ESCALONAMENTO DINÂMICO DISTRIBUÍDO EM MULTICOMPUTADORES DE ALTO DESEMPENHO

GUSTAVO PEIXOTO DE AZEVEDO

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇAO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Valmir Carneiro Barbosa, Ph.D.
(Presidente)

Prof. Claudio Luis de Amorim, Ph.D.

Prof. Júlio Salek Aude, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
MAIO DE 1992

AZEVEDO, GUSTAVO PEIXOTO DE

Escalonamento Dinâmico Distribuído em Multicomputadores de Alto Desempenho

[Rio de Janeiro], 1992

ix, 99pp., 29,7cm. (COPPE/UFRJ, M.Sc.,

Engenharia de Sistemas e Computação, 1992)

Tese - Universidade Federal do Rio de Janeiro

1. Escalonamento Distribuído 2. Comunicação entre Tarefas Migrantes

3. Multicomputadores 4. Simulação de Sistemas Distribuídos

I. COPPE/UFRJ II. Título (série).

"Se você já conseguiu tudo o que planejou, você não planejou o suficiente."
— Martial

Dedicatória

Dedico este trabalho aos meus pais Vampré e Esther e aos meus irmãos Rafael e Leonardo, que sempre me apoiaram e incentivaram em todos os momentos importantes da minha vida.

Dedico este trabalho também a minha namorada Narcisa pela sua compreensão, incentivo e paciência.

Agradecimentos

Agradeço ao Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro, pelos recursos materiais, sem os quais seria impossível para mim realizar este trabalho.

Ao meu orientador pelas oportunidades conferidas.

Aos membros da banca, que nos honraram com sua participação.

Aos colegas Rafael Azevedo e Carlo Emanoel pelas inúmeras sugestões.

Aos colegas do Grupo de Computação Gráfica do NCE, em especial ao Serpa, pelo desenvolvimento e adaptação do VISUALIDATA.

Aos colegas Rafael Azevedo e Luis Fernando Pereira pelo auxílio e sugestões para a melhor utilização do latex.

A Suzan, pela adaptação do latex para o formato tese.

E finalmente, a todos aqueles que, direta ou indiretamente, contribuíram (ou pelo menos não atrapalharam) a realização deste trabalho.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

Escalonamento Dinâmico Distribuído em Multicomputadores de Alto Desempenho

Gustavo Peixoto de Azevedo
maio de 1992

Orientador: Valmir Carneiro Barbosa
Programa: Engenharia de Sistemas e Computação

O objetivo principal de um sistema Multicomputador de Alto Desempenho (MAD) é tornar viável a resolução de problemas que os computadores baseados em arquiteturas tradicionais não têm a capacidade computacional necessária para resolver. Estes problemas podem ser resolvidos em um MAD, desde que seus métodos de solução sejam programados na forma de Aplicações Paralelas Distribuídas (APD), e que o software de controle do MAD seja capaz de distribuir eficientemente as tarefas constituintes da APD pelos computadores do sistema.

Este trabalho estuda o escalonamento dinâmico distribuído das tarefas de uma APD em um MAD, considerando as características fundamentais destas aplicações: comunicação entre tarefas e paralelismo de grão médio.

Algoritmos tradicionais de escalonamento dinâmico distribuído são comparados com versões respectivas adaptadas para considerarem as características acima.

Os algoritmos são avaliados por um simulador híbrido baseado em um modelo de execução de APD, que emula todo o software de controle do MAD, incluindo um novo algoritmo para a realização da comunicação entre tarefas migrantes.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

Distributed Dynamic Scheduling for High Performance Multicomputers

Gustavo Peixoto de Azevedo
May, 1992

Thesis Supervisor: Valmir Carneiro Barbosa
Department : Systems Engineering and Computing

The main objective for using a High Performance Multicomputer system (MAD) is to gain the ability to solve a class of problems, which computers based on traditional architectures do not have the necessary computational capacity for. These problems can be solved by a MAD, if their solution methods are programmed as Parallel Distributed Applications (APD) and if the control software in the MAD is capable of efficiently distributing the application tasks by the computers in the system.

This thesis studies the dynamic and distributed scheduling of a APD tasks in a MAD, considering two fundamental aspects of these applications: tasks intercommunication and medium grain parallelism. Traditional algorithms for dynamic and distributed scheduling are compared against their adapted versions that take these aspects into consideration.

The algorithms are evaluated by a hybrid simulator based on an execution model for the APD, that emulates all the control software in the MAD, including a new algorithm for the intercommunication of migrating tasks.

Índice

Capítulo I

Introdução

A contínua evolução na tecnologia de implementação dos computadores nas últimas décadas tem possibilitado enormes avanços a estes equipamentos, tanto em termos de capacidade de processamento como de custo de produção. Estes avanços têm resultado em um grande desenvolvimento da indústria de informática, com influência direta em todas as atividades humanas.

Apesar de todos estes avanços, a sociedade continuará a demandar computadores com desempenho cada vez maior, em razão da necessidade de avaliar modelos gradativamente mais realísticos de fenômenos naturais. Muitos problemas importantes em áreas como, por exemplo, previsão do tempo, exploração de petróleo e física nuclear, requerem um poder computacional muito além da capacidade dos computadores disponíveis atualmente [58].

A aproximação de limites físicos para a evolução da tecnologia de eletrônica e de microeletrônica [44] torna necessário buscar desempenhos superiores para os computadores através do desenvolvimento de novas arquiteturas e/ou de novas tecnologias de implementação.

Este trabalho insere-se no contexto do desenvolvimento de uma arquitetura para um sistema de computação distribuída paralela com o potencial de obter um alto desempenho a um custo inferior ao dos computadores de desempenho menor ou similar, baseados nas arquiteturas tradicionais.

As seções a seguir apresentam o escopo e um resumo deste trabalho e a forma como este texto está organizado.

I.1 Apresentação e Resumo da Tese

Um sistema **Multicomputador de Alto Desempenho** (MAD) é definido no contexto deste trabalho como sendo uma rede escalável de computadores baseada em conexões ponto a ponto. A escalabilidade de um sistema MAD significa que o número de seus componentes pode ser incrementado eficientemente, sem restrições quanto ao tamanho máximo do sistema.

O objetivo principal de um sistema MAD é tornar viável a resolução de problemas que demandam tanto processamento, que sua resolução em computadores tradicionais é inviável. O cumprimento deste objetivo depende da existência de algoritmos paralelos para a resolução destes problemas e do aproveitamento eficiente deste paralelismo. Exemplos de problemas para resolução por sistemas MAD são os problemas de inteligência artificial, de cálculo numérico, de otimização e de simulação [4].

Uma aplicação para um sistema MAD, que denominamos uma **aplicação paralela distribuída** (APD), é um conjunto de unidades de execução seqüencial, denominadas **tarefas**, que executam paralelamente e se comunicam através da troca de **mensagens**, colaborando para a resolução de um mesmo problema.

O mapeamento, ao longo do tempo, de tarefas aos computadores de um sistema MAD é denominado **escalonamento**. O escalonamento determina, conseqüentemente, a forma de utilização dos dois recursos fundamentais de um sistema MAD: capacidade de processamento e capacidade de comunicação. A otimização da utilização da capacidade de comunicação é conflitante com a otimização da utilização da capacidade de processamento. O escalonamento de tarefas em sistemas com vários processadores pertence a classe dos problemas NP-Completo [30]. Deste modo, não há métodos eficientes que garantam um escalonamento ótimo.

Quando o comportamento das tarefas que compõem a APD é determinístico e conhecido a priori, as decisões de escalonamento independem da evolução real da execução delas e, portanto, o escalonamento pode ser **estático**. Em caso contrário, para se atingir um melhor desempenho, as decisões de escalonamento têm que ser tomadas durante a execução, pois dependem da sua evolução, exigindo um escalonador **dinâmico**, que se adapte ao estado corrente de execução. Esta adaptabilidade dos escalonadores dinâmicos permite a sua utilização no escalonamento de qualquer APD, tornando-os assim mais genéricos.

A escalabilidade de um sistema MAD depende da sua arquitetura e de seu software de controle. O escalonador de um sistema MAD faz parte do software de controle. Para que o escalonador seja escalável, a responsabilidade pelas ações de

escalonamento deve estar distribuída pelos computadores do sistema MAD, ou seja, o escalonador deve implementar um algoritmo distribuído.

