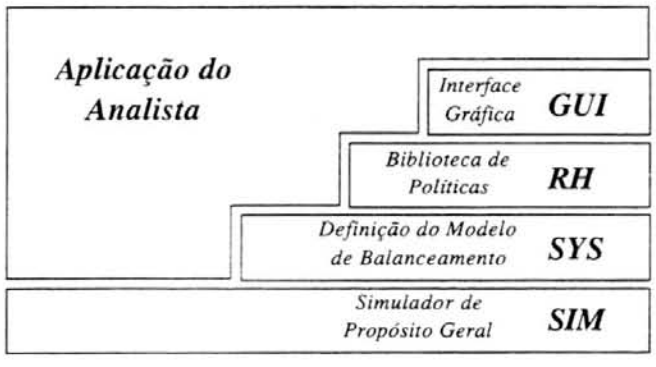


FIGURA 5.3 - Terceira versão do ambiente Robin Hood

Por fim, a terceira versão do ambiente — a atual — incluiu uma última camada, sobreposta a todas as anteriormente referidas, a camada **GUI** (*Graphical User Interface*). Esta camada implementa uma interface gráfica amigável para o analista, que objetiva esconder as minudências de implementação cujo conhecimento é completamente dispensável e agilizar os processos de confecção e análise de políticas de balanceamento. A figura 5.3 apresenta a versão atual do ambiente Robin Hood.

5.2 A Camada SIM — O Núcleo de Simulação

Como havíamos mencionado na seção precedente, a camada SIM funciona como um *framework* para o desenvolvimento e a simulação de modelos. Esta camada implementa um simulador dirigido por eventos de propósito geral e disponibiliza para a camada superior primitivas para a manipulação deste simulador e classes abstratas básicas, através de extensões das quais, via mecanismos de herança, as camadas superiores construirão seus modelos.

A literatura sobre simulação orientada por objetos apresenta, com fartura, exemplos de núcleos de simulação (ou camadas) de propósito geral. Entre estes, podemos citar [MAK 91][VAU 91][MAL 93][ZHE 93]. O projeto aqui apresentado para a camada SIM foi extraído, com algumas simplificações, das idéias contidas em [VAU 91]. Entre os objetivos últimos postulados para esta dissertação, não figuram o projeto e o desenvolvimento de um novo núcleo de simulação. O núcleo de simulação ocupa, no contexto desta dissertação, um papel secundário: o de servir exclusivamente de suporte à construção de um modelo de balanceamento. Não se justificaria, portanto, o dispêndio de esforços adicionais destinados à pesquisa e confecção de um núcleo de simulação original. O núcleo de simulação proposto em [VAU 91] se adequa perfeitamente às exigências que teríamos de uma ferramenta similar, eis o motivo de o escolhermos. O restante desta seção procurará expor, em linhas gerais, os principais atributos deste núcleo de simulação. Para informações suplementares, remetemos o leitor à já mencionada referência bibliográfica.

Quatro são as principais classes que compõem a camada SIM: **`Simulator`**, **`SimEvent`**, **`SimUnit`** e **`SimModel`**. Como já exposto na seção 2.6 e subseções, o exercício de um modelo de simulação discreto se dá graças à execução ordenada de procedimentos associados à ocorrência de eventos. A ocorrência de um evento pode acarretar, por sua vez, o agendamento de novos eventos para um tempo futuro, ou mesmo cancelar, antes que ocorram, eventos já agendados. Como todo evento deve ser agendado por um outro que o antecede, é necessário para que a simulação tenha início que alguns eventos sejam previamente estipulados em um período anterior à execução dos eventos. Estes ***eventos iniciais***, ao agendarem outros, darão continuidade ao exercício do modelo. A simulação tem o seu término quando não houver mais eventos pendentes para serem executados ou quando o tempo de simulação ultrapassar um dado limiar.

Cumpre à classe `Simulator` manter uma lista cronologicamente ordenada dos eventos pendentes e, um a um, ir executando os procedimento a eles associados, atualizando, à medida que estes vão sendo executados, o relógio de simulação (implementar um algoritmo para uma simulação discreta dirigida por eventos, como visto na seção 2.6.1). A lista de eventos é implementada por uma classe auxiliar, **`TimeQ`**, que permite a inserção ordenada, busca e remoção de eventos. O organograma relacionado à figura 5.4 retrata os relacionamentos de **herança** — representados por uma seta que parte da classe básica rumo à classe derivada — e **pertinência** — representados por uma linha contínua que une a classe proprietária às suas classes integrantes, dispostas dentro de um retângulo — existentes entre as classes que compõem a camada SIM.

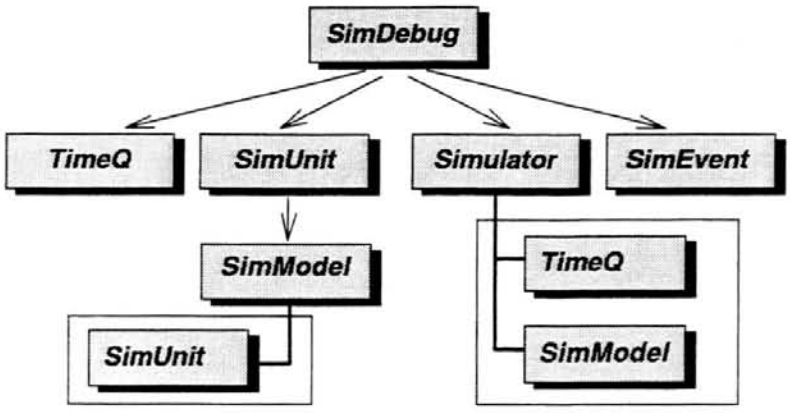


FIGURA 5.4 - Hierarquia de classes da camada SIM (simplificada)

A interface oferecida pela classe `Simulator` inclui métodos para: (*i*) iniciar o exercício do modelo; (*ii*) agendar um evento para um tempo futuro, relativo ao tempo presente (ocorra Ev_x daqui a s segundos); (*iii*) agendar um evento para um tempo futuro absoluto (ocorra Ev_x no minuto m e segundo s); (*iv*) cancelar a ocorrência de um evento antes agendado (cancele Ev_x); (*v*) recuperar o relógio de simulação.

Os eventos, representados pela classe `SimEvent`, possuem como interface apenas dois métodos virtuais: (*i*) um, que será executado no momento da ocorrência do evento; (*ii*) e o outro, que será executado caso o evento, porventura, seja cancelado. Estes métodos serão reescritos por extensões presentes na camada superior, SYS, onde lhes será atribuída uma real funcionalidade.

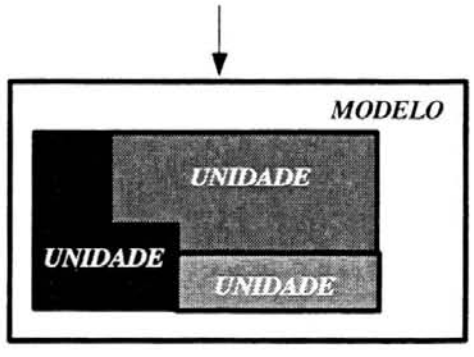


FIGURA 5.5 - Construção hierárquica do modelo

Resta falarmos dos conceitos de unidade de simulação e de modelo de simulação, representados, respectivamente, pelas classes `SimUnit` e `SimModel`. Tais conceitos dizem respeito à capacidade de construirmos, de **forma hierárquica**, através do repertório de classes da camada SIM, modelos de simulação. A **unidade de simulação** (classe `SimUnit`) pode ser compreendida como uma entidade simples, indivisível e de menor escala, com a combinação das quais serão construídos os modelos de simulação. O **modelo de simulação** (classe `SimModel`) pode ser encarado com uma unidade complexa, constituída de várias unidades de simulação ou mesmo de vários outros modelos de simulação.

Ao definirmos um modelo de simulação, combinando algumas unidades elementares de simulação, estamos, em verdade, construindo uma caixa preta, um bloco

fechado, que poderá ser utilizado futuramente na composição de outros modelos, sem que se precise conhecer o que há por dentro dele, como se unidade elementar fosse. A figura 5.5 procura aclarar um pouco a precisão destes conceitos. Nela, podemos ver um modelo, elaborado a partir de unidades elementares de simulação, **aparentar** externamente uma unidade elementar.

A versatilidade em combinarmos unidades de simulação a fim de compormos modelos — ou ainda de reutilizarmos, para o mesmo fim, modelos já construídos, e não apenas unidades elementares, num misto de modelos complexos e unidades elementares — é uma propriedade natural advinda da forma como tais conceitos, de modelo e de unidade, foram transpostos para a linguagem orientada por objetos que escolhemos. Como o organograma relacionado à figura 5.4 demonstra, a classe `SimModel` é derivada (ou herdeira) da classe básica `SimUnit`. De acordo com o paradigma de orientação por objetos, esta relação de herança faz com que um objeto `SimModel`, para todos os efeitos, se comporte como se `SimUnit` fosse, isto é, aqueles métodos que esperam receber como parâmetro um objeto `SimUnit` não se queixarão em receber, em troca — aliás, nem perceberão — um objeto `SimModel`. Destarte, a classe que define o modelo de simulação (`SimModel`) pode ser vista pelas demais de uma maneira dúbia: como modelo e como unidade elementar.

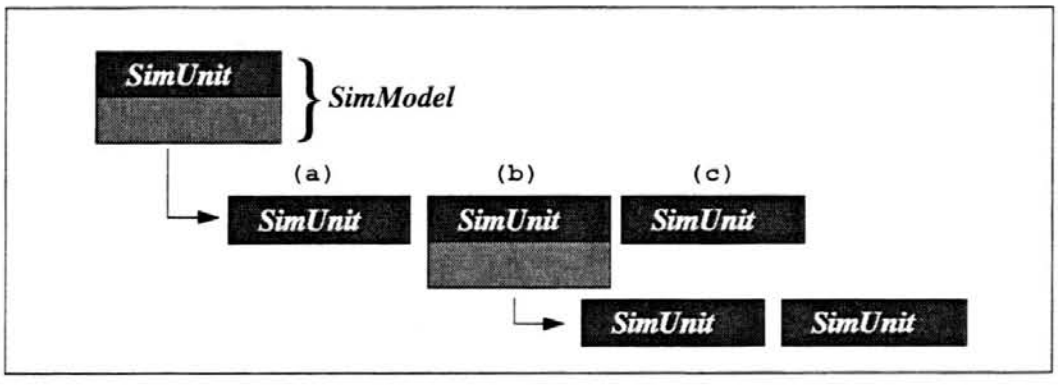


FIGURA 5.6 - Classes SimUnit / SimModel

A exemplo do que antes expressara o organograma 5.4, a figura 5.6 deixa também patente a relação de herança existente entre as classes `SimUnit` (classe básica) e `SimModel` (classe derivada ou herdeira). A relação de pertinência que envolve a classe `SimModel`, somente sugerida pelo organograma, é aqui retratada com maior ênfase. Como pode ser visto na figura 5.6, a classe `SimModel` possui uma lista de objetos da classe `SimUnit`. Os objetos apontados por esta lista constituem efetivamente o modelo (letras a, b, c). Vale aqui recordarmos a ressalva feita: os objetos apontados, que compõem o modelo, não são necessariamente unidades elementares de simulação (objetos `SimUnit`). São **enxergados** como se o fossem. Todavia poderiam muito bem ser — e aqui entra a construção hierárquica de um modelo — unidades complexas (objetos `SimModel`) passando-se por unidades elementares (letra b). Estas unidades complexas, por sua vez, eventualmente possuiriam também uma lista própria de objetos apontados, e quem sabe, nesta lista, outras unidades complexas. Este cascateamento, onde um objeto `SimModel` é constituído de várias unidades 'elementares', mas não sabemos, ao certo, se tais unidades elementares realmente o são, ou se escondem por

trás de si a composição de um outro modelo, sustenta e permite a construção hierárquica de modelos.

A invocação de um método de um objeto `SimModel` se propagará para todos as unidades 'elementares' que o constituem, a fim de que as partes que integram o modelo se apercebam das operações aplicadas sobre ele. Situações há em que esta propagação poderá se dar em vários níveis: de um modelo para as suas unidades constituintes, destas, caso constituam outros modelos, para as suas, e assim sucessivamente. Recomendamos que o leitor interessado se detenha um pouco sobre os protótipos das classes `SimModel`, `SimUnit`, `Simulator` e `SimEvent` fornecidos em um anexo, ao fim do texto. Embora esta leitura não seja fundamental, ela auxiliará a compreender grande parte do que aqui se procura explicar.

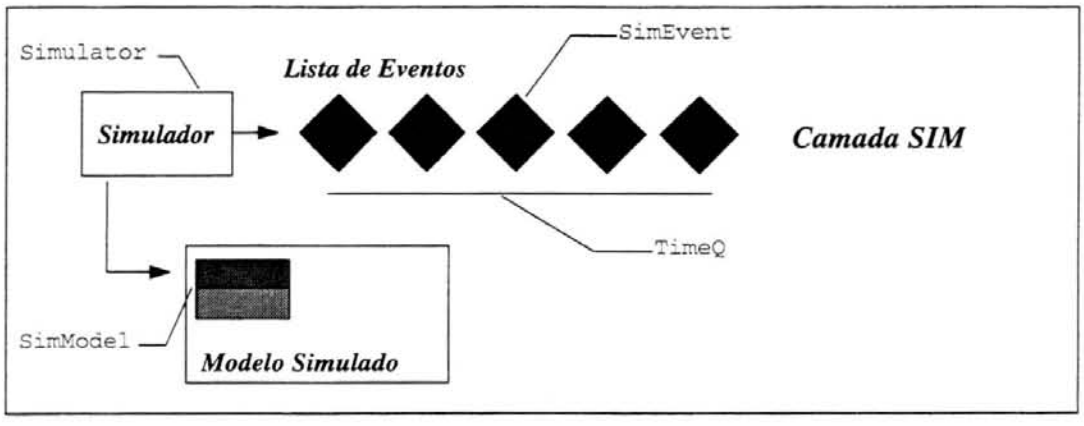


FIGURA 5.7 - Simulador (Camada SIM)

Conforme apresentado pelo organograma da figura 5.4, além da lista de eventos cronologicamente ordenada (`TimeQ`), integra o simulador (`Simulator`) o modelo a ser exercitado. Este modelo (`SimModel`) não é *a priori* conhecido pelo simulador. Será efetivamente construído através de extensões feitas pela camada superior, SYS, às classes `SimUnit` e `SimModel`. O modelo associado ao simulador esconderá por detrás de si toda uma hierarquia de unidades de simulação e modelos de simulação, a exemplo do que nos parágrafos precedentes sobre isto se falara. A figura 5.7 retrata a composição do simulador. Na próxima seção esta figura será refeita de maneira a comportar as extensões ao modelo implementadas pela camada SYS.