Uma aplicação paralela, quando programada para sistemas distribuídos com centenas de processadores, apresenta características fundamentais, que a princípio têm conseqüências para o escalonamento, pois influenciam a utilização dos recursos computacionais. São as seguintes características:

- **Comunicação entre as suas tarefas.** A demanda pelos recursos computacionais do sistema decorre não só da realização do processamento das tarefas, como também da comunicação entre as mesmas. Assim, o mapeamento, ao longo do tempo, das tarefas aos elementos processadores do sistema determina não só a utilização da capacidade de processamento do sistema, mas também a utilização da capacidade de comunicação do sistema.
- **Paralelismo de grão médio.** Um paralelismo de grão médio implica que o número típico de instruções executadas por uma tarefa não é significativamente muito maior que o necessário à realização de uma decisão de escalonamento. Assim os custos de escalonamento podem ser significativos comparados à utilização dos recursos computacionais realizada diretamente pelas aplicações.

Este trabalho estuda o escalonamento dinâmico distribuído das tarefas de uma APD, em um sistema MAD. A contribuição primordial deste trabalho é verificar a relevância, em relação ao desempenho, da consideração destas características pelos escalonadores dinâmicos distribuídos.

O método de avaliação adotado neste trabalho é a **simulação**. O comportamento de uma APD e um ambiente operacional para a sua execução são simulados. O comportamento de uma APD é definido através de um modelo estocástico de criação, execução e intercomunicação de suas tarefas componentes. O ambiente operacional é dado por um emulador de sistemas MAD e pela emulação dos mecanismos de suporte a execução das tarefas. Entre esses mecanismos se destacam o algoritmo de escalonamento em avaliação e o mecanismo que realiza a comunicação entre tarefas. Esta forma de avaliação possibilita a aquisição de estimativas mais precisas dos resultados proporcionados por escalonadores em um sistema MAD real.

O desenvolvimento deste trabalho exigiu a concepção dos seguintes ítens, que correspondem a mais algumas contribuições:

- **Um modelo de execução das aplicações paralelas distribuídas.** Segundo Domenico Ferrari [27], não há um modelo de carga computacional para as aplicações dos sistemas paralelos frouxamente conectados (como os sistemas MAD) que considere adequadamente o paralelismo e a comunicação entre tarefas, que são intrínsecos a estas aplicações.

- **Um algoritmo para a comunicação entre tarefas migrantes.** Não há um algoritmo para a gerência da comunicação entre tarefas migrantes que executam em um sistema distribuído com ligações ponto a ponto (como os sistemas MAD), que seja totalmente satisfatório.
- **Um simulador adequado à avaliação de escalonadores dinâmicos distribuídos em sistemas MAD.** O DDSS permite a avaliação dos algoritmos de escalonamento em sistemas distribuídos com ligações ponto a ponto, considerando-se a comunicação entre tarefas e o paralelismo de grão médio das aplicações.

I.2 Organização do Texto

Esta seção explica como este texto está organizado. São descritos os temas abordados em cada capítulo e como eles são apresentados.

O capítulo II apresenta as características fundamentais dos sistemas distribuídos, a arquitetura dos sistemas MAD, as características das suas aplicações, e algumas arquiteturas alternativas propostas para o estudo do escalonamento distribuído.

O capítulo III apresenta o estágio atual na área de escalonamento para sistemas distribuídos, visando a sua aplicação em sistemas MAD. Inicialmente são apresentados os princípios fundamentais do escalonamento de sistemas de computadores. É realizada uma revisão da teoria de escalonamento distribuído, propondo uma classificação e analisando questões importantes, algumas polêmicas, quanto ao escalonamento distribuído. A última seção faz um retrospecto dos trabalhos anteriores, com ênfase nos resultados obtidos e nos algoritmos apresentados.

O capítulo IV apresenta a metodologia para a avaliação de escalonadores dinâmicos distribuídos empregada neste trabalho. Inicialmente o objetivo da avaliação é formalizado. A seguir os trabalhos anteriores relacionados são apresentados. Na seção IV.3 os dois métodos básicos de avaliação são apresentados suscintamente e um dos métodos é escolhido em função de justificativas apresentadas. A seguir (seção IV.4) são discutidas as formas de modelamento do sistema. O capítulo termina com a apresentação de um modelo de execução das APD e com um modelo de funcionamento dos sistemas MAD.

O capítulo V aborda o sistema de comunicações desenvolvido para os sistemas MAD. Inicialmente (seções V.1 e V.2) são revistas as propriedades fundamentais requeridas e as medidas de desempenho dos sistemas de comunicação para multicomputadores. A seção V.3 apresenta a estrutura geral dos sistemas de comunicação

para multicomputadores, a partir de uma análise dos principais problemas relacionados ao seu projeto, com ênfase para o problema da realização da comunicação entre tarefas migrantes. A subseção V.3.5 analisa as soluções anteriores para o problema da realização da comunicação entre tarefas migrantes. A seguir a seção V.4, apresenta as soluções adotadas neste trabalho, com ênfase especial para o novo protocolo proposto para a comunicação entre tarefas migrantes. Nas subseções seguintes são apresentadas a definição deste protocolo (V.4.1), a verificação da sua correção (V.4.2) e a verificação da sua qualidade com relação a garantia das propriedades definidas na seção V.1.

O capítulo VI aborda o processo de simulação adotado, os componentes do simulador, a sua organização e as medidas de desempenho coletadas.

O capítulo VII descreve as simulações realizadas. São especificados os algoritmos escolhidos para avaliação, os parâmetros das cargas computacionais empregadas e os sistemas MAD selecionados. Por último são apresentados os resultados encontrados, uma avaliação do processo de simulação realizado e uma interpretação destes resultados.

Finalmente o capítulo VIII apresenta conclusões gerais do trabalho, avalia o trabalho em si e aponta direções para a continuação da pesquisa

Capítulo II

Sistemas MAD: Multicomputadores de Alto Desempenho

A definição dos sistemas MAD tem por objetivo estabelecer uma classe de sistemas distribuídos atendendo aos seguintes requisitos:

- Ser adequada ao estudo do escalonamento distribuído dinâmico em sistemas de alto desempenho.
- Representar uma relação de compromisso entre a flexibilidade da arquitetura, a expansibilidade e a escalabilidade visando a construção de sistemas de alto desempenho.

Este capítulo apresenta as características fundamentais dos sistemas distribuídos, a arquitetura dos sistemas MAD, as características das suas aplicações, e algumas arquiteturas alternativas propostas para o estudo do escalonamento distribuído.

II.1 Sistemas Distribuídos

Um sistema distribuído é uma coleção de computadores autônomos ligados através de uma rede de interconexão, que não possibilita o compartilhamento de memória entre os seus componentes, limitando a interação entre eles à realizada através de trocas de mensagens.

A organização de um conjunto de computadores sob a forma de um sistema distribuído tem por finalidade possibilitar a cooperação dos seus componentes para a realização de objetivos comuns. A integração dos componentes de um sistema

distribuído, aumenta a disponibilidade de recursos para uma mesma aplicação e torna viável a exploração de paralelismo pelas aplicações.

Os sistemas distribuídos podem ser classificados de acordo com a forma de conexão entre os seus componentes. Nos sistemas distribuídos ponto a ponto os computadores estão conectados dois a dois por canais de comunicação exclusivos, enquanto nos sistemas distribuídos por difusão cada componente está conectado a todos os demais por um ou mais canais de comunicação compartilhados.

Nos sistemas distribuídos por difusão todos os seus computadores podem se comunicar diretamente com todos os demais. Porém, como a comunicação ocorre através de canais de comunicação compartilhados, há o problema de colisão quando mais de um computador tenta utilizar simultaneamente um mesmo canal de comunicação.

Já nos sistemas distribuídos ponto a ponto não há problemas de colisão, pois toda comunicação ocorre através de canais de comunicação exclusivos. Em compensação, como seus componentes não estão necessariamente completamente conectados, a comunicação entre dois de seus computadores pode percorrer rotas que envolvam alguns dos outros componentes do sistema.

Os canais de comunicação de um sistema distribuído apresentam limitações físicas na sua capacidade de comunicação decorrentes da existência de uma latência nos seus canais de comunicação e de uma taxa máxima para a transferência de dados através deles. Estas limitações físicas resultam nas seguintes limitações lógicas:

- Sempre há um atraso na comunicação entre dois computadores de um sistema distribuído. Assim, não se pode garantir a atualidade das informações de uma mensagem no momento da sua recepção. Portanto, os sistemas distribuídos são incapazes para manter consistentes as informações globais de seu estado em pontos distribuídos de controle.
- Quanto maior o volume de informações sobre o estado global de um sistema distribuído, maior o tempo necessário a sua transmissão e conseqüentemente menor a acurácia destas informações.
- Considerando que um computador de um sistema distribuído só é capaz de conhecer os eventos relacionados a si próprio e aqueles relativos a transmissão e recepção de mensagens; e que um evento relacionado à chegada de uma mensagem necessariamente ocorre após a emissão dela, e a fortiori após todos os eventos anteriores a ela; então, os eventos que não antecedem as mensagens já recebidas por um computador, mas que já ocorreram nos demais computadores, não podem ser conhecidos por ele. Deste modo, é impossível colocar

em ordem temporal qualquer par arbitrário de eventos e conseqüentemente a percepção desta ordem por um observador é arbitrária. Assim, em um instante qualquer, não é possível a um computador componente de um sistema distribuído distinguir todos os eventos que já ocorreram, e portanto ele desconhece o estado global do sistema.

Os sistemas distribuídos ideais apresentam as seguintes qualidades:

- *Escalabilidade*: Capacidade de crescimento de um sistema. A escalabilidade de um sistema está associada ao limite superior para o crescimento do sistema; quanto mais escalável, maior será o tamanho máximo potencial do sistema. O crescimento de um sistema altamente escalável implica em um aumento correspondente mínimo na sua complexidade, seja da arquitetura do elemento processador, seja da rede de interconexão e de seus componentes, ou seja do software de controle [43].
- *Expansibilidade*: Capacidade de um sistema para crescimento incremental. A expansibilidade de um sistema está associada à continuidade no crescimento do sistema.

Em síntese, sistemas distribuídos são caracterizados pela interação de seus computadores unicamente através de um sistema de comunicação baseado na troca de mensagens. Esta propriedade, se por um lado dificulta o projeto de um sistema distribuído em razão da impossibilidade do conhecimento do estado global por um dos seus computadores a qualquer instante, por outro lado possibilita a construção de sistemas expansíveis e escaláveis.

II.2 Arquitetura dos Sistemas MAD

Daniel Reed [58] define o termo **multicomputador** para designar uma classe de sistemas distribuídos com um grande número de **nós computacionais** interconectados em uma topologia regular determinada por **ligações ponto a ponto**, que cooperam assincronamente por troca de mensagens para executar as tarefas de um programa paralelo. Cada nó computacional, fabricado com um pequeno número de circuitos VLSI, contém um **processador computacional**, um **módulo de memória local** e um **processador de comunicações** capaz de rotear as mensagens sem atrasar o processador computacional.

Os sistemas MAD representam uma subclasse dos multicomputadores definidos por Daniel Reed [58], tendo como principal característica a grande escalabilidade,

cujo objetivo é possibilitar a construção de sistemas distribuídos com um alto desempenho.

Os processadores de comunicação gerenciam toda a comunicação em um MAD. Eles são responsáveis pela interface com os canais de comunicação e possibilitam a troca de mensagens entre as tarefas e entre os escalonadores. Eles executam em paralelo com os processadores computacionais e são capazes de lidar plenamente com toda a capacidade de comunicação dos canais de comunicação que gerenciam.

Um MAD pode ser **homogêneo** ou **heterogêneo**, ou seja, os componentes do sistema podem ser idênticos ou não. A heterogeneidade neste caso refere-se apenas à diferença nas velocidades de operação entre os componentes similares do sistema, que são funcionalmente iguais.

II.3 Aplicação dos Sistemas MAD

As aplicações ideais para os sistemas MAD implementam algoritmos distribuídos assíncronos com um grau elevado de paralelismo de grão médio. Elas são então denominadas neste trabalho por **aplicações paralelas distribuídas** (APD).

Os algoritmos devem ser paralelos em grau elevado, porque um sistema MAD normalmente é composto por muitos processadores e está dedicado à execução de apenas uma aplicação a cada momento. Portanto, para que os muitos processadores de um sistema MAD mantenham-se em execução, é necessário que a aplicação seja costituída por ainda mais unidades de execução.

Os algoritmos devem ser distribuídos porque são constituídos por unidades de execução que cooperam para a resolução de um mesmo problema, interagindo através da troca de mensagens.

O grão de paralelismo das aplicações dos sistema MAD deve ser médio pois:

- Em um paralelismo de grão pequeno, a comunicação entre as tarefas é muito intensa e predomina em relação ao seu processamento; não é, portanto, adequado aos sistemas MAD, onde a comunicação se processa apenas por mensagens que percorrem canais de comunicação de capacidade limitada.
- Em um paralelismo de grão grande, o grau de paralelismo é limitado, não sendo portando adequado aos sistemas MAD que apresentam muitos processadores.
- Em um paralelismo de grão médio, por sua vez, o grau de paralelismo pode ser elevado, porém há um equilíbrio nas demandas por comunicação e processamento das tarefas, que podem ser eficientemente atendidas pelos sistemas

MAD.

Quanto à sua origem, o paralelismo de uma aplicação pode ser decorrente de uma divisão algorítmica, ou de uma divisão do domínio de atuação. Paralelismo algorítmico refere-se a algoritmos que são constituídos de um número de tarefas logicamente distintas, que cooperam para cumprir um único objetivo. Paralelismo por divisão de domínio refere-se a algoritmos que realizam as mesmas operações sobre conjuntos distintos de dados.

Os sistemas MAD podem executar qualquer APD, independentemente da origem do paralelismo da aplicação. As APD que podem executar sobre um domínio extensamente divisível como os problemas de *inteligência artificial*, *cálculo numérico* e *simulação* [4], podem ser executados em sistemas MAD. Também podem ser executadas em sistemas MAD as APD algorítmicas em que tarefas são criadas e dinamicamente alocadas aos processadores de acordo com os dados de entrada. São exemplos de aplicações as implementações dos algoritmos paralelos de *"divide-and-conquer"* e de *buscas em árvores de "jogos mini-max"*, que apresentam este comportamento dependente do tempo de execução [57].

O projeto de um algoritmo distribuído envolve a especificação da computação a ser realizada por cada tarefa, bem como da comunicação entre elas [1]. A qualidade de um algoritmo distribuído pode ser medida pela densidade de tráfego gerada sobre os canais de comunicação [56].

Uma importante propriedade de um algoritmo distribuído é o seu grau de particionamento. O particionamento está relacionado com o nível de simetria das suas tarefas. Os níveis de simetria podem ser [56]:

- **Assimetria**: cada tarefa executa um código diferente, é o caso, por exemplo, de um algoritmo cliente servidor.
- **Simetria textual**: todas as tarefas executam códigos idênticos, mas cada tarefa mantém a sua identidade própria.
- **Simetria forte**: todas as tarefas executam códigos idênticos, as tarefas não são identificadas, mas o comportamento delas depende das mensagens recebidas por cada uma, que podem ser diferentes.
- **Simetria total**: todas as tarefas executam códigos idênticos e se comportam do mesmo modo.

Para possibilitar um paralelismo intenso independentemente da sua origem, é necessário que a migração de tarefas seja eficiente. Para tanto, o grau mínimo de particionamento das aplicações corresponde à simetria textual. Assim, o código

da aplicação paralela é replicado em todos os computadores do sistema MAD e a migração de tarefas pode ser muito eficiente, pois não é necessária a migração do código em execução.

II.4 Sistemas Distribuídos Alternativos

Esta seção apresenta a arquitetura de sistemas distribuídos propostos por outros autores para o estudo do escalonamento dinâmico distribuído.

A maioria dos trabalhos recentes em alocação dinâmica de recursos em sistemas frouxamente conectados tem estudado principalmente sistemas de estações de trabalho ligadas em redes de difusão ou em hipercubos. Ambos os tipos de sistemas têm uma limitada escalabilidade; o primeiro em razão das limitações de banda passante dos canais de comunicação compartilhados e o segundo tipo pelo número limitado de canais de comunicação disponíveis em um computador [43].

Macharia [43] enfoca o modelo celular de distribuição de carga. O modelo celular baseia-se no paradigma celular e tem por princípios fundamentais a distribuição das ações, a descentralização das decisões e a localidade na interação entre computadores. As arquiteturas celulares apresentam topologias MIMD com memória distribuída, e são constituídas por elementos processadores autônomos.

Um computador corresponde a célula do modelo de Macharia. Ele é modelado segundo Traub como duas camadas autônomas que interagem: a camada de rede e a camada do processador/memória. A camada de rede realiza assincronamente a distribuição de carga e a comunicação entre tarefas. Isto inclui a troca de informações sobre a carga computacional, a migração de tarefas e a comunicação entre as tarefas distantes. A camada do processador/memória realiza o escalonamento local, a gerência de memória e o encaminhamento das mensagens entre tarefas.

Os principais fatores que tornam o modelo celular atraente são a suas características intrínsecas de localidade e escalabilidade. Vários estudos têm sugerido que um alto nível de localidade existe na interação entre as tarefas e os computadores em muitas aplicações [43]. As arquiteturas celulares são especialmente escaláveis em topologias com um número fixo de vizinhos por computador.

Shin [64] utiliza um modelo bem genérico. Os computadores estão conectados por uma rede arbitrária. Cada computador está equipado com um processador de rede, que realiza a comunicação e a migração de tarefas, sem atrapalhar o seu processador.