O funcionamento do simulador se resume a uma seqüência bem simples de passos:

1. ***Invoca o método virtual*** `reset()` ***do modelo a ser simulado*** — A invocação deste método será propagada a todas as unidades que compõem o modelo. O simulador não precisa saber (e efetivamente não sabe) quem são estas unidades, ou mesmo quantas são. Ao invocar este método solicita ao modelo a inicialização de suas estruturas de dados — alocação de memória, atribuição de valores iniciais para as variáveis, *etc.* Cabe ao modelo propagar a invocação do método às suas unidades constituintes, passando-lhes a tarefa de continuar com a propagação, caso eventualmente sejam unidades complexas. Ao final, o método `reset()` atingirá todas as unidades

elementares que constituem o modelo, ainda que sejam estas completamente desconhecidas por parte do simulador.

2. ***Invoca o método virtual*** `start()` ***do modelo a ser simulado*** — De forma análoga ao que foi dito no passo anterior, não obstante a ignorância do simulador sobre a real constituição do modelo, a invocação deste método será propagada a todas as unidades elementares que compõem o modelo. Esta invocação tem como propósito avisar ao modelo (e por conseguinte, a todas as unidades elementares) do início da simulação. O agendamento dos ***eventos iniciais*** ocorre graças a esta invocação, quando as unidades elementares, ao terem este método invocado, se decidirão por agendar a ocorrência de eventos futuros. A execução dos procedimentos associados aos eventos iniciais (passo 3) darão prosseguimento à simulação. Geralmente estes eventos agendam outros para ocorrer, e assim indefinidamente, até um tempo futuro determinado, quando a simulação é forçosamente interrompida, havendo ou não eventos pendentes.

3. ***Enquanto houver eventos pendentes e o tempo de simulação for inferior a um determinado valor:***

   3.1 Extrai o próximo evento pendente da lista de eventos

   3.2 Atualiza o relógio de simulação para o tempo de ocorrência deste evento

   3.3 Executa o método virtual `doEvent()` ***associado ao evento.***

5.3 A Camada SYS — O Modelo de Balanceamento

Empregando as facilidades fornecidas pela camada SIM, a camada SYS procura construir um modelo adequado à descrição e análise de políticas de balanceamento. Nesta camada, estão implementados os conceitos relacionados diretamente ao problema modelado, tais como: carga de trabalho, fonte, nodo, módulo de balanceamento de carga, mensagens, custos de migração e de envio de mensagens, *etc*.

Duas são as principais abstrações definidas por esta camada: (*i*) a **fonte**, incumbida de gerar, segundo uma determinada distribuição de tempo entre chegadas, a carga de trabalho; (*ii*) o **nodo**, responsável pela execução da carga de trabalho que lhe é imputada, assim como, por proceder com a redistribuição da carga do sistema nos momentos em que julgar apropriado.

O organograma relacionado à figura 5.8 retrata as relações de herança e pertinência existentes entre as principais classes que integram a camada SYS. Neste organograma, as classes `SysSource` e `SysNode` implementam respectivamente a fonte e o nodo.

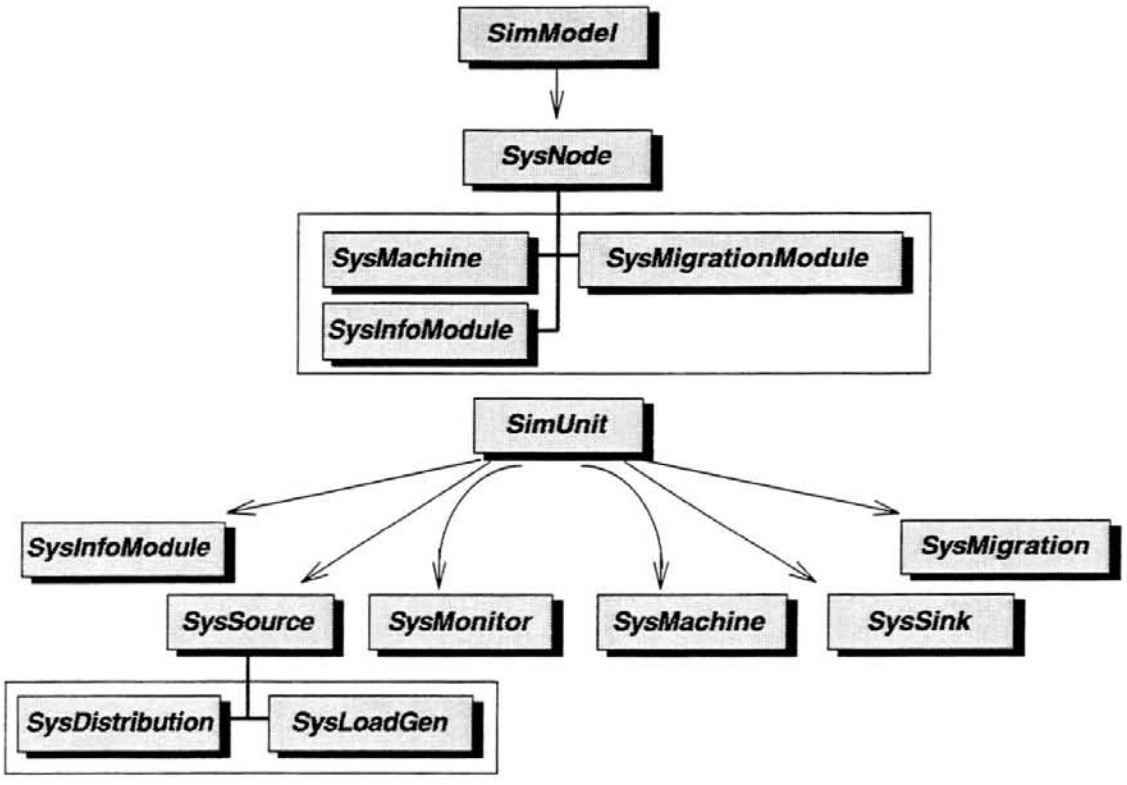


FIGURA 5.8 - Hierarquia de classes da camada SIM (simplificada)

Como pode ser visto, a fonte (classe `SysSource`) é composta por duas outras classes auxiliares: `SysDistribution` e `SysLoadGen`. Cabe à primeira destas classes determinar **a distribuição de tempo entre as chegadas** das cargas, e à segunda especificar **a demanda** associada à carga que chega. O algoritmo seguinte descreve o papel desempenhado pela fonte durante o exercício do modelo.

Procedimento associado ao evento ATIVAÇÃODAFONTE:

1. ***Obtém a carga de trabalho que será enviada ao nodo através do método*** `getWorkLoad()` ***da classe*** `SysLoadGen`.

2. ***Agenda para ocorrer agora o evento CHEGADADECARGA, e associa, à ocorrência deste evento, a carga de trabalho obtida no passo anterior.***

3. ***Agenda para ocorrer daqui a s segundos o evento ATIVAÇÃODAFONTE, onde s é o valor retornado pelo método*** `getX()` ***da classe*** `SysDistribution`.

A fonte é invocada quando o evento ATIVAÇÃODAFONTE ocorre. Este evento assinala o momento em que uma carga **deve** chegar ao sistema. A fonte solicita, então, ao método `getWorkLoad()` a criação de uma nova carga de trabalho, e agenda a **efetiva** chegada desta carga para ocorrer instantaneamente (evento CHEGADADECARGA). Em seguida, a fonte determina, através do método `getX()`, que retorna o comprimento do intervalo de tempo entre chegadas, qual é o próximo instante em que outra carga deve chegar ao sistema, e agenda para ocorrer novamente aí o evento ATIVAÇÃODAFONTE.

Entre as ações relacionadas à ocorrência do evento ATIVAÇÃODAFONTE está o próprio agendamento de outro evento ATIVAÇÃODAFONTE. Com isto, uma ativação da fonte tem sempre como pressuposto seu agendamento pela ativação anterior. Uma pergunta sensata seria como, então, ocorre a primeira ativação, uma vez que ela deve prescindir deste agendamento anterior? A resposta a esta questão já foi dada na seção anterior. A invocação do método virtual `start()` (passo 2), pelo simulador, se propaga a todas as unidades de simulação que integram o modelo. A fonte, pelo fato de ser uma unidade de simulação, herdeira da classe `SimUnit` (vide organograma relacionado à figura 5.8), e integrar a construção do modelo, terá, portanto, invocado o seu método `start()`. Através da invocação deste método, a fonte agendará para ocorrer o primeiro evento ATIVAÇÃODAFONTE. Este primeiro desencadeará as ativações subseqüentes, de acordo com o algoritmo apresentado.

As classes que compõem a fonte, `SysDistribution` e `SysLoadGen`, não implementam nenhum dos métodos que oferecem. Apenas os definem. Estas classes funcionam como protótipo para a criação de classes herdeiras. Estabelecem uma interface, elencando métodos virtuais, mas deixam a implementação destes a cargo das classes que lhe são derivadas. A literatura sobre orientação por objetos comumente se refere a tais classes como 'classes postergadas' (*deferred classes*) [MEY 88] ou 'classes abstratas' (*abstract classes*) [STR 91][ELL 90].

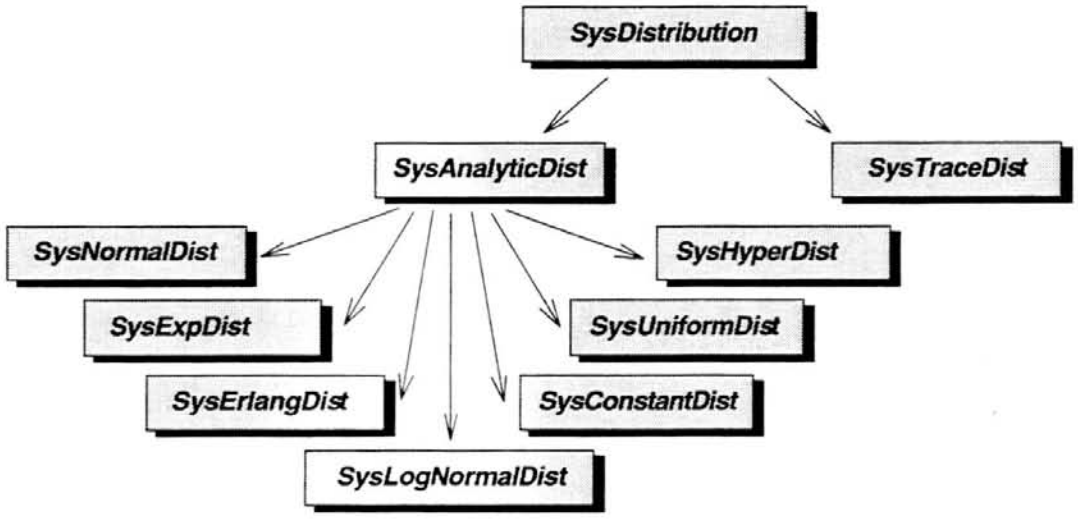


FIGURA 5.9 - Distribuições estatísticas

Destarte, a classe `SysDistribution` não implementa o método `getX()`. Quem efetivamente o faz são suas classes herdeiras. Cada uma destas pode fornecer uma implementação particular do método `getX()`. Com isto, da classe genérica e abstrata `SysDistribution`, seriam derivadas várias outras classes, cada uma, através de sua própria redefinição de `getX()`, implementando uma distribuição específica. O organograma relacionado à figura 5.9 ajuda a elucidar a questão. Nele podemos observar que, tendo a classe `SysDistribution` como raiz, surge uma família de classes herdeiras — `SysNormalDist`, `SysExpDist`, `SysErlangDist`, *etc.* —, cada qual implementando uma distribuição particular — normal (ou gaussiana), exponencial, erlang, *etc.*

As classes que se originam de SysLoadGen, redefinindo cada qual, a seu modo, o método getWorkLoad() constituem, a exemplo de SysDistribution, um organograma similar. Podemos citar, a título de exemplo, as classes SysExpLoadGen, SysNormalLoadGen, SysTraceLoadGen que fornecem, como resposta à invocação do método getWorkLoad(), cargas em cuja demanda consta somente o tempo de processamento requerido, sendo este estipulado, respectivamente, de acordo com as distribuições exponencial, normal e mediante um arquivo de rastro.

A ligação entre a fonte e a classe que efetivamente implementa o método getX() (herdeira de SysDistribution) ou a que implementa o método getWorkLoad() (herdeira de SysLoadGen) será realizada posteriormente. Não é dado saber à fonte, *a priori*, qual classe herdeira realmente implementa os métodos virtuais que invoca. A fonte se limita a conhecer (e utilizar) a interface das classes abstratas que a compõem, SysDistribution e SysLoadGen, ignorando sua real implementação. A vinculação entre as classes herdeiras que efetivamente implementam os métodos getX() e getWorkLoad() e a fonte ocorrerá somente na última fase de confecção de uma política de balanceamento, quando o analista já tiver definido todos os parâmetros exigidos pelo ambiente, inclusive a própria política que deseja ver simulada.