Capítulo III

Escalonamento de Sistemas Distribuídos

Este capítulo apresenta o estágio atual na área de escalonamento para sistemas distribuídos, visando a sua aplicação em sistemas MAD. Inicialmente são apresentados os princípios fundamentais do escalonamento de sistemas de computadores. É realizada uma revisão da teoria de escalonamento distribuído, propondo uma classificação e analisando questões importantes, algumas polêmicas, quanto ao escalonamento distribuído. A última seção faz um retrospecto dos trabalhos anteriores, com ênfase nos resultados obtidos e nos algoritmos apresentados.

III.1 Princípios Fundamentais

O mapeamento, ao longo do tempo, de tarefas aos computadores de um sistema distribuído é denominado **escalonamento**. O escalonamento determina, conseqüentemente, a forma de utilização dos dois recursos fundamentais de um sistema distribuído: **capacidade de processamento** e **capacidade de comunicação**, proporcionados respectivamente pelos computadores e canais de comunicação.

A qualidade de um escalonamento deve ser avaliada a partir de duas de suas propriedades: desempenho e eficiência. O desempenho está relacionado aos resultados proporcionados pelo escalonamento, enquanto a eficiência refere-se aos custos do escalonamento. Assim, um algoritmo de escalonamento busca otimizar o desempenho do sistema eficientemente.

O objetivo primordial dos sistemas de alto desempenho, para os seus usuários, é conseguir executar em tempo hábil, aplicações que exigem muita computação. Como conseqüência o critério de desempenho mais importante para a avaliação destes

sistemas é o tempo de resposta das aplicações. Assim, na avaliação de escalonadores para estes sistemas, o mesmo critério é utilizado. O tempo de resposta depende não só do desempenho do escalonador mas também da sua eficiência. No escalonamento de aplicações paralelas, quanto menor o grão de paralelismo, maior a interferência do escalonador, e portanto, maior a importância da sua eficiência para o tempo de resposta.

Os problemas de determinação de uma atribuicão de tarefas a processadores, de modo a satisfazer a algum critério de desempenho pertencem a classe dos problemas NP-Completo [30]. Deste modo, o escalonamento de sistemas distribuídos é um problema NP-Completo e, como tal, não possui um método eficiente para a sua resolução.

Como conseqüência, um escalonamento ótimo, além de requerer o conhecimento a priori das características a tempo de execução de todas as tarefas da aplicação, também é computacionalmente muito caro [5]. Para que um escalonador dinâmico distribuído seja capaz de garantir um escalonamento ótimo, ele necessita da informação completa e acurada descrevendo o estado corrente da aplicação e também da seqüência de eventos futuros que afetarão o sistema [10].

Um sistema distribuído tem dois recursos computacionais: a capacidade de processamento e a capacidade de comunicação. A otimização da utilização de apenas um destes recursos computacionais pode levar à degradação do outro e assim, à degradação do sistema como um todo. Isto decorre do fato de que a utilização eficiente da capacidade de processamento está relacionada à minimização da ociosidade dos processadores, enquanto a utilização eficiente da capacidade de comunicação está relacionada à minimização da contenção dos canais de comunicação, objetivos que são conflitantes.

A contenção dos canais de comunicação pode ser reduzida se o escalonamento aproxima tarefas que se comunicam com maior freqüência. Esta aproximação, entretanto, pode aumentar a ociosidade dos processadores. Por outro lado, a maximização do uso da capacidade de processamento não implica em um maior uso desta capacidade diretamente pela APD – pode decorrer de uma maior atividade do software de controle – e, portanto, não garante um tempo mínimo de execução para a APD. Conseqüentemente, um escalonador para sistemas MAD deve procurar atender aos seus objetivos através de uma solução de compromisso, envolvendo a utilização da capacidade de processamento e da capacidade de comunicação.

O objetivo desta seção é evidenciar as questões teóricas pertinentes ao escalonamento para os sistemas MAD. A subseção a seguir define uma terminologia para a

expressão destas questões, que são analisadas na subseção III.1.2.

III.1.1 Classificação

A definição de parâmetros para a classificação das estratégias de escalonamento é importante para determinação de uma terminologia própria que facilite o entendimento e a expressão das questões relacionadas. Esta subseção apresenta uma proposta de classificação baseada nos trabalhos de Casavant e Kuhl [11] e de Baumgartner [7].

A classificação de um escalonador depende das seguintes características funcionais:

- **Escopo de atuação.** Um escalonador pode ser local ou global. Ele é **local** se atua apenas no escalonamento de tarefas a um computador. Um escalonador **global** decide em que computador cada tarefa deve executar, deixando a decisão de escalonamento das tarefas em um computador para o escalonador local.
- **Base de ação.** Define se as ações do escalonador decorrem das características estáticas do sistema ou do estado dinâmico do sistema. Se as informações utilizadas pelo escalonador têm que estar disponíveis antes da execução das tarefas, o escalonador é **estático**. Se elas são obtidas durante a execução, o escalonador é **dinâmico**. Normalmente os escalonadores estáticos realizam o escalonamento *antes* da execução das tarefas, durante a fase de geração destas. Os escalonadores dinâmicos sempre realizam o escalonamento *durante* a execução das tarefas, com base no comportamento apresentado pelo sistema.
- **Responsabilidade pela execução das ações de escalonamento.** Os escalonadores podem ser distribuídos ou não distribuídos. São **distribuídos** os escalonadores onde todos os computadores participam das atividades de obtenção de informações sobre o sistema e de execução das decisões da escalonamento.
- **Autoridade para a tomada de decisões de escalonamento.** Os escalonadores podem ser descentralizados ou centralizados. Nos escalonadores **centralizados** as decisões de escalonamento são tomadas por apenas um computador. Nos escalonadores **descentralizados** todos computadores estão autorizados a tomar decisões de escalonamento.
- **Autonomia na utilização de recursos.** Os escalonadores distribuídos descentralizados podem ser classificados de acordo com o grau de autonomia na determinação de como seus recursos devem ser utilizados. Eles podem ser

cooperativos ou não cooperativos. Se cada computador realiza ações independentemente das ações realizadas pelos demais, se preocupando apenas com as conseqüências locais destas ações, o escalonador é **não cooperativo**. Nos escalonadores **cooperativos** cada computador realiza as ações que lhe competem quanto ao escalonamento, mas todos os computadores trabalham de acordo com objetivos comuns quanto ao escalonamento global e eventualmente sacrificam o seu desempenho individual em benefício do desempenho do todo.

- **Estratégia para a distribuição da carga computacional.** Um escalonador pode ter uma estratégia de balanceamento de carga ou de divisão de carga. A estratégia de **balanceamento de carga** visa distribuir igualmente a carga pelos computadores. Já a estratégia de **divisão de carga** visa apenas manter todos os computadores trabalhando.
- **Mobilidade das tarefas.** Determinado pelo conjunto das tarefas que podem migrar. Os escalonadores com **migração restrita** só migram tarefas que não tenham ainda executado. Os escalonadores com **migração irrestrita** migram quaisquer tarefas.
- **Iniciativa das ações de escalonamento.** Refere-se à qual computador toma a iniciativa para a migração de tarefas. Um escalonador pode ser **iniciado pelo fornecedor** de tarefas, **iniciado pelo receptor** de tarefas, ou por ambos.

III.1.2 Escalonamento Distribuído

Nesta subseção são apresentados alguns conceitos e discutidas a aplicação e as características de algumas estratégias componentes de um algoritmo de escalonamento para sistemas distribuídos. São analisadas as bases de ação de um escalonamento, as estratégias para a distribuição de carga, a mobilidade das tarefas, o conceito de estabilidade de um escalonador, os componentes de uma estratégia de distribuição de cargas e o método para a avaliação da carga computacional utilizado pelos escalonadores.

Conforme a classificação descrita na subseção anterior, os escalonadores podem ser estáticos ou dinâmicos de acordo com sua base de ação. A escolha de uma destas opções depende das características das aplicações a serem escalonadas e da aplicabilidade, flexibilidade e custo que se deseja para o escalonador.

Os escalonadores estáticos podem produzir um escalonamento ótimo e podem apresentar um custo extra mínimo durante a execução, porque as decisões de esca-

lonamento são realizadas antes da execução da aplicação, utilizando o conhecimento a priori das características de execução da aplicação. Entretanto, eles são inviáveis para muitas aplicações paralelas, para as quais a obtenção deste conhecimento é impossível ou demasiadamente onerosa.

O escalonamento dinâmico permite a realização de ações sempre que surgirem desequilíbrios na distribuição das cargas computacionais aos computadores do sistema. Deste modo, o escalonador tem a possibilidade de transferir tarefas se, durante a execução de uma aplicação, ele perceber que a distribuição das tarefas pelos computadores piorar (de acordo com critérios estabelecidos pelo escalonador).

Em sistemas reais, tipicamente não se conhece, detalhadamente e a priori, a carga de processamento de uma aplicação [73]. Um número significativo de problemas importantes são difíceis de decompor de um modo direto. Além disso, a distribuição da carga de processamento pode se alterar de acordo com os dados de entrada e/ou com a evolução da computação. Considerando-se estas dificuldades e limitações presentes em sistemas reais, somente um escalonador dinâmico tem a capacidade de obter resultados com boa qualidade para qualquer tipo de aplicação paralela [33].

Os escalonadores dinâmicos podem se adaptar ao comportamento de execução de qualquer aplicação e podem apresentar um desempenho razoavel quase sempre, utilizando heurísticas simples, que implicam em um custo adicional quase desprezível por chamada.

A determinação da estratégia de distribuição das cargas computacionais é outra questão importante para o escalonamento. O princípio fundamental das políticas de balanceamento de carga é que a minimização do tempo total de execução de uma aplicação pode ser obtida através da minimização contínua da diferença de carga entre os computadores. Enquanto o princípio fundamental das políticas de divisão de carga é que a minimização do tempo total de execução de uma aplicação pode ser obtida pela manutenção de todos os computadores ocupados durante a execução da aplicação, sem que necessariamente a carga se mantenha igualmente distribuída pelos computadores.

Alguns autores tem assumido implicitamente que o balanceamento de carga é a única estratégia viável para a otimização da distribuição de carga. No entanto, para se obter um desempenho ótimo, é suficiente garantir a condição de não haver processadores ociosos, enquando houver tarefas prontas para executar [20]. Assim a estratégia de divisão de carga também é viável.

A ação principal de escalonamento é a migração de tarefas. A migração de uma tarefa que não foi ainda executada, pode ser realizada a um custo menor do que

o referente a migração de uma tarefa que já iniciou a sua execução. Além disso não são todos os sistemas distribuídos que permitem a migração de tarefas após sua primeira execução.

Uma questão importante relativa aos algoritmos de escalonamento refere-se à estabilidade de seu funcionamento. Na tentativa de explorar totalmente a capacidade de processamento, uma política complexa pode tomar decisões com base em repentinas e aparentemente más distribuições de carga. Esta estratégia de reagir a pequenas oscilações aliada à intrínseca inacurácia das informações sobre o estado global do sistema e a rapidez com que ele se altera, podem fazer a política reagir de um modo instável [20].

Um dos exemplos de instabilidade no funcionamento de escalonadores é o fenômeno denominado por *"state woggling"* [49], que ocorre quando a troca de mensagens entre as instâncias do escalonador sobrecarrega os canais de comunicação, devido a alterações freqüentes e alternadas dos estados dos computadores.

Outro exemplo de instabilidade ocorre quando o sistema entra em um estado em que tarefas estão migrando constantemente em uma tentativa de melhor distribuir a carga entre os computadores [25]. Este fenômeno, denominado *"processor trashing"* [9], causa um desperdício quase total da capacidade computacional do sistema apenas nas atividades de escalonamento. Os algoritmos de balanceamento de carga são mais suceptíveis à instabilidade do que os algoritmos de divisão de carga.

Os componentes lógicos de um escalonador são:

- **Método de avaliação da carga computacional.** Estabelece os índices para a avaliação da carga computacional e a forma como são calculados.
- **Gerência de informações.** Determina quando, como e quais informações, são trocadas entre as instâncias do escalonador.
- **Política de transferência.** Decide quando realizar uma migração de tarefa, qual tarefa migrar e de/para qual computador migrar.

A escolha de um método para a avaliação da carga computacional é um problema difícil, para o qual não há uma solução completamente satisfatória [49]. Esta avaliação é uma informação útil para a identificação pelo escalonador das oportunidades para a realização das migrações de tarefas.

Uma solução ideal previria os tempos de execução das tarefas, o que é impossível, sem a execução anterior das tarefas. Ao invés disso, é necessário estimar os tempos de execução a partir de parâmetros mensuráveis. Eles podem incluir fatores dependentes do tempo, como demandas por recursos, *"mixes"* de instruções, número de

tarefas, e fatores dependentes dos computadores, como arquitetura e velocidade dos processadores e estratégias do sistema operacional [49].

Na execução de um escalonador dinâmico, a carga computacional precisa ser avaliada com grande freqüência, portanto seu cálculo deve ser muito eficiente. Isto torna o uso de um grande número de parâmetros inconveniente. Ao invés, um pequeno subconjunto dos parâmetros deve ser utilizado junto com heurísticas para a avaliação da carga de processamento [49]. Muitos algoritmos para escalonamento dinâmico distribuído utilizam o número de tarefas presentes como medida da estimativa da carga de processamento. Uma outra medida que pode ser utilizada é o número de tarefas prontas.

III.2 Trabalhos Anteriores

Em 1973, Farber [24] apresentou o algoritmo de "*bidding*" (leilão) para escalonamento dinâmico distribuído. Sempre que um novo processo é criado, o escalonador difunde pela rede uma requisição de lances ("*bids*") para a execução do processo. Após todos os computadores haverem retornado seus lances, o escalonador transfere o processo para o computador que oferecer o melhor lance. Smith [66], em 1980, propôs uma extensão deste algoritmo, na tentativa de criar uma estrutura genérica para a resolução distribuída de problemas.

Em 1985, Barak [5] propôs um algoritmo distribuído para o escalonamento de processos em uma rede local de estações de trabalho. Este algoritmo utiliza a estratégia de balanceamento de carga, e requer que a rede de comunicação conecte completamente todos os computadores por conexões diretas.

Ni e Gendreau [49] apresentaram, em 1985, um algoritmo distribuído de escalonamento com a estratégia de balanceamento de carga. O algoritmo "*drafting*" (licitação) tem como características importantes a independência em relação à topologia da rede e o fato de buscar uma relação de compromisso entre dois objetivos contraditórios: maximizar a utilização dos processadores e minimizar os custos de comunicação.

No algoritmo "*drafting*" os computadores apresentam um estado de carga de acordo com o número de tarefas presentes. Os computadores são classificados em subcarregados, normalmete carregados e sobrecarregados. Cada computador tem uma tabela com o último estado informado por seus vizinhos.

Quando um computador está subcarregado e tem pelo menos um vizinho sobrecarregado ele inicia o processo de "*drafting*", enviando uma requisição de "*draft*"

(proposta) a estes vizinhos. Após todos os vizinhos responderem esboçando o quão vantajoso é a migração que podem realizar, um "draft" mínimo aceitável é calculado e enviado ao computador que enviou o melhor "draft". Este computador ao receber o "draft" mínimo aceitavel, envia uma tarefa, se ainda puder atendê-lo, ou uma mensagem de "tarde demais". Os processos só migram uma vez, como forma de evitar a instabilidade do algoritmo.

Em 1986, Eager, Lazowska e Zahorjan [19] realizaram uma comparação entre diversos algoritmos de escalonamento distribuído dinâmico, todos baseados em limiares, alguns iniciados pelo fornecedor e outros iniciados pelo receptor. Eles chegaram às seguintes conclusões:

- Tanto as estratégias de escalonamento iniciadas pelo fornecedor, quanto as iniciadas pelo receptor apresentam um melhor desempenho que nenhuma estratégia.
- As estratégias iniciadas pelo fornecedor são preferíveis para as cargas computacionais leves a moderadas.
- As estratégias iniciadas pelo receptor são preferíveis para as cargas computacionais pesadas, mas apenas se o custo de transferência de tarefas para estas estratégias for comparável ao custo de transferência de tarefas para as estratégias iniciadas pelo fornecedor.
- Se o custo de transferência de tarefas para as estratégias iniciadas pelo receptor for significativamente superior ao das estratégias iniciadas pelo fornecedor (porque, por exemplo, as tarefas ativas também podem ser transferidas), então as estratégias iniciadas pelo fornecedor uniformemente apresentam um melhor desempenho.
- A modificação das estratégias iniciadas pelo receptor para que transfiram apenas tarefas recem criadas conduz a um desempenho insatisfatório.

Em [20] os mesmos autores estudam três algoritmos distribuídos para escalonamento dinâmico, todos iniciados pelo fornecedor e utilizando limiares, mas que apresentam políticas com complexidades diferentes para a escolha do computador que recebe as tarefas acima do limiar. Neste trabalho, o objetivo é determinar qual o nível de complexidade adequado para as políticas de escalonamento dinâmico distribuído. As conclusões são as seguintes:

- Políticas de escalonamento dinâmico distribuído extremamente simples – que coletam uma pequena quantidade de informação de estado e que utilizam esta informação de modo simples – apresentam um desempenho muito melhor do que se não houvesse uma política de escalonamento.