Retomando o organograma anterior (figura 5.8), observamos que o nodo é composto por três outras classes auxiliares: SysMachine, SysInfoModule, SysMigrationModule. A primeira destas, SysMachine, representa a unidade de processamento associada ao nodo, sua CPU. Cumpre à classe SysMachine escalonar para a execução, segundo uma dada disciplina de fila, os processos que demandam por processamento no nodo. A classe SysInfoModule representa, ao que no capítulo anterior já nos referíramos, o Módulo de Informação e Balanceamento. Esta classe se incumbe de gerenciar o tráfego de informação sobre o estado de carga dos vizinhos, de decidir o momento adequado ao balanceamento do sistema e de perfazê-lo. Por fim, através da classe SysMigrationModule é que são estipulados os custos referentes à transferência de uma carga de uma máquina para a outra.

As extensões fornecidas pelo analista a estas classes, acrescidas da escolha acerca das distribuições que regerão o comportamento da fonte, determinarão a composição final do modelo de balanceamento a ser simulado. A figura 5.10 apresenta uma visão sintética das principais classes que integram a camada SYS, e de como extensões a estas classes são providas pelo usuário a fim de compor um modelo completo de balanceamento, passível de simulação. No restante desta seção, sempre que nos referirmos às classes que integram o nodo, SysMigrationModule, SysMachine e SysInfoModule, estaremos também nos referindo às eventuais extensões a estas classes providas pelo analista.

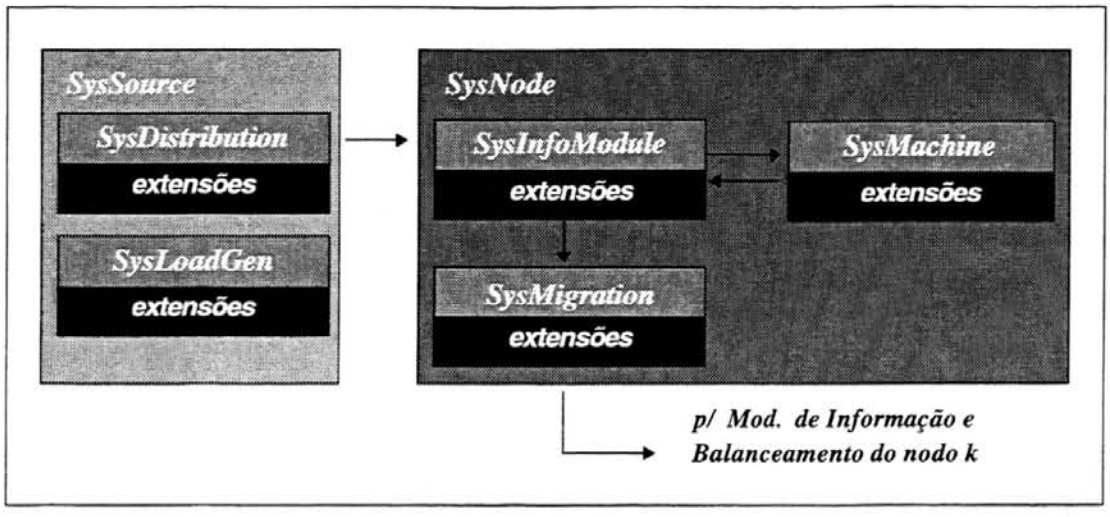


FIGURA 5.10 - Relacionamento entre as classes da camada SYS

É de se esperar que, a exemplo do que acontece com a classes que integram a fonte, também aqui, as classes que integram o nodo são despidas de real funcionalidade. Apenas fornecem uma interface, mas não chegam a implementar os métodos que disponibilizam. Isto é feito pelas classes herdeiras, que especializam o que antes fora colocado de forma genérica, imprimindo-lhe uma marca particular. A classe `SysMachine` possui como herdeiras as classes `SysCpuRoundRobin` e `SysCpuSharedServer`, cada qual implementando uma disciplina de fila própria: a primeira destas com uma fila circular de processos, onde, sucessivamente, cada um destes se apropria do processador da máquina por um *quantum* de tempo, ao fim do qual, forçosamente, o processo retorna para o final da fila; a segunda implementando um servidor compartilhado, caso limite da disciplina de fila anterior, quando o *quantum* de tempo facultado aos processos se aproxima de zero, e estes dividem o poder de processamento de uma forma matematicamente equivalente.

A classe `SysMigrationModule`, através de suas herdeiras, dá origem a um organograma similar ao apresentado para a classe `SysDistribution`: suas classes derivadas imputam à transferência de cargas custos determinados pelas principais distribuições estatísticas — normal, erlang, *etc*. Em verdade, as classes derivadas de `SysMigrationModule` empregam diretamente as facilidades fornecidas pelas herdeiras de `SysDistribution`.

A classe `SysInfoModule` possui como herdeira a classe `SysMsgInfoModule`, e esta a classe `SysTimerInfoModule`. Sucessivamente funcionalidades vão sendo agregadas, pelas extensões, à classe `SysInfoModule`. Sua herdeira imediata, `SysMsgInfoModule`, permite e exige, por parte de suas herdeiras, a definição de primitivas que controlem o fluxo de mensagens pela rede. Originalmente, a classe `SysInfoModule` não implementa nenhuma facilidade para o envio e recebimento de mensagens. A primeira vista isto pode parecer absurdo, mas políticas de balanceamento há que não necessitam de troca de informação para o seu funcionamento. Bons exemplos são a política de localização aleatória [ZHO 88], definida na seção 3.7.3.2, e políticas nas quais é arbitrariamente fixado para quem (ou de quem) enviarão (ou receberão) suas cargas em situação de sobrecarga (ou de ociosidade) [WAN 85].

SysTimerInfoModule, herdeira de SysMsgInfoModule, implementa a ativação periódica do Módulo de Informação e Balanceamento. Às facilidades referentes à troca de mensagens, herdadas de SysMsgInfoModule, é acrescentada um outra, que vincula a execução do Módulo, não somente à chegada ou partida de cargas (SysInfoModule), ou também ao recebimento de mensagens (SysMsgInfoModule), mas ainda a pulsos de um temporizador. De tempos em tempos, o Módulo é despertado a fim de que concretize as ações de balanceamento que almeja. Muitas políticas de balanceamento se enquadram aqui, pois exigem uma periódica intervenção do Módulo no sentido de reduzir as diferenças de carga entre os nodos. Vale frisar que sucessivas heranças agregam, mais e mais, funcionalidade à classe original. Assim, as facilidades definidas por SysMsgInfoModule quanto ao envio e recebimento de mensagens são transmitidas às suas herdeiras, entre elas a classe SysTimerInfoModule.

A tabela 5.1 ilustra a questão. Nela, as classes supracitadas são dispostas, em um crescente, de acordo com o número de facilidades que disponibilizam. A primeira, e a mais simples delas, da qual as demais derivam, é SysInfoModule. Em seguida temos SysMsgInfoModule, e, por fim, SysTimerInfoModule.

TABELA 5.1 - Classe SysInfoModule e derivadas

Módulo de Informação e Balanceamento	***Funcionalidades***	***Momentos de Ativação***	***Métodos Associados aos Momentos de Ativação***
SysInfoModule	• primitivas para a migração de cargas	• chegada de carga (*) • partida de carga (*)	• loadArrival(...) • loadDepart(...)
SysMsgInfoModule	• primitivas para a migração de cargas •primitivas para o envio e recebimento de mensagens	• chegada de carga (*) • partida de carga (*) • recebimento de mensagens (†)	• loadArrival(...) • loadDepart(...) • receiveMessage(...)
SysTimerInfoModule	• primitivas para a migração de cargas • primitivas para o envio e recebimento de mensagens • primitivas para o controle da temporização do módulo	• chegada de carga (*) • partida de carga (*) • recebimento de mensagens (†) • pulso do temporizador (†)	• loadArrival(...) • loadDepart(...) • receiveMessage(...) • timeExpired(...)

(*) Definição opcional
(†) Definição obrigatória

Fica livre para o analista escolher qual destas classes estenderá, na definição de sua política de balanceamento. Se a política concebida não precisa de informação sobre o restante do sistema, o que torna prescindível quaisquer primitivas de troca de mensagens, o analista sensatamente optará por definir o seu Módulo como uma extensão da classe SysInfoModule. Caso contrário, para um política na qual um nodo periodicamente verifica o estado de seus vizinhos, o analista optará por SysTimerInfoModule.

A carga que acaba de ingressar no nodo, seja gerada originariamente pela fonte associada a este nodo (evento CHEGADADECARGA), seja resultado de uma ação de balanceamento (evento TRANSFERÊNCIADECARGA), antes mesmo de ser enviada à CPU (classe SysMachine), é recebida pelo método loadArrival() do Módulo de

Informação e Balanceamento (`SysInfoModule`). Este método, como a tabela acima indica, não precisa necessariamente ser definido. A classe `SysInfoModule` possui uma definição para ele, na qual, a carga que ingressa é repassada diretamente para a CPU, não dando margem a ações de balanceamento. Caso, nas extensões que porventura fizer, o analista não redefina este método, assim ocorrerá para todas as cargas que chegarem ao nodo. De forma similar, o método `loadDepart()`, relacionado à partida das cargas, também possui uma definição, onde nenhuma ação é executada. Ambos podem, opcionalmente, ser reescritos pelas extensões do analista. Os demais métodos elencados na tabela acima, `receiveMessage()` e `timeExpired()`, não são definidos pelas classes que os disponibilizam, o que faz com que sua definição seja obrigatória por parte de quem queira estender tais classes.

As classes definidas pela camada SYS — ou suas herdeiras, presentes na aplicação do analista — são extensões feitas a unidades de simulação (`SimUnit`) e modelos de simulação (`SimModel`) da camada inferior. O simulador de propósito geral definido pela camada SIM exercitará, sem que o saiba, o modelo de balanceamento parcialmente definido pela camada SYS e complementado pela aplicação do analista.

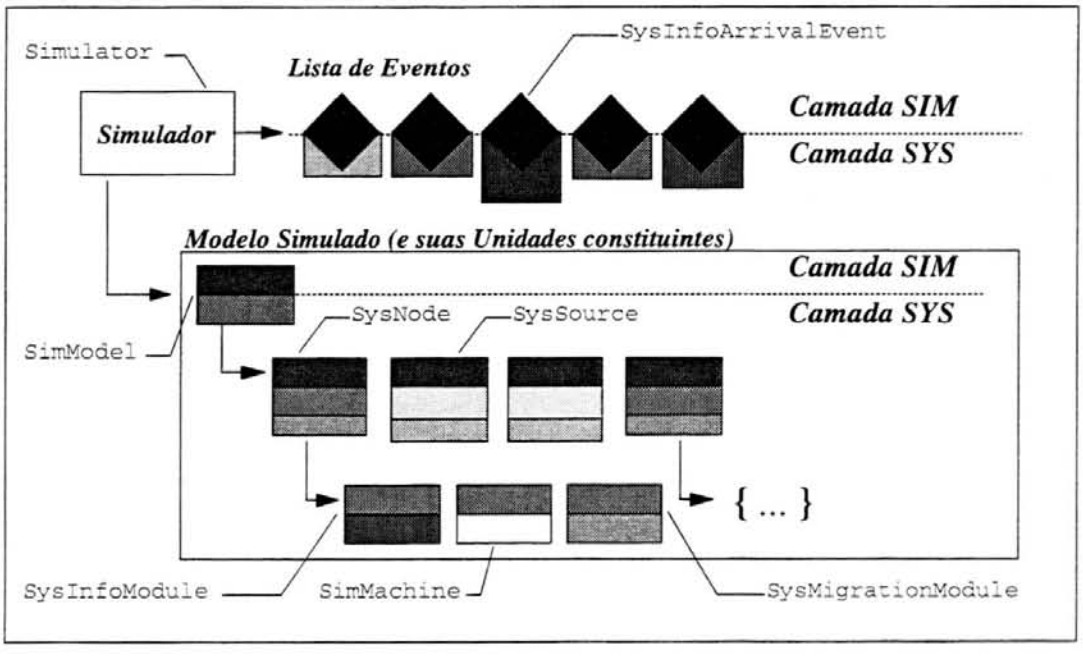


FIGURA 5.11 - Primitivas e entidades de balanceamento (Camada SIM/SYS)

Através de extensões feitas aos eventos (`SimEvent`), bem como de extensões feitas às unidades de simulação (`SimUnit`) e modelo de simulação (`SimModel`), a camada SYS foi construída. A figura 5.11 procura esquematizar o relacionamento existente entre as camadas. Por trás do modelo de simulação (objeto `SimModel`), apontado pelo simulador (objeto `Simulator`), está a efetiva construção da camada SYS: uma lista composta de unidades de modelo (tais como `SysNode`) e unidades de simulação (tais como `SysSource`), sobre as quais incidirão as operações associadas à ocorrência dos eventos estendidos (tais como CHEGADADECARGA — `SysInfoArrivalEvent`, ATIVAÇÃODAFONTE — `SysSouceEvent`, *etc.*).

A invocação dos métodos `reset()` e `start()` da classe `SimModel`, feita pelo simulador antes da execução dos eventos com o intuito de permitir a inicialização das

estruturas de dados e agendamento dos eventos iniciais, será propagada a todos as unidades constituintes do modelo (lista de objetos `SimUnit` apontados), e atingirá, graças aos mecanismos de redefinição de funções virtuais, todos as classes herdeiras de `SimUnit` que efetivamente implementam tais métodos (`SysSource`, `SysTimerInfoModule` e outras). O próximo passo dado pelo simulador, o ciclo de extração e execução dos eventos, promoverá a invocação dos métodos virtuais `doEvent()` associados a cada um deles, o que dará prosseguimento ao exercício do modelo, acarretando a manipulação do estado das entidades que o compõem e o agendamento de novos eventos.

A figura 5.11 apresenta como integrantes do modelo as classes definidas pela camada SYS: `SysSource` , `SysNode`, `SysInfoModule` e outras. O propósito desta figura é somente salientar os relacionamentos que existem entre as camadas SIM e SYS, pois, na prática, as classes que realmente integrarão o modelo serão as extensões definidas pelo analista às classes da camada SYS. A seção seguinte se deterá sobre a elaboração da aplicação do analista.