- Estas políticas extremamente simples, de fato, conseguem um desempenho que se esperaria de políticas complexas, que coletam um grande volume de informações e que tentam realizar a melhor decisão – políticas cuja viabilidade é questionável.
- Estes resultados são válidos com relação a uma grande faixa de parâmetros de sistema.

Em 1986, Hsu [32] descreveu três algoritmos probabilísticos para balanceamento dinâmico de carga. Os algoritmos tomam decisões baseadas em estatísticas do estado de execução do sistema. Hsu assume que os computadores estão completamente conectados e que trocam informações sempre que há uma modificação qualquer no estado do sistema.

```
Sempre que houver modificação na carga interna:
se carga $geq$ HIGH
        estado := ABUNDANT;
senão se carga < LOW
        estado := IDLE;
senão  estado := NEUTRAL;

Sempre que houver modificação no potencial dos vizinhos
ou na carga interna:
caso estado seja:
        IDLE:
                potencial := 0;

        NEUTRAL:
                potencial := 1 + min {potencial dos vizinhos};
                se potencial > WMAX
                        potencial = WMAX;
        ABUNDANT:
                se min {potencial dos vizinhos} $geq$ WMAX
                        potencial := WMAX;
                senão
                        potencial = min {potencial dos vizinhos} + 1;
                        transfere uma tarefa para o vizinho de
                        menor potencial;
                        recomece;
fim do caso

Se o potencial se alterou
        avise aos vizinhos;
```

Figura III.1: Algoritmo Gradiente

Ainda em 1986, Lin e Keller [39] propuseram um método para balanceamento

dinâmico de carga para multicomputadores. O método se baseia em uma abordagem dirigida por demanda, o modelo gradiente, que transfere tarefas excessivas para o vizinho disponível mais próximo via um plano de gradiente, estabelecido pelas demandas dos processadores disponíveis. O algoritmo é totalmente distribuído e assíncrono. A figura III.1 mostra o algoritmo gradiente.

O modelo gradiente é um método de balanceamento de carga localizado, onde todos os computadores só interagem com os vizinhos. Um balanceamento global é obtido pela propagação das demandas em todo o sistema e pelos sucessivos refinamentos da distribuição de carga. Uma demanda é sempre iniciada por um computador disponível e então espalhada pelo sistema. Uma demanda termina quando é satisfeita pela recepção de uma tarefa para executar, ou quando o sistema estabiliza.

Hác e Jin [31] analisaram em 1987, um algoritmo descentralizado para o escalonamento dinâmico distribuído de processos longos, que utiliza a migração de processos e arquivos e que foi efetivamente implementado. Neste algoritmo os computadores pouco carregados consultam todos os demais para descobrir em quais há trabalho disponível para ser transferido. O estado de cada computador é sempre difundido a todos os demais.

Ainda em 1987, Casavant e Kuhl estudaram [10] a relação entre o nível de informação global requerida por uma política de escalonamento dinâmico e o desempenho e eficiência do escalonamento. O estudo abrangeu três conjuntos de algoritmos para escalonamento dinâmico. Os três algoritmos são distribuídos, cooperativos e têm a estratégia de balanceamento de carga. Estes algoritmos utilizam níveis crescentes de informação global. Os autores chegaram às seguintes conclusões:

- A tentativa de obter dinamicamente informações que descrevam o estado global do sistema não é benéfica.
- É melhor utilizar apenas informações sobre um pequeno subconjunto do sistema, as quais são acuradas, do que tentar manter informações sobre todo o sistema, as quais podem não ser acuradas.
- O tempo adicional necessário à transmissão de informações completas atrasa o sistema, aumentando os tempos de resposta.
- A coleta dinâmica de tanta informação, pode utilizar tão significativamente recursos computacionais, que se torne contraprodutiva.
- Estes resultados não são necessariamente aplicáveis a qualquer computação distribuída, mas àquelas pertencentes a classe de computações identificada por Stankovic [68] como funções replicadas estocásticas.

- A obtenção e utilização de uma pequena quantidade de informações sobre uma única característica do estado global é uma boa estratégia para se evitar a instabilidade do sistema.

Em fevereiro de 1988, Casavant e Kuhl publicaram um trabalho muito importante [11] para o entendimento do problema de escalonamento distribuído. Com este trabalho eles foram os primeiros a propor uma taxonomia abrangente para o problema e uma sistematização das suas possíveis soluções. Assim, além de prover uma terminologia comum para o problema, os autores criaram um mecanismo de classificação necessário a um melhor entendimento dos trabalhos anteriores, que são apresentados em uma bibliografia comentada.

Em junho de 1988, Ferguson, Yemini e Nikolao [25] aplicaram conceitos de microeconomia à alocação e ao compartilhamento de recursos computacionais em um sistema distribuído. A abordagem microeconômica resulta em um algoritmo orientado pela competição, ao invés da cooperação. Este algoritmo se destaca ainda pela unificação na alocação dos recursos de processamento e de comunicação.

O funcionamento do algoritmo é baseado na atribuição de preços aos recursos computacionais e valores às demandas das tarefas. Esta característica acarreta duas desvantagens para o algoritmo:

- É necessário o conhecimento a priori das demandas das tarefas. Isto limita o seu escopo de aplicação.
- O estabelecimento dos preços e valores exige muitos cálculos e comparações. Conseqüentemente, os custos de execução do algoritmo só são viáveis para o escalonamento de processos longos.

Assim, o algoritmo é inaplicável quando as demandas computacionais não são conhecidas a priori e é ineficiente para o escalonamento das tarefas de uma APD.

Em junho de 1988, Pulidas, Towsley e Stankovic [54] apresentaram um estudo, realizado a partir de simulações, do comportamento de um algoritmo descentralizado otimizante com o objetivo de estimar os melhores valores para os limiares de um algoritmo dinâmico de escalonamento baseado em limiares, com a estratégia de balanceamento de carga, durante o escalonamento. O algoritmo entretanto é muito oneroso, pois envolve cálculos excessivamente complexos para ser utilizado em tempo real no escalonamento dinâmico de aplicações paralelas de grão médio ou pequeno.

Chowkwanyun e Hwang [13] propuseram em setembro de 1988 um algoritmo híbrido, baseado em limiares, cuja iniciativa para migração de tarefas tanto pode ocorrer por parte do fornecedor como por parte do receptor, de acordo com a carga do sistema. O modo de funcionamento iniciado pelo receptor é baseado no algoritmo

"drafting" [49]. Enquanto o modo de funcionamento iniciado pelo fornecedor é baseado no algoritmo gradiente [39].

Todos os computadores iniciam o seu funcionamento no modo iniciado pelo fornecedor. O modo de operação de um computador muda de acordo com o número dos seus vizinhos que estão sobrecarregados. Se este número for superior a um limiar o computador opera no modo iniciado pelo receptor ("drafting"), em caso contrário ele opera no modo iniciado pelo fornecedor (gradiente). Os computadores processam todas as mensagens de escalonamento recebidas, independentemente do seu modo de operação.

O objetivo deste algoritmo é combinar o melhor desempenho dos algoritmos iniciados pelo receptor quando o sistema está sobrecarregado, com o melhor desempenho dos algoritmos iniciados pelo fornecedor quando o sistema está subcarregado. Os autores, no entanto, não explicitaram completamente o algoritmo. Não é especificado precisamente como os dois algoritmos são combinados.

Em setembro de 1988, Baumgartner e Wah [7] apresentaram uma nova taxonomia para o escalonamento em sistemas computacionais distribuídos. Eles também propuseram uma estratégia de escalonamento que se baseia em uma característica particular das redes locais de multiacesso. A capacidade de difusão das redes locais de multiacesso é utilizada para a identificação dos computadores com a maior e a menor carga de processamento.

Estes autores não utilizaram a taxonomia proposta por Casavant e Kuhl [11] principalmente porque consideraram que neste trabalho há uma sobreposição entre a classificação das estratégias de escalonamento e os requerimentos para estas estratégias. Baumgartner e Wah consideraram o balanceamento de carga como um requerimento da estratégia de escalonamento, que garante um melhor desempenho, no que descordam Casavant e Kuhl.

Neste trabalho de tese consideramos que o balanceamento de carga é, na realidade, apenas uma das estratégias de distribuição de carga, que não apresenta necessariamente o melhor desempenho em todos os casos, conforme é explicado na seção III.1.2.

Baumgartner e Wah observam que o escalonamento de tarefas que se comunicam é um problema mais difícil, porém mais realista, do que o escalonamento de tarefas que não se comunicam. Eles também colecionaram as seguintes conclusões, ainda não mencionadas, de trabalhos anteriores:

- Balanceamento de carga é útil já que em sistemas distribuídos com mais de uma dezena de computadores, freqüentemente ocorre um desequilíbrio na dis-

tribuição da carga de processamento [41].