5.4 A Camada RH — Uma Biblioteca de Políticas

A camada RH é composta por uma biblioteca de políticas clássicas de balanceamento já implementadas. Tais políticas são extensões feitas às classes da camada SYS, e em nada diferem das extensões encontradas na aplicação do analista. O objetivo da camada RH é poupar o tempo do analista, fornecendo-lhe um conjunto de políticas já implementadas para a elaboração de estudos comparativos. O analista poderá também as tomar como exemplo na elaboração de suas próprias, aprendendo por elas a utilizar as facilidades disponibilizadas pelo ambiente Robin Hood.

Apresentaremos uma de tais políticas implementadas pela camada RH, e o que precisa ser feito para vê-la simulada. O exemplo serve simultaneamente a dois propósitos: o primeiro de familiarizar o leitor com as extensões contidas na aplicação do analista, uma vez que as políticas implementadas pela camada RH constituem aplicações prontas; o segundo, de reunir os conceitos vistos até então, compondo uma aplicação real, apesar de simples, e passível de simulação.

As tabelas de configuração introduzidas no parágrafo seguinte servem exclusivamente para guiar a efetiva simulação da política de balanceamento, definindo o ambiente computacional sob o qual esta política será simulada. Não fazem parte, portanto, da camada RH. Esta é composta somente pelo conjunto de classes (extensões) que implementam as políticas prontas que a camada disponibiliza. O intuito de apresentar tais tabelas é mostrar a seqüência de passos que precisam ser dados rumo, não somente à definição de uma política, mas também à sua simulação.

A figura 5.12, disposta abaixo, contêm dois arquivos de configuração utilizados pelo ambiente Robin Hood. Ambos os arquivos especificam um conjunto de parâmetros que determinará a maneira pela qual se efetuará a simulação de uma dada política de balanceamento.

Definem ambos uma topologia composta por quatro nodos, em que cada um deles se liga a todos os demais (asserções `Node(1){2 ...}`), e cujas fontes possuem,

invariavelmente, distribuições exponenciais com médias de, respectivamente, 20.0 e 16.0, para os intervalos entre chegadas de cargas e para o tempo de processamento (asserções `IATDistribution=...` e `LoadGenerator=...`).

```
// arquivo 'table-random'
// política aleatória (seção 3.7.3.2)
NumNodes = 4;
System {
  SimTime           =  8000.0;
  RHHomeDir         = "~/projeto";
  UserHomeDir       = "~/projeto/usr";
  SourceFiles       = "rhrandom.C";
  IncludeFiles      = "rhrandom.h";
  ExecutableName    = "random";
  NumPartitions     = 10;
  MonitoringQuantum= 1.0;
}
DefaultNode {
  Machine         = SysCpuRoundRobin(0.1);
  IATDistribution= SysExpDist(20.0);
  LoadGenerator   = SysExpLoadGen(16.0);
  BalancingModule= RHRandomPolicy(4,2);
  MigrationModule= SysConstantMigCost(0.1);
}
Node(1) { Neighbors = 2 3 4; }
Node(2) { Neighbors = 1 3 4; }
Node(3) { Neighbors = 1 2 4; }
Node(4) { Neighbors = 1 2 3; }
```

```
// arquivo 'table-nomig'
// não há ações de balanceamento
NumNodes = 4;
System {
  SimTime           =  8000.0;
  RHHomeDir         = "~/projeto";
  UserHomeDir       = "~/projeto/usr";
  ExecutableName    = "nomig";
  NumPartitions     = 10;
  MonitoringQuantum= 1.0;
}
DefaultNode {
  Machine         = SysCpuRoundRobin(0.1);
  IATDistribution= SysExpDist(20.0);
  LoadGenerator   = SysExpLoadGen(16.0);
  BalancingModule= SysInfoModule();
}
Node(1) { Neighbors = 2 3 4; }
Node(2) { Neighbors = 1 3 4; }
Node(3) { Neighbors = 1 2 4; }
Node(4) { Neighbors = 1 2 3; }
```

FIGURA 5.12 - Arquivos de configuração (política aleatória / sem migração)

Os arquivos variam somente quanto ao Módulo de Informação e Balanceamento (asserção `BalancingModule=...`) que estipulam. O primeiro destes arquivos, `table-random`, situado à esquerda da figura, atribui para o Módulo de Informação e Balanceamento a classe `RHRandomPolicy`, definida nos arquivos `rhrandom.h` e `rhrandom.C` (asserção `SourceFiles=...` e `IncludeFiles=...`). Tal classe (`RHRandomPolicy`) implementa uma política de balanceamento **aleatória**, à qual já antes aludíramos. Para maiores informações sobre esta política, remetemos o leitor à seção 3.7.3.2 [ZHO 88].

O segundo arquivo, `table-nomig`, situado à direita, não fornece extensão alguma às classes da camada SYS, atribuindo ao Módulo de Informação e Balanceamento a classe `SysInfoModule`. Não precisa, portanto, indicar arquivos fontes ou de cabeçalhos, onde se encontrariam eventuais extensões. De acordo com o que antes já disséramos na seção precedente, o método `loadArrival()` da classe `SysInfoModule` se limita a repassar imediatamente para a CPU a carga que acaba de ingressar, não dando espaço a ações de balanceamento. Com tal atribuição, este arquivo determina a simulação do sistema, sob as mesmas condições que as especificadas no arquivo anterior, só que sem a presença de qualquer tentativa de redistribuição de carga.

Os demais parâmetros se destinam a controlar a geração da massa de resultados (`NumPartitions=...` e `MonitoringQuantum=...`), a especificar o tempo de simulação (`SimTime=...`) e os diretórios do analista e do ambiente Robin Hood (`RHHomeDir=...` e `UserHomeDir=...`).

Os arquivos rhrandom.C e rhrandom.h, aos quais anteriormente se referira o arquivo table-random (lado esquerdo da figura 5.12) através das asserções SourceFiles=... e IncludeFiles=..., se encontram, ambos, contidos na figura 5.13 apresentada abaixo. Em tais arquivos estão definidas as extensões feitas às classes da camada SYS, com o propósito de completar e construir um modelo de balanceamento passível de simulação.

```
// arquivo 'rhrandom.h'
#include "sysworkload.h"
#include "sysinfo.h"

class RHRandomPolicy : public SysInfoModule {
   int upLoad;       // threshold value
   int upTransfer;   // maximun # of transfers
public:
   RHRandomPolicy(int upL, int upT):
      upLoad(upL), upTransfer(upT)  { ; }

   void  loadArrival(SysWorkLoad *load);
};

// arquivo 'rhrandom.C'
#include "rhrandom.h"
void RHRandomPolicy::loadArrival(SysWorkLoad *load)
{
   if((nLoadsInCpu() + 1   > upLoad) &&
      (load->timesMigrated < upTransfer))
      migrateTheLoad(irand(nNeighbors()), load);
   else
      passToCpu(load);
}
```

FIGURA 5.13 - Extensões às classes da camada SYS fornecidas pela aplicação do analista

O arquivo rhrandom.C implementa exclusivamente o método loadArrival(), invocado sempre que uma carga chega ao nodo. Quando isto ocorre, é verificado o estado de carga do nodo. Se o número de cargas que este possuir (nLoadsInCpu()), acrescido da carga que ingressa (+ 1), for superior à carga mínima necessária (upLoad), e ainda, o número de vezes que já foi anteriormente transferida a carga que chega (load->timesMigrated) não ultrapassar ao máximo permitido (upTransfer), então, terá aí lugar uma ação de redistribuição de carga. A carga que acaba de ingressar será transferida (migrateTheLoad(...)) para um dos vizinhos do nodo aleatoriamente escolhidos (irand(nNeighbors()) — onde a função irand(N) retorna um valor aleatório entre [1..N], no caso, o número de vizinhos do nodo).

Estabelecidas **as condições** sob as quais a política de balanceamento será simulada (através de um arquivo similar ao table-random) e definidas **as extensões** que a implementarão (através de arquivos similares aos rhrandom.C e rhrandom.h), o analista poderá dar início ao exercício do modelo, bastando para isto executar o aplicativo mk.sh. Este aplicativo interpretará a tabela de configurações definida pelo usuário, gerará com as especificações nela contida a rotina inicial (main()), compilará os arquivos de extensão definidos pelo analista, os ligará com a biblioteca rh (onde estão

definidas as classes SYS e SIM) e, por fim, dará início ao exercício do modelo. A rotina `main()` promoverá a montagem do ambiente computacional sob o qual a política de balanceamento atuará.

Espera-se que, com o exemplo apresentado, o leitor tenha uma forte noção da seqüência de ações necessária à simulação da política que deseja. Esta seqüência corresponde diretamente à exposta na seção 4.9.

Além da política aleatória, várias outras (sobre as quais nos detivemos nas seções 3.7.1.3, 3.7.2.3 e 4.6) se encontram implementadas na camada RH:

- Política de limiar (*threshold policy*) [ZHO 88];
- Política de menor-valor (*lowest policy*) [ZHO 88];
- Política global (*global policy*) [ZHO 88];
- Política central (*central policy*) [ZHO 88];
- Política UDI1/UDI2/UDI3/UDI4 (*unidirectional incremental transfers policies*) [CAS 88];
- Política estocástica (*stochastic learning automata*) [STA 85];

5.5 A Camada GUI — A Interface Gráfica do Ambiente

A camada GUI foi criada com o propósito de facilitar o desenvolvimento e a simulação de políticas de balanceamento, fornecendo ao analista uma interface gráfica de fácil manuseio. As várias etapas pelas quais tem de passar o analista na simulação de seu modelo podem ser, através das facilidades da interface gráfica, executadas de maneira intuitiva. A figura 5.14 apresenta a tela principal da interface.

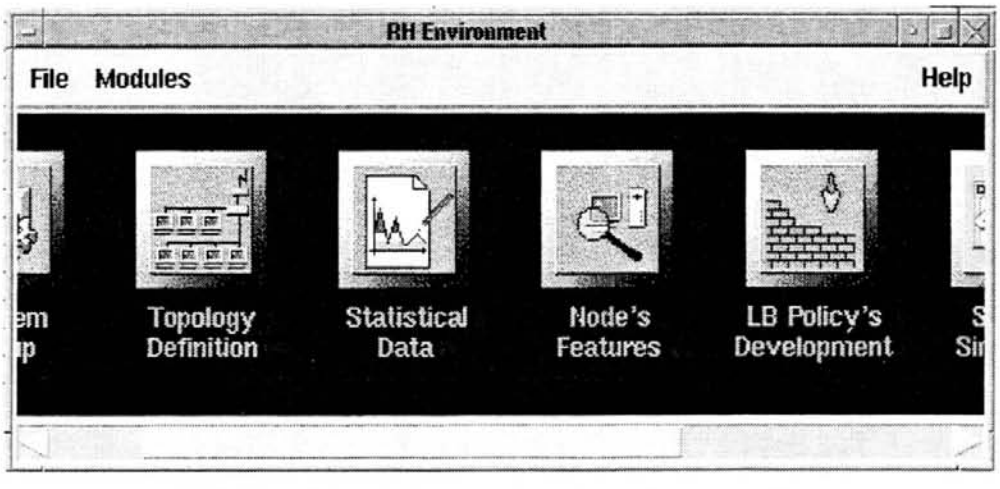


FIGURA 5.14 - Tela principal do ambiente gráfico

Cada um dos botões (ou ícones) que compõem a tela principal conduz o usuário à especificação de um dos elemento necessários à construção e simulação do modelo.

Desta forma, ao navegar pelos botões da tela principal, o analista irá construindo, parte a parte, a política de balanceamento que almeja e o ambiente onde esta será executada. O último botão disparará o exercício da política escolhida no ambiente computacional determinado pelo analista (topologia, descrição da carga submetida ao sistema, custos, *etc.*), e fornecerá os resultados desta simulação. De forma sucinta, as subseções seguintes descreverão as funcionalidades que se escondem por detrás dos principais botões da tela principal.

5.5.1 Biblioteca de Topologias

O primeiro botão de relevância (***Topology Definition***) conduz o analista à tela onde poderá ser especificada a topologia do sistema. Esta tela lhe apresenta um conjunto de topologias já definidas que podem ser reaproveitadas (***System Topologies***).

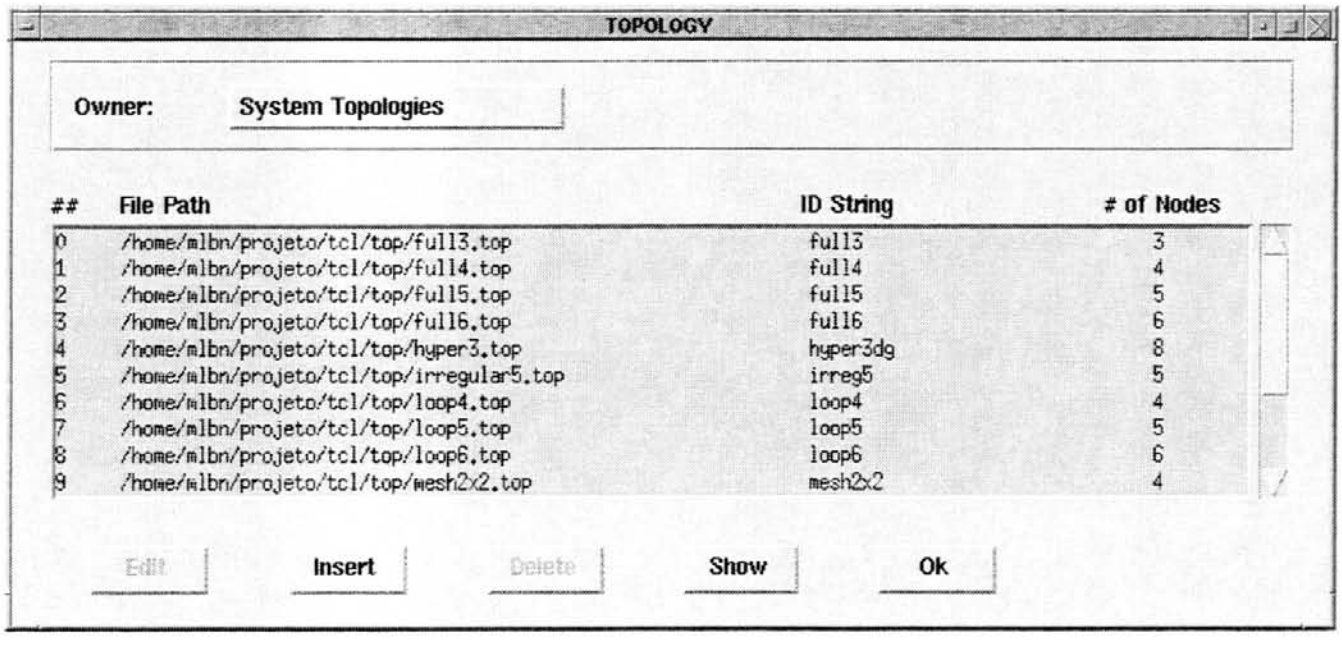


FIGURA 5.15 - Tela para a definição de topologias

Caso a topologia que deseja não conste entre as que o ambiente oferece, o analista poderá definir a sua. A nova topologia definida será, então, validada, verificando se não apresenta inconsistências. Após validada, será inserida no conjunto de topologias definidas pelo analista (***User Topologies***), para futuro uso. Tem-se, com isto, uma biblioteca permanente de topologias, à qual o analista poderá acrescentar as suas próprias, junto das demais topologias oferecidas pelo ambiente Robin Hood. A figura 5.15 apresenta a tela destinada à especificação da topologia.