- As estratégias dinâmicas têm um melhor potencial do que as estáticas.
- As estratégias centralizadas podem ter problemas de confiabilidade e podem se transformar em um gargalo no sistema.
- A comunicação de informações de escalonamento não deve interferir com a comunicação regular entre as tarefas da aplicação.

O trabalho de Efe e Groselj [22] em 1989, tem por finalidade verificar diferentes suposições e modelos empregados na literatura, através da simulação de uma política genérica de escalonamento dinâmico. Além disto eles propuseram dois algoritmos centralizados para o escalonamento dinâmico.

Eles investigam as seguintes questões:

1. Uma política de escalonamento deve tentar balancear a carga computacional entre os computadores, ou basta manter todos os computadores carregados ? Enfim, qual é a melhor estratégia para a distribuição de carga ?
2. O que acontece se a suposição de que os custos de escalonamento são desprezíveis for falsa ?
3. Como os vários níveis de custos adicionais devidos a transferência de tarefas influenciam o desempenho de uma política de escalonamento ?

Eles encontram as seguintes respostas:

1. Sob uma carga computacional moderada ou leve, não há uma significativa diferença de desempenho entre uma variedade de estratégias de distribuição de carga. Sob carga pesada, contudo, a estratégia de divisão de carga, ou seja, manter todos os processadores ocupados apresenta um desempenho melhor do que a estatégia de balancear a carga.
2. Quando os custos de escalonamento e de transferência são pequenos, o desempenho das políticas de escalonamento pode ser muito bom, confirmando resultados anteriores. Se no entanto, estes custos não são pequenos, então a utilizacão de um algoritmo de escalonamento pode apresentar um desempenho pior do que o desempenho apresentado sem a sua utilização.
3. Um número excessivo de transferências de tarefas pode degradar o sistema pesadamente, especialmente se os atrasos decorrentes das transferências aumentam em relação ao tempo de serviço médio das tarefas.

Como conclusão eles recomendam que uma política de escalonamento deve tentar maximizar a taxa de sucesso da escolha apropriada dos destinos das tarefas transferidas, enquanto minimiza os custos adicionais de controle do escalonamento.

Os algoritmos propostos visam diminuir os custos adicionais de controle, relativos às mensagens com a carga dos computadores. No primeiro algoritmo os níveis de carga são estimados através de uma heurística simples. O segundo representa um melhoramento do primeiro ao substituir transferências desnecessárias por uma consulta.

Shin e Chang [64] propuseram, em agosto de 1989, um algoritmo dinâmico para o escalonamento de sistemas distribuídos de tempo real. A principal característica do algoritmo apresentado refere-se à estratégia da gerência de informações. Eles avaliam o desempenho do algoritmo proposto tanto analiticamente, quanto por simulação.

A estratégia da gerência de informações de escalonamento visa utilizar:

- Meios eficientes de se obter e de se atualizar informação de estado; a obtenção de informação de estado não deve prejudicar as comunicações normais, como as comunicações entre tarefas e as transferências de tarefas.
- Um método automático para a seleção de um receptor, se há mais de um computador subcarregado, minimizando a possibilidade de que outros computadores sobrecarregados transfiram simultaneamente tarefas a um mesmo computador subcarregado.

As soluções apresentadas são:

- Cada computador só mantém a informação do estado de um pequeno conjunto de outros computadores, denominado *"buddy set"*.
- As mensagens de mudança de estado de um computador é difundida para os computadores do seu *"buddy set"*, para obtenção e atualização das informações de estado. Só são anunciadas as mudanças de subcarregado para sobrecarregado e vice-versa. O tráfego adicional destes anúncios pode ser controlado ajustando os limiares que definem o estado de um computador.
- Cada computador tem uma lista estática de preferidos para a recepção de tarefas, que é função da topologia do sistema. Os computadores são permutados nas listas de preferidos de modo que para o sistema como um todo não haja privilégios para um dos computadores. Os computadores que se tornam sobrecarregados saem da lista de receptores, e voltam aos seus lugares quando se tornam subcarregados novamente.

Assim, um computador sobrecarregado é capaz de tomar uma decisão de transferir uma tarefa baseado apenas na informação de estado já disponível, sem ter que esperar pelo resultado de consultas.

Macharia [43] propôs em 1989, uma classe de algoritmos de escalonamento. Ele denomina distribuição celular de carga a classe dos algoritmos dinâmicos descentralizados de escalonamento de aplicações com paralelismo de grão fino a médio em sistemas MIMD altamente paralelos, com memória distribuída. Ele propôs ainda três algoritmos de escalonamento.

Esta classe de escalonadores apresenta como características importantes a localidade de interação, a escalabilidade e a expansibilidade virtualmente ilimitada. Estes escalonadores são especialmente bem adaptados para as cargas computacionais que se difundem.

Os algoritmos propostos por Macharia são algoritmos síncronos, iniciados pelo fornecedor e adotam o balanceamento como estratégia de distribuição de carga computacional. O que os distingue são os níveis crescentes das ações em direção ao balanceamento da carga computacional.

Macharia assume que os custos de processamento para a migração de tarefas e a troca de informações entre as instâncias dos escalonadores são desprezíveis. Apesar de considerar que a comunicação entre tarefas é um dos custos mais importantes nas aplicações paralelas de grão fino ou médio, os algoritmos apresentados não a consideram.

Em março de 1990, Chowdhury [12] propôs um algoritmo dinâmico, iniciado pelo fornecedor, de divisão de carga, denominado *"greedy"*. Este novo algoritmo é comparado com uma algoritimo baseado em limiares e quase sempre apresenta um melhor desempenho.

O algoritmo *"greedy"* é muito simples. Sempre que uma tarefa é criada, em um computador com n tarefas, procura-se um computador que tenha $f(n)$ tarefas ou menos. Se houver, transfere-se a tarefa. A função $f(n)$ que apresenta o melhor desempenho, e que portanto, foi escolhida é $f(n) = n \; div \; 3$.

Mirchan [45], em 1990, estuda as características de desempenho de algoritmos simples de escalonamento dinâmico distribuído, por divisão de carga em sistemas distribuídos heterogêneos. Assume que os atrasos para transferência de tarefas não são desprezíveis, nem a obtenção de informações dos nós remotos. Analisa os efeitos destes atrasos em dois algoritmos: *Forward* e *Reverse*. Estes são exemplos dos algoritmos clássicos para escalonamento dinâmico distribuídos, baseados em limiares, iniciado pelo receptor (Reverse) e iniciado pelo fornecedor (Forward) de carga computacional.

Ele considera que enquanto os trabalhos anteriores analisam vários aspectos de distribuição de carga, o problema dos custos de comunicação não foi ainda investi-

gado em grande detalhe. Apesar disto, não considera a comunicação entre tarefas.

A avaliação dos algoritmos é realizada analiticamente. Os computadores são modelados por cadeias de Markov, que são resolvidas utilizando a técnica de solução da matriz geométrica.

A partir do retrospecto dos trabalhos anteriores, pode-se verificar que:

> *Não há qualquer trabalho anterior que efetivamente proponha, ou avalie algoritmos para o escalonamento dinâmico distribuído de aplicações paralelas com grão médio ou fino em sistemas altamente paralelos como os sistemas* MAD.

Não há um trabalho anterior que avalie, com base em modelos realistas, o consumo efetivo dos recursos computacionais (capacidade de processamento e capacidade de comunicação) realizado tanto pela aplicação, como pelo software de controle do sistema.

Um modelo realista de uma APD conforme explicado na seção IV.5 tem que reconhecer que há muita interação entre as tarefas de uma APD. Esta interação é realizada através de trocas de mensagens, que consomem uma parte não desprezível da capacidade de comunicação do sistema, que assim como a capacidade de processamento, também é limitada.

Um modelo realista do sistema computacional para a execução de APD tem que reconhecer que os gastos de recursos computacionais do sistema realizados pelo software de controle, sempre que é ativado, podem ter uma ordem de grandeza próxima da ordem de grandeza dos gastos das atividades de uma tarefa da aplicação. Assim a utilização de recursos computacionais pelo software de controle, seja para a realização das decisões de escalonamento, seja para possibilitar a comunicação entre tarefas, não são desprezíveis e, portanto têm que ser contabilizados.

Capítulo IV

Metodologia de Avaliação

Este capítulo apresenta a metodologia para a avaliação de escalonadores dinâmicos distribuídos empregada neste trabalho. Inicialmente o objetivo da avaliação é formalizado. A seguir os trabalhos anteriores relacionados são apresentados. Na seção IV.3 os dois métodos básicos de avaliação são apresentados suscintamente e um dos métodos é escolhido em função de justificativas apresentadas. A seguir (seção IV.4) são discutidas as formas de modelamento do sistema. O capítulo termina com a apresentação de um modelo de execução das APD e com um modelo de funcionamento dos sistemas MAD.