5.5.2 Massa de resultados fornecidos

O segundo botão conduz o analista à tela onde serão escolhidos os resultados de saída fornecidos pela simulação da política. O analista poderá optar entre resultados que procuram, através de médias e desvios estatísticos, resumir o comportamento do ambiente durante todo o intervalo de simulação (***Range Simulation***), ou resultados que,

coletados periodicamente e sem pretensão alguma de síntese, descrevem como o sistema evolui ao longo deste intervalo (***Specific Simulation***). Para estes, é facultado ao analista descrever o intervalo com o qual as mensurações serão feitas, e para aqueles os índices de desempenho de relevância (utilização de CPU médio, tempo de resposta médio, tempo de espera em filas médio, *etc.*). A figura 5.16 apresenta a tela destinada à escolha dos resultados fornecidos.

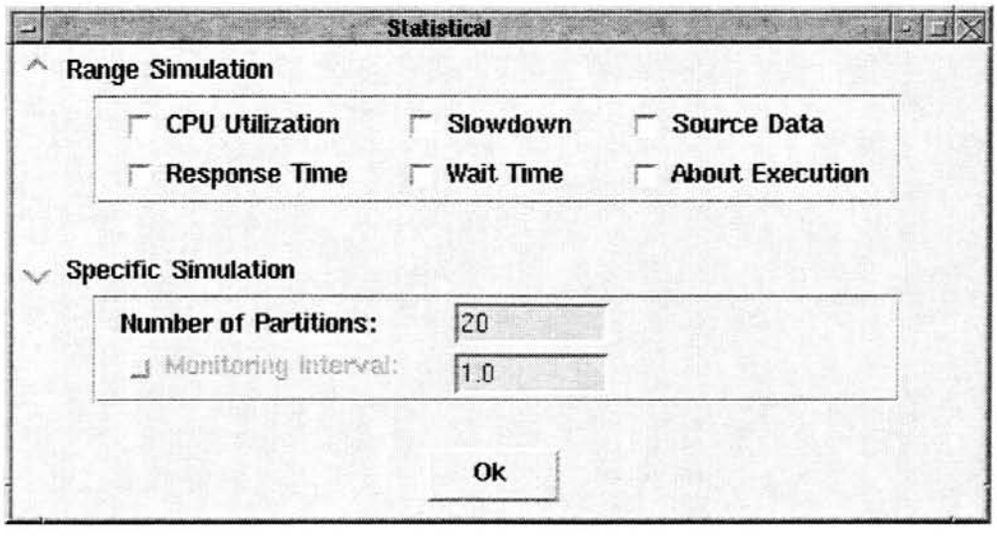


FIGURA 5.16 - Tela para escolha dos resultados fornecidos

5.5.3 Biblioteca de Configurações para o Ambiente Computacional

O terceiro botão conduz o analista à tela onde serão definidas as configurações para a carga submetida ao sistema e para os Módulos de Informação e Balanceamento em cada um dos nodos. A exemplo da tela onde eram definidas as topologias, esta permite ao analista que reaproveite configurações definidas pelo ambiente Robin Hood, ou mesmo suas próprias configurações, criadas anteriormente. Desta forma, o analista poderá criar uma "situação" de carga alta vinculada a uma topologia escolhida, e salvar esta configuração para uso futuro. Poderá, de forma, similar, definir uma biblioteca de configurações tais como: "carga média", "carga alta", "carga baixa", *etc.*

Aqui, o analista deverá indicar, para cada um dos nodos, qual classe será responsável pela determinação do intervalo de tempo entre chegadas (asserção `IATDistribution=...` da seção precedente) e qual será pela especificação da demanda (asserção `LoadGenerator=...`), bem como os parâmetros que estas classes receberão.

Deverá indicar ainda: o Módulo de Informação e Balanceamento do nodo (asserção `BalancingModule=...`); os custos relativos à migração das cargas (asserção `MigrationModule=...`); e a classe responsável pelo efetivo processamento das cargas (asserção `Machine=...`). Como pode se ver, a interface gráfica funciona simplesmente como um anteparo, pois exige a definição dos mesmos parâmetros requeridos pelas tabelas de configuração da seção precedente. Torna dispensável, no

entanto, o conhecimento da sintaxe associada a tais tabelas, bem como concentra todas as atividades relativas à simulação de uma política. A figura 5.17 apresenta uma tela intermediária, subseqüente à diretamente invocada pelo botão da tela principal, onde tais parâmetros são especificados.

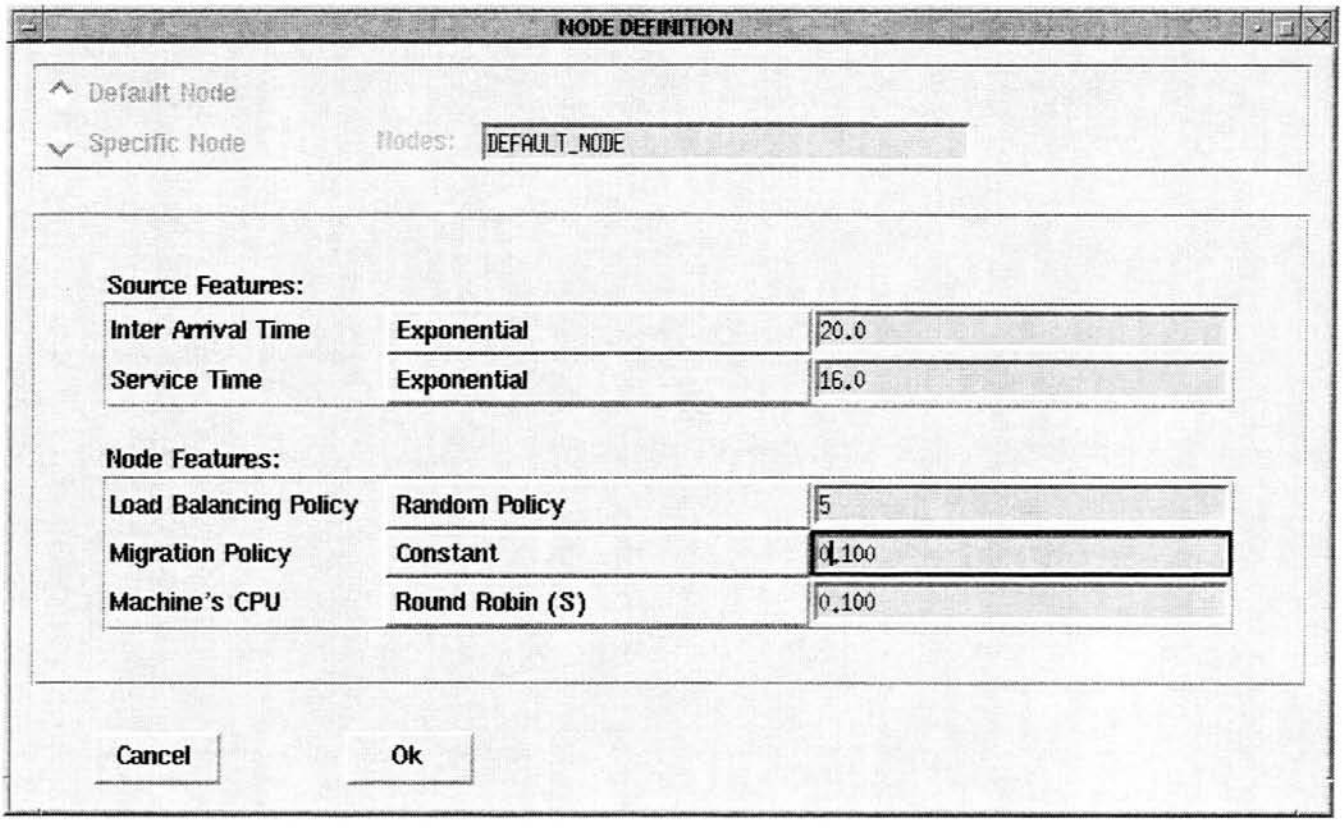


FIGURA 5.17 - Tela para especificação das características dos nodos

5.6 Decisões de Projeto

Ambas as camadas da ferramenta, SIM e SYS, foram implementada em C++. Várias foram as razões de projeto que levaram à escolha de tal linguagem, entre as quais, pode-se citar:

- **disponibilidade**, a linguagem C++ é extremamente difundida e conhecida, e existe um compilador C++ para a maioria das plataformas encontradas;
- **portabilidade**, o código fonte da ferramenta não explora características peculiares a nenhum sistema operacional específico, sendo facilmente portável para qualquer plataforma em que exista um compilador C++;
- **facilidade de uso**, sendo plausível supor que o analista já conheça e domine a linguagem em questão, não se exige, aqui, por parte dele, o conhecimento de uma linguagem nova;
- **eficiência**, o desempenho apresentado pela linguagem C++ é bem superior ao apresentado por linguagens de simulação convencionais [DOY 90];

- **extensibilidade**, mecanismos intrínsecos à filosofia de orientação por objetos, tais como herança, encapsulamento de dados, funções virtuais e sobrecarga de funções, fazem com que as definições de procedimentos e dados sejam reutilizáveis e extensíveis.

5.7 Conclusão

Este capítulo descreveu cada uma das camadas que integram a arquitetura do ambiente Robin Hood. Procurou-se, ao longo do capítulo, enfatizar o estilo de 'construção hierárquica de modelos', que somente se fez possível graças, em primeiro plano, à concepção que foi dada para a camada SIM (extraída de [VAU 91]) e, subsidiariamente, às propriedades oriundas da orientação por objetos. Tal estilo permitiu, não somente que o ambiente, como um todo, pudesse ser construído sobre o núcleo de simulação implementado pela camada SIM, como também que fosse facultado ao analista a possibilidade de redefinir os principais elementos do modelo, adicionando sua próprias extensões. A hierarquização permitiu que, paulatinamente, complexidades crescentes fossem adicionadas ao modelo, sem que se precisasse alterar o que já houvera sido definido.

A arquitetura do ambiente Robin Hood pôde, com isto, ser elegantemente construída. Entre as qualidades do ambiente estão, sem dúvida, o seu alto grau de flexibilidade em acomodar alterações ou extensões e a clareza que perpassa toda o projeto de engenharia de software. Camada por camada, separando elementos díspares e integrando elementos símiles, através da sobreposição crescente de serviços prestados, foi concebido e projetado o ambiente.

O capítulo seguinte apresentará as conclusões finais acerca deste trabalho, enumerando as conquistas por ele realizadas, e as inevitáveis falhas que possui. Serão elencados ainda um conjunto de metas, para o futuro desenvolvimento do ambiente aqui proposto.

6 Considerações Finais

O presente trabalho se propôs desenvolver um ambiente (uma ferramenta) destinado à avaliação de políticas de balanceamento de carga. O primeiro passo rumo à confecção do ambiente foi a definição de um modelo apropriado para o problema em questão. Modelo este que contemplasse os principais aspectos e particularidades inerentes ao problema de balanceamento. Uma vasta revisão bibliográfica foi feita (capítulo 3), objetivando reconhecer as principais contribuições dentro deste segmento de pesquisa e, principalmente, visando compor um modelo de balanceamento a partir dos elementos considerados relevantes pelos autores dos referidos trabalhos na elaborações de seus modelos.

Identificados os elementos que integrariam o modelo — entidades constituintes, tais como nodo, carga, fonte; medidas de relevância para análise do desempenho do sistema, tais como tempo de resposta, *slowdown*; distribuições apropriadas para descrever a evolução dos estados do sistema — o passo seguinte foi esboçar as funcionalidades que seriam oferecidas pelo ambiente e antever uma maneira concreta de implementá-las.

As funcionalidades oferecidas pelo ambiente Robin Hood, nome escolhido para a ferramenta de análise que desenvolvemos, permitem que o analista de desempenho facilmente descreva a política de balanceamento que quer ver simulada e o sistema onde tal política atuará (capítulo 4). Poucas linhas são necessárias para codificar a política de balanceamento (seção 4.7 e 5.4) e, de forma automática, um completo conjunto de resultados sobre o comportamento desta política é gerado pelo ambiente (seção 4.6).

Com o propósito ainda de facilitar a definição e a avaliação de políticas de balanceamento, uma interface gráfica foi adicionada ao ambiente (seção 5.5). Através da interface gráfica, o analista pode armazenar e recuperar as novas políticas de balanceamento que desenvolver (seção 5.5.3). O mesmo acontece para a definição de novas topologias (seção 5.5.1). Detalhes sobre como foi implementado o ambiente Robin Hood podem ser encontrados no capítulo 5 e nos anexos ao final do texto.

Cumpre fazermos aqui uma análise crítica sobre o trabalho desenvolvido, salientando as inevitáveis falhas encontradas durante o percurso e os aspectos positivos que o mesmo apresenta. Como aspectos positivos, podemos apontar:

1. O ambiente Robin Hood permite analisar uma dada política de balanceamento frente à variação de um conjunto de fatores bem mais rico que o possibilitado por estudos de balanceamento feitos diretamente sobre um sistema real — nos quais processos reais são efetivamente transferidos, a exemplo de [DIK 89] [ZHO 87] [LEM 93][KUN 91] [BAR 85][MIL 94].

   Tais estudos apresentam a desvantagem inerente à técnica de análise que adotam, qual seja de não poder se desvincular da topologia física do sistema em questão. Os custos de comunicação e de transferência são forçosamente determinados pelo meio físico, pelo sistema operacional e pelos aplicativos associados às facilidades de balanceamento. Não há como, em tais estudos, arbitrariamente estabelecer valores para tais custos. A topologia física não pode também ser alterada. Muito embora seja

possível construir uma topologia lógica sobre a topologia física, os verdadeiros custos ainda serão determinados por esta. Outra desvantagem, quanto aos estudos feitos sobre sistemas reais, diz respeito à limitada capacidade de repetir os experimentos, dada a extrema dificuldade de se reestabelecerem condições idênticas às originais.

O ambiente Robin Hood não padece de nenhum dos males elencados: por extrair os índices do modelo construído, e não diretamente do sistema real, não possui vínculo com nenhuma topologia física. Tanto a topologia 'física' quanto os custos de comunicação podem ser estabelecidos ao gosto do analista. Fica a cargo deste determinar como se interconectarão os nodos do sistema. Permite ainda a repetição fidedigna dos experimentos.

À alegação de que, ao confiar o dimensionamento dos custos a um modelo, os resultados fornecidos pelo ambiente difeririam daqueles oriundos de um sistema real, se contrapõe a justificativa de que, caso se queira, tais custos podem advir de um arquivo de rastro. Observações feitas sobre a transferência de dados em um sistema real podem alimentar, ou servir de base ao cálculo, dos custos considerados pelo ambiente. Obviamente não se pretende aqui confrontar a validade dos dados produzidos pelo ambiente Robin Hood com os produzidos diretamente sobre um sistema real. Almeja-se apenas assinalar a possibilidade de se aperfeiçoarem as abstrações feitas pelo modelo, aproximando o comportamento de tais abstrações do constatado em elementos reais;

2. Os resultados apresentados pelo ambiente Robin Hood compõem um panorama completo acerca do desempenho final do sistema, bem como da evolução deste ao longo do tempo. O ambiente permite ainda a análise estática [CAS 88] do comportamento das políticas (seção 4.6). Tal análise revela importantes informações sobre estabilidade e tempo de estabilização, que, muitas das vezes, ficam mascaradas ou escondidas por análises convencionais;

3. O desenvolvimento de **políticas hierárquicas** é permitido e em nada difere do desenvolvimento de outros tipos de política. De igual forma, podem ser simuladas:

    - **políticas centralizadas** — um único detém o conhecimento acerca dos demais — ou **distribuídas** — os nodos são equivalentes e gerenciam uma parcela equivalente de informação [ZHO 88];

    - **síncronas** — a política de balanceamento evolui por fases cadenciadas: (*i*) troca de informação entre os nodos e sincronização dos mesmos e (*ii*) transferência de cargas — ou **assíncronas** — são aceitos atrasos de comunicação arbitrariamente grandes e não há qualquer sincronização entre os nodos [SON 95] [CAS 88];

4. O ambiente pode ser facilmente estendido. Novas distribuições estatísticas podem ser acrescentadas (como herdeiras de `SysDistribution`) e, de forma equivalente, novas classes geradoras de carga (como herdeiras de `SysLoadGen`). A demanda pode ser detalhada pelo analista, e nela incluído o consumo de recursos adicionais;

5. O analista, utilizando a biblioteca da camada RH, pode facilmente traçar paralelos entre a sua política e várias outras. Pode também compor a sua própria biblioteca de políticas;

6. Por fim, o ambiente Robin Hood aparece em um segmento de pesquisa onde foram encontradas poucas ferramentas de análise destinadas ao mesmo fim. Entre as pesquisadas, a única de propósito similar é [HEI 92b]. As demais, ou abrangem um campo de atuação bem maior, objetivando a avaliação de sistemas computacionais paralelos [PEA81], ou perseguem um propósito diverso, como [SNY 84], ferramenta que auxilia encontrar um mapeamento adequado de um programa para um hardware específico.

Como aspectos negativos assinalamos:

1. Alguns custos ainda não foram introduzidos no protótipo (e. g. custo de ativação dos módulos, custo de processamento para o envio e recebimento de mensagens e custo para a obtenção de informação de carga);

2. Somente um único paradigma de comunicação é fornecido, a saber o envio e o recebimento **assíncronos** de mensagens. Aqui caberia ainda implementar outras formas de comunicação entre os módulos de balanceamento.

3. A interface gráfica não possui ainda um módulo especialmente dedicado à visualização dos resultados de desempenho.

6.1 Trabalhos Futuros

A partir das falhas diagnosticadas, e visando também a ampliação de certas facilidades já disponibilizadas pelo ambiente, um rol de metas foi estabelecido para o futuro desenvolvimento do trabalho. Entre os objetivos futuros estipulados podemos citar:

- Desenvolvimento de ferramentas de visualização adequadas ao problema de balanceamento. Parte dos dados gerados assume a forma textual apenas. Restaria incorporar um novo módulo à interface gráfica, com o exclusivo propósito de transpor os resultados coletados para uma forma gráfica mais apropriada e intuitiva. Gráficos que assinalassem a evolução do estado de carga dos nodos ao longo do tempo, gráficos sobre o número de mensagens trocadas pelos nodos e sobre o impacto dos custos de comunicação sobre o desempenho seriam importantes extensões a acrescentar;

- Inclusão de outros paradigmas de comunicação entre os processos, tais como envio e recebimento síncronos e envio assíncrono/recebimento síncrono de mensagens.

- Inclusão de custos adicionais ainda não considerados pelo protótipo, tais como custo de ativação dos módulos, custo de processamento para o envio e recebimento de mensagens e custo para a obtenção de informação de carga;

- Ampliação da biblioteca de políticas de balanceamento disponibilizada pela camada RH.

Anexo 1 Classes da camada SIM

Classe SimDebug

```
#if !defined(SIMDEBUG_H)
#define SIMDEBUG_H

#define NAMESIZE 20
class SimDebug {
private:
   static long int   qty;    // # of SimDebug objects in the system
   static long int   id;     // # of created SimDebug objects

   char        debugName[NAMESIZE];  // individual name and...
   long int    debugId;              // ...id for debug purposes

public:
   SimDebug(char* = "_UNDEFINED_");
   virtual ~SimDebug();

   long int      nCreatedObjects()       { return id;  }
   long int      nExistingObjects()      { return qty; }
   virtual void  simPrintDebug();
   virtual void  simErrorDebug(char *);
};

#endif
```

Classe SimEvent

```
#if !defined(SIMEVENT_H)
#define SIMEVENT_H

class SimEvent : public SimDebug {
   // no data members
public:
   SimEvent(char *n):  SimDebug(n)  { ; }

   virtual void doEvent()       { ; }
   virtual void cancelEvent()   { ; }
   virtual void shutdown()      { ; }
};

#endif
```

Classe SimUnit

```
#if !defined(SIMUNIT_H)
#define SIMUNIT_H

class SimUnit : public SimDebug {
protected:
   Simulator*   simulator;
   SimModel*    model;

public:
   SimUnit(char *n):   SimDebug(n),
      simulator(NULL), model(NULL)                { ; }

   // to be implemented in derived classes
   virtual void  simStart()                       { ; }
   virtual void  simReset()                       { ; }
   virtual void  simPrintStatistics(FILE *)       { ; }
   virtual void  simSetSimulator(Simulator *s)    { simulator = s; }

   // to set the appropriate model
   void          simSetModel(SimModel* m) { model = m; }

   // for getting some info
   SimModel*   getModel()     const   { return model;     }
   Simulator*  getSimulator() const   { return simulator; }
};

#endif
```

Classe SimModel

```
#if !defined(SIMMODEL_H)
#define SIMMODEL_H

class Simulator;

class SimModel : public SimUnit {
protected:
   SimUnitList subunits;

public:
   SimModel(char *n): SimUnit(n)     { ; }
  ~SimModel();

   // methods overriden from SimUnit
   void  simStart();
   void  simReset();
   void  simPrintStatistics(FILE *);
   void  simSetSimulator(Simulator *);

   // SimUnitList handling
   void  simAddUnit(SimUnit *u)    {
                                      u->simSetModel(this);
                                      subunits.insertFirst(u);
                                    }
   void  simRemoveUnit(SimUnit *u) {
                                      u->simSetModel(NULL);
                                      subunits.removeThe(u);
                                    }

   // debug
   void  simPrintDebug();
};
#endif
```

Classe Simulator

```
#if !defined(SIMULATOR_H)
#define SIMULATOR_H

class Simulator : public SimDebug {
private:
   SimTime    maxTime;      // the upper bound
   SimTime    clock;        // system's clock
   TimeQ      timeq;        // list of events
   SimModel  *model;        // the 'top model'

   // statistics
   long int   nEvents;      // # of events executed
public:
  ~Simulator();
   Simulator(char *n, SimTime max): SimDebug(n),
      maxTime(max), clock(0), model(NULL), nEvents(0) { ; }

   Simulator(char *n, SimTime max, SimModel *m): SimDebug(n),
      maxTime(max), clock(0), model(m), nEvents(0)    { ; }

   // main functions
   void simStart()                  { model->simStart(); }
   void simReset()                  { model->simReset(); }
   void simSetModel(SimModel *m)    { model = m;         }
   void simPrintStatistics(FILE *);
   void simulate();

   // for event scheduling
void scheduleEventNow(SimEvent *ev)                  {timeq.insertAt(ev, clock);   }
void scheduleEventAbsolute(SimEvent *ev, SimTime t)  {timeq.insertAt(ev, t);       }
void scheduleEventRelative(SimEvent *ev, SimTime dt) {timeq.insertAt(ev, clock-dt);}

   // for event canceling
   void cancelEvent(SimEvent *ev);

   // for getting some info
   SimModel*  getModel()        const  { return model; }
   SimTime    getTime()         const  { return clock;  }
   SimEvent*  getCurrentEvent() const  { return timeq.getCurrentEvent(); }

   // debug
   void    simPrintDebug() { SimDebug::simPrintDebug();  model->simPrintDebug(); }
};

#endif
```

Classe TimeQ

```
#if !defined(TIMEQ_H)
#define TIMEQ_H

class QElement : public SimDebug {
private:
   QElement    *next;
   SimEvent    *event;
   SimTime      at;

public:
   QElement(SimEvent *ev, SimTime t) : SimDebug("QELEMENT"),
      next(NULL), event(ev), at(t)              { ; }
   friend class TimeQ;
};

class TimeQ : public SimDebug{
private:
   QElement    *first;
   QElement    *last;

public:
   TimeQ() : SimDebug("TIMEQ"), first(NULL), last(NULL)    { ; }
  ~TimeQ();

   // handling functions
   SimEvent*   removeCurrent()          { return removeThe(first->event); }
   SimEvent*   removeThe(SimEvent *);
   void        insertAt(SimEvent *, SimTime);
   Boolean     existMoreEvents() const  { return (first?TRUE:FALSE);  }

   // for getting some info
   SimEvent*   getCurrentEvent() const  { return first->event; }
   SimTime     getCurrentTime()  const  { return first->at;    }

   // debug
   void simPrintDebug();
};

#endif
```

Anexo 2 Classes da camada SYS

Classe SysSource

```
#if !defined(SYSSOURCE_H)
#define SYSSOURCE_H

class SysSource: public SimUnit {
private:
   SysInfoModule      *infoModule;  // conected to...
   SysDistribution    *iat;         // to get the inter-arrival time
   SysLoadGen         *gen;         // to get the real 'load'

   // statistics
   int               nArrival;
   int               maxArrivals;
public:
   ~SysSource();
   SysSource(char *n, SysInfoModule *info,
             SysDistribution *t, SysLoadGen *g):
      SimUnit(n),  infoModule(info), iat(t), gen(g),
      nArrival(0), maxArrivals(INT_MAX)                { ; }


   SysSource(char *n, int max, SysInfoModule *info,
             SysDistribution *t, SysLoadGen *g):
      SimUnit(n),  infoModule(info), iat(t), gen(g),
      nArrival(0), maxArrivals(max)                    { ; }

   // needful redefinitons
   void   simStart();

   // handling function
   void   activate();

   // for getting some info
   void   simPrintStatistics(FILE *);
};

#endif
```

Class SysWorkLoad

```
#if !defined(SYSWORKLOAD_H)
#define SYSWORKLOAD_H

// the 'concept' of load in the system
class SysWorkLoad : public SimDebug {
public:
   SimTime     startTime;
   SimTime     finishTime;
   SimTime     age;
   SimTime     duration;
   int         timesMigrated;

   SysWorkLoad(SimTime dur): SimDebug("SYSWORKLOAD"),
      startTime(UNKNOWN), finishTime(UNKNOWN),
      age(0), duration(dur), timesMigrated(0)       { ; }

};

#endif
```

Classe SysDistribution e derivadas

```
#if !defined(SYSDIST_H)
#define SYSDIST_H

class SysDistribution : public SimDebug {
   // no data members
public:
   SysDistribution(char *n = "SYSDISTRIBUTION") : SimDebug(n)     { ; }
   virtual double  getX()=0;
};

#define UNUSED   0.0          // sometimes stdDev is needless...

// analytical classes:
//    -- pseudo-random number generation --
class SysAnalyticDist : public SysDistribution {
protected :
   SimRandomGen  gen;
   double        average;
   double        stdDev;
   uint          seed;

public:
   // (av, std, --[seed]--)
   SysAnalyticDist(char *n, double av, double std):
      SysDistribution(n), average(av),
      stdDev(std), seed((uint) rand())     { gen.srandom(seed); }

   // (av, std, seed)
   SysAnalyticDist(char *n, double av, double std, uint sd):
      SysDistribution(n), average(av),
      stdDev(std), seed(sd)                { gen.srandom(seed); }
};

// Constant Distribution
class SysConstantDist : public SysAnalyticDist {
public:
   SysConstantDist(double av):                    // (av, --[std]--)
      SysAnalyticDist("SYSCONSTANTDIST", av, UNUSED)      { ; }

   double getX();
};

// Uniform Distribution
class SysUniformDist : public SysAnalyticDist {
public:
   SysUniformDist(double av):                   // (av, --[std]--)
      SysAnalyticDist("SYSUNIFORMDIST", av, UNUSED)       { ; }

   SysUniformDist(double av, uint seed):        // (av, --[std]--, seed)
      SysAnalyticDist("SYSUNIFORMDIST", av, UNUSED, seed) { ; }

   double getX();
};


// Exponential Distribution
class SysExpDist : public SysAnalyticDist {
public:
   SysExpDist(double av):                     // (av, --[std]--)
      SysAnalyticDist("SYSEXPDIST", av, UNUSED)           { ; }

   SysExpDist(double av, uint seed):          // (av, --[std]--, seed)
      SysAnalyticDist("SYSEXPDIST", av, UNUSED, seed)     { ; }

   double getX();
};

// Normal (Gaussian) Distribution
class SysNormalDist: public SysAnalyticDist {
public:
```

```
   SysNormalDist(double av, double std):            // (av, std)
      SysAnalyticDist("SYSNORMALDIST", av, std)          { ; }

   SysNormalDist(double av, double std, uint seed): // (av, std, seed)
      SysAnalyticDist("SYSNORMALDIST", av, std, seed)    { ; }

   double getX();
};


// Hyperexponential Distribution
class SysHyperDist:  public SysAnalyticDist {
public:
   SysHyperDist(double av, double std):             // (av, std)
      SysAnalyticDist("SYSHYPERDIST", av, std)           { ; }

   SysHyperDist(double av, double std, uint seed):  // (av, std, seed)
      SysAnalyticDist("SYSHYPERDIST", av, std, seed)     { ; }

   double getX();
};


// Erlang Distribution
class SysErlangDist:  public SysAnalyticDist {
public:
   SysErlangDist(double av, double std):            // (av, std)
      SysAnalyticDist("SYSERLANGDIST", av, std)          { ; }

   SysErlangDist(double av, double std, uint seed): // (av, std, seed)
      SysAnalyticDist("SYSERLANGDIST", av, std, seed)    { ; }

   double getX();
};

#endif
```

Classe SysLoadGen e derivadas

```
// responsible for generate a 'useful load'
class SysLoadGen : public SimDebug {
   // no data members
public:
   SysLoadGen(char *n = "SYSLOADGEN") : SimDebug(n) { ; }
   virtual SysWorkLoad*   getWorkLoad()=0;
};


// Load generation with 'constant' service time
class SysConstantLoadGen:  public SysLoadGen {
private:
   SysConstantDist  st;                    // service time for load

public:
   SysConstantLoadGen(SimTime average):
      SysLoadGen("SYSCONSTANTLOADGEN"), st(average)      { ; }

   SysWorkLoad*   getWorkLoad()  { return new SysWorkLoad(st.getX()); }
};


// Load generation with 'uniform' service time
class SysUniformLoadGen:  public SysLoadGen {
private:
   SysUniformDist   st;

public:
   SysUniformLoadGen(SimTime average):
      SysLoadGen("SYSUNIFORMLOADGEN"), st(average)       { ; }

   SysUniformLoadGen(SimTime average, uint seed):
      SysLoadGen("SYSUNIFORMLOADGEN"), st(average, seed) { ; }

   SysWorkLoad*   getWorkLoad()  { return new SysWorkLoad(st.getX()); }
};


// Load generation with 'exponential' service time
class SysExpLoadGen : public SysLoadGen {
private:
   SysExpDist       st;

public:
   SysExpLoadGen(SimTime average):
      SysLoadGen("SYSEXPLOADGEN"), st(average)        { ; }

   SysExpLoadGen(SimTime average, uint seed):
      SysLoadGen("SYSEXPLOADGEN"), st(average, seed)  { ; }

   SysWorkLoad*   getWorkLoad()  { return new SysWorkLoad(st.getX()); }
};

// Load generation with 'normal' service time
class SysNormalLoadGen:  public SysLoadGen {
private:
   SysNormalDist    st;

public:
   SysNormalLoadGen(SimTime average, SimTime stdDev):
      SysLoadGen("SYSNORMALLOADGEN"), st(average, stdDev)           { ; }

   SysNormalLoadGen(SimTime average, SimTime stdDev, uint seed):
      SysLoadGen("SYSNORMALLOADGEN"), st(average, stdDev, seed)     { ; }

   SysWorkLoad*   getWorkLoad()  { return new SysWorkLoad(st.getX()); }
};


// Load generation with 'hyperexponential' service time
```

```
class SysHyperLoadGen : public SysLoadGen {
private:
   SysHyperDist      st;

public:
   SysHyperLoadGen(SimTime average, SimTime stdDev):
      SysLoadGen("SYSHYPERLOADGEN"), st(average, stdDev)            { ; }

   SysHyperLoadGen(SimTime average, SimTime stdDev, uint seed):
      SysLoadGen("SYSHYPERLOADGEN"), st(average, stdDev, seed)      { ; }

   SysWorkLoad*   getWorkLoad()  { return new SysWorkLoad(st.getX()); }
};


// Load generation with 'erlang'' service time
class SysErlangLoadGen:  public SysLoadGen {
private:
   SysErlangDist     st;

public:
   SysErlangLoadGen(SimTime average, SimTime stdDev):
      SysLoadGen("SYSERALNGLOADGEN"), st(average, stdDev)           { ; }

   SysErlangLoadGen(SimTime average, SimTime stdDev, uint seed):
      SysLoadGen("SYSERLANGLOADGEN"), st(average, stdDev, seed)     { ; }

   SysWorkLoad*   getWorkLoad()  { return new SysWorkLoad(st.getX()); }
};

#endif
```

Classe SysNode

```
#if !defined(SYSNODE_H)
#define SYSNODE_H

class SysNode : public SimModel {
public:
   int                  globalId;
   SysSink              *sink;
   SysInfoModule        *infoModule;
   SysMigrationModule   *migModule;

   // node's information
   SysMachine            *myCPU;
   SysInfoModule        **myNeighbors;
   int                    nNeighbors;

   // destructor function
   ~SysNode();

   // constructor functions
   SysNode(char*, int id, SysSink*, SysMachine*, SysInfoModule*);
   SysNode(char*, int id, SysSink*, SysMachine*, SysInfoModule*,
           SysMigrationModule*);

   // to be set 'a posteriori'
   void setNeighbors(SysInfoModule **, int n);
};

#endif
```

Classe SysModule

```
class SysModule : public SimUnit {
protected:
   SysNode            *myNode;
   SysMachine         *myCPU;

public:
   // to SysMachine
   SysLoadIndex       myLoadIndex()       { return myCPU->getLoadIndex();     }
   int                nLoadsInCpu()       { return myCPU->nLoadsInCpu();      }

   SysWorkLoad*       startIteration()    { return myCPU->startIteration();   }
   SysWorkLoad*       getNextLoad()       { return myCPU->getNextLoad();      }
   Boolean            canIterateYet()     { return myCPU->canIterateYet();    }

   // there (inNeighbor), what is the local ID to my neighbor (thisNeighbor)?
   int mapMyLocalId(int ofNode, int inNode);

   SysModule(char *n): SimUnit(n), myNode(NULL), myCPU(NULL)  { ; }
   void setParameters(SysNode *nd, SysMachine *cpu)
                        { myNode = nd; myCPU = cpu; }

   // to allow module initialization
   void simReset()                        { moduleInitialization(); }
   virtual void moduleInitialization()    { ; }

   // some useful shortcuts to SysNode
   int                myGlobalId()        { return myNode->globalId;          }
   int                nNeighbors()        { return myNode->nNeighbors;        }
   SysInfoModule*     myNeighbor(int i)   { assert(i <= myNode->nNeighbors);
                                            return myNode->myNeighbors[i];    }
};

#endif
```

Classe SysMigrationModule

```
#if !defined(SYSMIG_H)
#define SYSMIG_H

class SysMigrationModule : public SysModule {
   int   *vecMigrations;         // # of migrations to [1..nNeighbors]
   int   *vecBadMigrations;      // target node overloaded
public:
   ~SysMigrationModule();
   SysMigrationModule(char *n = "SYSMIGMODULE"):
      SysModule(n), vecMigrations(NULL), vecBadMigrations(NULL)   { ; }

   // virtual function to deal with the actual migration
   virtual void     migrateTheLoad(int toNode, SysWorkLoad *load);
   virtual void     migrateNLoads(int toNode, int n);
   virtual SimTime  migrationCost(SysWorkLoad *) = 0;

   // other aux virtual functions (needed for statistics)
   virtual void    moduleInitialization();
   virtual void    simPrintStatistics(FILE *);
   virtual void    accountStatistics(int toNode, SysInfoModule *toInfoModule);

};

#endif
```

Classe SysInfoModule e derivadas

```
#if !defined(SYSINFO_H)
#define SYSINFO_H

class SysInfoModule  : public SysModule {
public:
   SysInfoModule(char *n = "SYSINFOMODULE"): SysModule(n)  { ; }

   // external activation points (called from SysSource)
   virtual void     loadArrival(SysWorkLoad *);
   virtual void     loadDepart()          { ; }

   // some shorcuts
   void   passToCpu(SysWorkLoad *load);
   void   migrateTheLoad(int toNode, SysWorkLoad *load);
   void   migrateNLoads(int toNode, int n);

   // messages haven't been implemented yet...
   virtual Boolean  acceptMessages()      { return FALSE;     }

   // must be redefined by derived classes
   virtual Boolean  amIOverloaded()       { return UNKNOWN;   }
};

class SysMsgInfoModule  : public SysInfoModule {
   // no data memsers
public:
   SysMsgInfoModule(char *n = "SYSMESSAGEINFO"): SysInfoModule(n) { ; }

   // functions to deal with messages
   void   sendMessage(int toNode, SysMessage *msg);

   // to be implemented
   virtual SimTime messageDelay(SysMessage *msg) = 0;
   virtual void    receiveMessage(int fromNode, SysMessage *msg) = 0;

   // a necessary redefinition
   Boolean  acceptMessages()              { return TRUE; }
};

class SysTimerInfoModule : public SysMsgInfoModule {
   SimTime              dTime;      // interval between activations
public:
   SysTimerInfoModule(SimTime t): SysMsgInfoModule("SYSTIMERINFO"), dTime(t) { ; }
   SysTimerInfoModule(char *n, SimTime t): SysMsgInfoModule(n), dTime(t)     { ; }

   // to 'registrate' the timer in the system
   void simStart();

   // function called by timer
   virtual void  timeExpired() =0;
};
#endif
```

Classe SysMachine

```
#if !defined(SYSMACHINE_H)
#define SYSMACHINE_H

class SysMachine : public SimUnit {
protected:
   SysNode        *myNode;        // attached to...
   SysLoadQ        loadq;         // queue of loads

public:
   SysMachine(char *n = "SYSMACHINE"): SimUnit(n), myNode(NULL)  { ; }

   void setParameters(SysNode *nd)                { myNode = nd;             }
   int  myGlobalId()                              { return myNode->globalId; }

   // handling functions
   virtual void           loadArrival(SysWorkLoad *)   =0;
   virtual void           loadDepart(SysWorkLoad  *);
   virtual SysWorkLoad*   extractTheLoad(SysWorkLoad *)=0;
   virtual SysWorkLoad*   extractLastLoad(void)        =0;

   // the 'load index' notion
   virtual SysLoadIndex  getLoadIndex()    =0;

   // number of loads in cpu
   virtual int  nLoadsInCpu() =0;

   // to iterate over the 'load list'
   SysWorkLoad*  startIteration()      { return loadq.startIteration(); }
   SysWorkLoad*  getNextLoad()         { return loadq.getNextLoad();    }
   Boolean       canIterateYet()       { return loadq.canIterateYet();  }
};


#endif
```

Bibliografia

[AND 91] ANDREWS, G. R. **Concurrent Programming Principles and Practice.** Redwood City, California: The Benjamin Cummings Publishing, 1991. 637 p.

[AND 93] ANDREWS, G. R.; OLSSON, R. A. **The SR Programming Language Concurrency in Practice.** Redwood City, California: The Benjamin Cummings Publishing, 1993. 345 p.

[BAL 89] BAL, H. E.; STEINER, J. G.; TANEMBAUM, A. S. Programming Languages for Distributed Computing. **ACM Computing Surveys**, New York, v. 21, n. 3, p. 261-321, Sept. 1989.

[BAR 85] BARAK, A.; SHILOH, A. A. Distributed Load Balancing Policy for a Multicomputer. **Software-Practice and Experience**, London, v. 15, n. 9, p. 901-913, Sept. 1985.

[BAT 94] HARCHOL-BALTER, M. **Process Lifetimes are Not Exponential, more like 1/T: Implications on Dynamic Load Balancing.** Berkeley: University of California, 1994. 24 p. (Technical Report n. UCB/CSD 94/826).

[BAT 95] HARCHOL-BALTER, M.; DOWNEY, A. B. **Exploiting Process Lifetime for Dynamic Load Balancing.** Berkeley: University of California, 1995. 18 p. (Technical Report n. UCB/CSD 95/887).

[BIS 91] BISHAK, D. P. ; ROBERTS, S. D. Object Oriented Simulation. In: WINTER SIMULATION CONFERENCE, 1991. **Proceedings...** San Diego: SCS, 1991. p. 194-203.

[BOK 79] BOKHARI, S. H. Dual Processor Scheduling with Dynamic Reassignment. **IEEE Transactions on Software Engineering**, New York, v. SE-5, p. 326-334, July 1979.

[BRI 92] BRICKER, A.; LITZKOW, M.; LIVNY, M. **Condor Technical Summary.** Madison: University of Wisconsin, 1992. 10 p. Disponível por ftp em ftp.cs.wisc.edu.

[CAS 88] CASAVANT, T. L.; KUHL, J. G. Effects of Response and Stability on Scheduling in Distributed Computing Systems. **IEEE Transactions on Software Engineering**, New York, v. 14, n. 11, p. 1578-1588, Nov. 1988.

[CAS 88b] CASAVANT, T. L.; KUHL, J. G. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. **IEEE Transactions on Software Engineering**, New York, v. 14, n. 2, p. 141-154, Feb. 1988.

[CHO 82] CHOU, T. C. K.; ABRAHAM, J. A. Loading Balancing in Distributed Systems. **IEEE Transactions on Software Engineering**, New York, v. SE-8, n. 4, p. 401-412, July 1982.

[CHW 79] CHOW, Y. C. ; KOHLER W. H. Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System. **IEEE Transactions on Computers,** New York, v. 28, p. 354-361, May 1979.

[COP 96] COPSTEIN, B.; PEREIRA, C. E.; WAGNER, F. R. The Object-Oriented Approach and The Event Discrete Simulation Paradigms. In: EUROPEAN SIMULATION MULTICONFERENCE, 1996, Budapest, Hungary. **Proceedings...** San Diego: SCS, 1996. p. 57-61.

[COT 92] COTA, B.; SARGENT, R. G. A Modification of the Process Interaction World View. **ACM Transactions on Modeling and Computer Simulation**, New York, v. 2, n. 2, p. 109-129, Apr. 1992.

[COU 94] COULORIS, G.; DOLLIMORE, J.; KINDBERG, T. **Distributed Systems - Concepts and Design.** 2. ed. [S. l.]: Addison-Wesley, 1994. 644 p.

[DEV 89] DEVARAKONDA, M.; IYER, R. K. Predictability of process resource usage: A measurement-based study of UNIX. **IEEE Transactions on Software Engineering**, New York, v. 15, n. 12, p. 1579-1586, Dec. 1989.

[DIK 89] DIKSHIT, P.; TRIPATHI, S. K.; JALOTE, P. SAHAYOG: A Test Bed for Evaluating Dynamic Load Sharing Policies. **Software-Practice and Experience**, London, v. 19, n. 5, p. 411-435, May 1989.

[DOW 95] DOWNEY, A. B. ; HARCHOL-BALTER M. **A note on 'The Limited Performance of Migrating Active Process for Load Sharing'.** Berkeley: University of California, 1995. 7 p. (Technical Report n. UCB/CSD 95/888).

[DOY 90] DOYLE, R. J. Object Oriented Simulation Programming In: SCS MULTICONFERENCE ON OBJECT ORIENTED SIMULATION, 1990, San Diego, California. **Proceedings...** San Diego: SCS, 1990. p. 1-6.

[EAG 86] EAGER, D. L.; LAZOWSKA, E. D.; ZAHORJAN, J. Adaptive Load Sharing in Homogeneous Distributed Systems. **IEEE Transactions on**

Software Engineering, New York, v. SE-12, n. 5, p. 662-675, May 1986.

[EFE 82] EFE, K. Heuristic Models of Task Assignment Scheduling in Distributed Systems. **IEEE Computer**, New York, v. 15, p. 50-56, June 1982.

[ELL 93] ELLIS, M. A. ; STROUSTRUP, B. **C++ Manual de Referência Anotado.** Rio de Janeiro, R. J.: Campus, 1993. 546 p.

[FER 78] FERRARI, D. **Computer System Performance Evaluation.** Englewood Cliffs, N. J.: Prentice Hall,1978. 554 p.

[FER 83] FERRARI, D.; SERAZZI, G.; ZEIGNER, A. **Measurement and Tuning of Computer Systems.** Englewood Cliffs, N. J.: Prentice Hall, 1983.

[FER 85] FERRARI, D. **A Study of Load Indices for Load Balancing Schemes.** Berkeley: University of California, 1985. 9 p. (Technical Report n. UCB/CSD 86/262).

[FER 87] FERRARI, D. ; ZHOU, S. **An Empirical Investigation of Load Indices for Load Balancing Applications.** Berkeley: University of California, 1987. 14 p. (Technical Report n. UCB/CSD 87/353).

[FRI 96] FRIEDMAN, R. ; MOSSE, D. **Load Balancing Schemes for High-Throughput Distributed Fault-Tolerant Servers.** [S. l.]: Cornell University, Computer Science Department, 1996. 16 p. (Technical Report n. TR96-1616).

[GOR 69] GORDON, G. **System Simulation.** Englewood Cliffs, N. J.: Prentice Hall, 1969.

[GOS 93] GOSWAMI, K. K. M.; DEVARAKONDA, M.; IYER, R. K. Prediction-Based Dynamic Load-Sharing Heuristics. **IEEE Transactions on Parallel and Distributed Systems**, New York, v. 4, n. 6, p. 638-648, June 1993.

[HAC 89] HAC, A. Loading Balancing in Distributed: A Summary. **Performance Evaluation Review**, [S. l.], v. 16, p. 17-25, Feb. 1989.

[HEI 92] HEIβ, H. U.; SCHMITZ, M. **Distributed Load Balancing Using a Physical Analogy.** Karlsruhe: University of Karlsruhe, Department of Computer Science, 1992. (Internal Report n. 5/92).

[HEI 92b] HEIβ, H. U. ; PAYER, A. PASTE: A Tool for Evaluation of Processor Allocation Strategies. In: INTERNATIONAL CONFERENCE ON

MODELING TECHNIQUES AND TOOLS FOR COMPUTER PERFORMANCE EVALUATION, 1992, Edinburgh, Scotland. **Proceedings...** Edinburgh: Edinburgh University Press, 1992. p. 16-18.

[HER 90] HERRING, C. ModSim: A New Object-Oriented Simulation Language. In: SCS MULTICONFERENCE ON OBJECT ORIENTED SIMULATION, 1990, San Diego, California. **Proceedings...** San Diego: SCS, 1990. p. 55-59.

[HOO 86] HOOPER, J. W. Strategy-related characteristics of discrete-event languages and models. **Simulation**, San Diego, v. 46, n. 4, p. 153-159, Apr. 1986.

[KAR 91] KARIAN, Z. A.; DUDEWICS, E. J. **Modern Statistical Systems, and GPSS Simulation — The first course.** New York: W. H. Freeman and Company, 1991.

[KIN 90] KING, P. J. B. **Computer and Communication Systems Performance Modeling.** Englewood Cliffs, N. J.: Prentice Hall, 1990.

[KUN 91] KUNZ, T. The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme. **IEEE Transactions on Software Engineering**, New York, v. 17, n. 7, p. 725-730, July 1991.

[LEM 93] WILLEBEEK-LEMAIR, M. H.; REEVES, A. P. Strategies for Dynamic Load Balancing on Highly Parallel Computers. **IEEE Transactions on Parallel and Distributed Systems**, New York, v. 4, n. 9, p. 979-993, Sept. 1993.

[LOH 96] LOH, P. K. K et al. How Network Topology Affects Dynamic Load Balancing. **IEEE Parallel & Distributed Technology**, New York, v. 4. n. 3, p. 25-35, Fall 1996.

[MAL 93] MALLOY, B.; HARROLD, M. J.; MCGREGOR, J. D. The Implementation of a Simulation Language using Dynamic Binding. In: SCS WESTERN SIMULATION MULTICONFERENCE, 1993, La Jolla, California. **Proceedings...** San Diego: SCS, 1993. p. 3-8.

[MAK 91] MAK, V. W. DOSE: A Modular and Reusable Object-Oriented Simulation Environment. In: SCS MULTICONFERENCE ON OBJECT ORIENTED SIMULATION, 1991, Anaheim, California. **Proceedings...** San Diego: SCS, 1991. p. 3-11.

[MEY 88] MEYER, B. **Object-Oriented Software Construction.** Englewood Cliffs, N. J.: Prentice Hall,1985. 534 p.

[MIL 94] MILOJICIC, D. S. **Load Distribution - Implementation for the Mach Microkernel.** Wiesbaden: Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, 1994. 149 p.

[MUN 95] MUNIZ, F. J. ; ZALUSKA, E. J. Parallel load-balancing: An extension to the gradient model. **Parallel Computing**, Netherlands, v. 21, p. 287-301, 1995.

[NAI 85] MACNAIR, E. A.; SAUER, C. H. **Elements of Practical Performance Modeling.** Englewood Cliffs, N. J.: Prentice Hall,1985.

[NAN 81] NANCE, R. E. The Time and State Relationship in Simulation Modeling. **ACM Communications**, New York, v. 24., n. 4, p. 173-179, Apr. 1981.

[NI 85] NI, L. M.; XU, C. W.; GENDREAU, T. B. A Distributed Drafting Algorithm for Load Balancing. **IEEE Transactions on Software Engineering**, New York, v. SE-11, n. 10, p. 1153-1161, Oct. 1985.

[NIC 87] NICHOLS, D. Using idle workstations in a shared computing environment. In: ACM SYMPOSIUM ON OPERATING SYSTEM PRINCIPLES, 11., 1987, Austin, TX. **Proceedings...** New York: ACM, 1987. p. 5-12.

[NOG 95] NOGUEIRA, MAURO L. B. **Migração de Processos** - Um Estudo de Casos: Trabalho Individual. Porto Alegre: CPGCC/UFRGS, 1995 (TI-526).

[PEA 91] PEASE, D; GHAFOOR, A. ; AHMAD, I. PAWS: A Performance Evaluation Tools for Parallel Computing Systems. **IEEE Computer**, New York, p. 18-29, Jan. 1991.

[QUI 87] QUINN, M. J. **Designing Efficient Algorithms for Parallel Computers.** New York: McGraw-Hill, 1987. 288p.

[RUS 90] RUSSEL, E. C. **Course Notes - Simulation and Modeling.** [S.l.]: Russel Software Technology, 1990.

[SAU 81] SAUER, C. H.; CHANDY, K. M. **Computer System Performance Modeling**. Englewood Cliffs, N. J.: Prentice Hall, 1981.

[SHA 75] SHANNON, R. E. **Systems Simulation — the art and science.** Englewood Cliffs, N. J.: Prentice-Hall, 1975.

[SNY 84] SNYDER, L. Parallel Programming and the Poker Programming Environment. **IEEE Computer**, New York, p. 27-36, July 1984.

[SOA 90] SOARES, L. F. G. **Modelagem e Simulação Discreta de Sistemas.** São Paulo: IME-USP, 1990.

[SON 94] SONG., J. A Partially Asynchronous and Iterative Algorithm for Distributed Load Balancing. **Parallel Computing**, Netherlands, v. 20, n. 6, p. 853-868, June 1994.

[STA 85] STANKOVIC, J. A. Stability and Distributed Scheduling Algorithms. **IEEE Transactions on Software Engineering**, New York, v. SE-11, n. 10, p. 1141-1152, Oct. 1985.

[STR 91] STROUSTRUP, B. **The C++ Programming Language**. 2. ed. [S. l.]: Addison-Wesley, 1991. 691 p.

[TAN 89] TANENBAUM, A. S. **Computing Networks.** 2. ed. Englewood Cliffs, N. J.: Prentice Hall, 1989. 658 p.

[TAN 95] TANENBAUM, A. S. **Distributed Operating Systems.** Upper Saddle River, N. J.: Prentice Hall, 1995. 614 p.

[THE 85] THEIMER, M.; LANTZ, D.; CHERITON, D. Preemptable remote execution facilities for the V-system. In: SYMPOSIUM ON OPERATING SYSTEM PRINCIPLES, 10., 1985, Austin, TX. **Proceedings...** [S.l.:s.n.], 1985. p. 2-12.

[VAU 91] VAUGHAN, P. W.; NEWTON, D. E.; JOHNS, R. P. PRISM: An Object-Oriented System Modeling Environment in C++. In: SCS MULTICONFERENCE ON OBJECT ORIENTED SIMULATION, 1991, Anaheim, California. **Proceedings...** San Diego: SCS, 1991. p. 32-39.

[WAN 85] WANG, Y. T.; MORRIS, R. J. T. Load Sharing in Distributed Systems. **IEEE Transactions on Computers**, New York, v. c-34, n. 3, p. 204-217, Mar. 1985.

[ZHE 93] ZHEN, Q.; CHOW P. EXsim: A General Purpose Object-Oriented Environment for Discrete Event Simulations. In: SCS WESTERN SIMULATION MULTICONFERENCE, 1993, La Jolla, California. **Proceedings...** San Diego: SCS, 1993. p. 15-21.

[ZHO 87] ZHOU, S.; FERRARI, D. **An Experimental Study of Load Balance Performance.** Berkeley: University of California, 1987. 26 p. (Technical Report UCB/CSD 87/336).

[ZHO 88] ZHOU, S. A Trace-Driven Simulation Study of Dynamic Load Balancing. **IEEE Transactions on Software Engineering**, New York, v. 14, n. 9, p. 1327-1341, Sept. 1988.

Informática  UFRGS

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

"ROBIN HOOD - Um Ambiente para a Avaliação de políticas de Balanceamento de Carga "

por

Mauro Lúcio Baioneta Nogueira

Dissertação apresentada aos Senhores:

Prof. Dr. Celso Maciel da Costa (PUC-RS)

Prof. Dr. Rômulo Silva Oliveira

Profa. Dra. Ingrid Eleonora Schreiber Jansch Pôrto

Vista e permitida a impressão.
Porto Alegre, 12/04/2000

Prof. Dr. Cláudio Fernando Resin Geyer,
Orientador.

Profa. [illegible] Freitas
Coordenadora [illegible] Pós Graduação
em Ciência da Computação - PPGC
Instituto de Informática - UFRGS