IV.1 Objetivo da Avaliação

A avaliação tem por objetivo comparar o desempenho de escalonadores dinâmicos distribuídos em sistemas MAD. O método a ser adotado deve considerar as características peculiares dos sistemas MAD e das suas aplicações: um paralelismo de grão médio e comunicação intensa entre as unidades de execução da aplicação.

Como conseqüência destas características, a avaliação do desempenho geral dos escalonadores implica na consideração da utilização de recursos computacionais direta e indiretamente pela aplicação. A utilização indireta, refere-se à realização das atividades necessárias para a execução da aplicação, como as atividades decorrentes do mecanismo de escalonamento e do mecanismo de comunicação.

IV.2 Trabalhos Anteriores

Em relação à escolha de um método para a análise dos escalonadores dinâmicos distribuídos para multicomputadores, Daniel Reed [57] considera que a maior parte

dos modelos analíticos assume um comportamento estacionário, portanto eles não podem ser aplicados ao estudo de computações paralelas dependentes do tempo.

Em particular, os modelos para sistemas distribuídos de computação dependente do tempo devem incluir cargas de processamento que variam ao longo do tempo, a distribuição dessas cargas às tarefas e o mapeamento destas tarefas aos processadores usando apenas um conhecimento parcial do estado global do sistema. Considerando que não há técnica analítica conhecida capaz de representar precisamente este comportamento, Daniel Reed também adota a simulação.

Baccelli [3] propõe um novo modelo de redes de filas para a avaliação de sistemas multiprogramados, multitarefas e multiprocessados, cujas cargas computacionais consistem de programas paralelos. O modelo assume que todos os programas têm a mesma estrutura de tarefas e que o mapeamento das tarefas aos processadores é *estático*.

Baccelli [3] ainda atesta que o modelamento para a análise do desempenho de sistemas multiprogramados, multitarefas e multiprocessados, onde a alocação de tarefas aos processadores é dinâmica ou os programas têm uma estrutura variável ao longo do tempo de execução é um *problema completamente aberto.*

IV.3 Determinação do Método de Avaliação

Um método para a avaliação de desempenho pode ser de natureza matemática, ou lógico matemática. É matemático e chamado de **método analítico**, se o sistema é representado por uma série de equações matemáticas, através das quais o comportamento do sistema é obtido analiticamente, pela definição dos parâmetros do sistema e pela solução das equações. O método lógico matemático utiliza **simuladores**. Eles representam o sistema por uma estrutura lógico matemática, que ao ser exercitada mimetiza o seu funcionamento.

A solução por simulação é mais genérica e permite uma maior flexibilidade, podendo ser aplicada a situações mais complexas. Não há restrições para a sua aplicação neste trabalho. Além disso, nos simuladores híbridos os algoritmos em avaliação podem ser efetivamente executados, o que permite a especificação e a avaliação completa de qualquer algoritmo. No entanto, os resultados apresentados por simuladores são aproximados e podem requerer a realização de muita computação.

A solução analítica, é geralmente mais rápida, obtém resultados exatos e é preferida quando pode ser aplicada. No entanto, para modelos mais complexos, que são expressos por *redes de filas*, pode ser necessário adotar muitas hipóteses de

simplificação, para possibilitar a sua resolução.

Os métodos para a resolução analítica de redes de filas, em razão de limitações de ordem matemática, impõem algumas restrições ao sistema [67]. Uma destas restrições inviabiliza a utilização da solução analítica neste trabalho. Esta restrição e a sua conseqüência para este trabalho é a seguinte:

- As decisões de roteamento têm que obedecer a um conjunto de probabilidades de ramificação. Para o escalonamento isto implica em que a decisão de transferir ou não uma tarefa e para onde, obedeça a um conjunto de probabilidades que se mantém constante durante a execução da APD e, portanto, independe do estado corrente do sistema. Considerando que os escalonadores dinâmicos, por definição, tomam suas decisões exatamente em função do estado corrente do sistema, esta restrição inviabiliza a avaliação destes escalonadores.

Como decorrência, o método de avaliação adotado neste trabalho é o método lógico matemático, implementado por um simulador híbrido, o simulador de escalonadores dinâmicos distribuídos (DDSS).

IV.4 Modelamento de Sistemas

Um modelo de um sistema é uma abstração deste sistema, e corresponde a uma descrição deste sistema. Um modelo para simulação é uma representação lógico matemática de um sistema, que pode ser exercitada experimentalmente em um computador digital [67].

Os modelos para simulação podem ser discretos ou contínuos. O comportamento do sistema a ser modelado determina qual deles é o mais adequado em cada situação. O comportamento lógico de um sistema computacional digital é por definição discreto, portanto pode ser mais adequadamente modelado por um modelo discreto.

A formulação de um modelo para a simulação discreta deve obedecer a uma estrutura conceitual. Ela determina o enfoque dentro do qual, as relações funcionais entre os elementos do sistema são percebidas e descritas [67]. Há três estruturas conceituais básicas:

- **Simulação orientada a eventos**: os sistemas são modelados pela definição das mudanças no seu estado, que se processam quando da ocorrência de cada evento possível.
- **Simulação orientada a atividades**: os sistemas são modelados pela definição das atividades nas quais os componentes do sistema se envolvem.

- **Simulação orientada a processos**: os sistemas são modelados pela definição dos processos através do qual os componentes do sistema passam.

O propósito da simulação neste trabalho é a avaliação de algoritmos de escalonamento. Para possibilitar uma maior flexibilidade de ação a estes algoritmos, eles têm que ter a capacidade de tomar decisões em função da ocorrência de cada evento significativo para o escalonamento. Para que eles possam ser notificados da ocorrência destes eventos, o simulador deste trabalho tem que ser orientado a eventos.

O modelamento de um sistema para simulação discreta exige uma caracterização do seu funcionamento. Um sistema pode ser caracterizado em vários níveis de abstração; desde o nível funcional até o nível físico. Uma caracterização funcional reflete apenas as propriedades relativas a um sistema, enquanto, no outro extremo, uma caracterização física depende da configuração do sistema e de seus componentes, pois ela descreve o sistema em termos do consumo de seus recursos [27].

As caracterizações físicas são requeridas pelas técnicas mais utilizadas para modelamento de sistemas e pelos procedimentos disponíveis para a criação de modelos de cargas computacionais. Portanto, qualquer caracterização de um sistema terá que ser transformada para uma caracterização física [27].

O funcionamento geral do sistema pode ser caracterizado por dois modelos complementares: o modelo da carga computacional e o modelo do sistema computacional. Neste trabalho, eles são respectivamente: o **modelo de execução das** APD e o **modelo de funcionamento dos sistemas** MAD. Eles caracterizam o sistema fisicamente. O primeiro caracteriza uma APD em termos de demanda por recursos computacionais, enquanto o segundo caracteriza o sistema computacional em termos da disponibilidade destes recursos.

IV.5 Modelo de Execução das Aplicações Paralelas Distribuídas

Esta seção apresenta o modelo de carga computacional que é utilizado neste trabalho. A definição deste modelo determina a forma de representação do funcionamento das APD para o processo de simulação. Inicialmente são definidas as características, que o modelo deve apresentar, para melhor representar o comportamento das APD. A seguir o modelo adotado, criado para este trabalho, é descrito detalhadamente.

A adoção de um modelo criado especialmente para este trabalho, se deve ao fato de que segundo Domenico Ferrari [27] não há, e é importante que seja criado, um

modelo de carga computacional, que caracterize o comportamento das APD em um sistema frouxamente conectado (como os sistemas MAD).

Ele argumenta que o principal fonte de problemas para a caracterização das cargas computacionais de sistemas distribuídos é o fato de estarem envolvidos grupos de processos que interagem entre si. A comunicação entre processos é uma característica distinta delas. Esta comunicação tem que ser adequadamente representada em qualquer modelo destas cargas computacionais, já que o desempenho dos sistemas distribuídos pode ser profundamente influenciado por ela.

O modelo de execução das APD é uma proposta de modelo de carga computacional para APD, que considera a existência de grupos de processos que interagem entre si, através de trocas de mensagens.

IV.5.1 Princípios Básicos

Uma APD é composta por tarefas (unidades de execução seqüencial) paralelas que colaboram para a solução de um problema. Esta colaboração implica na interação entre as tarefas, que nos MAD ocorre através da troca de mensagens. A seguinte regra geral é admitida como obedecida pelas aplicações paralelas:

> *"quanto mais intensamente paralela for a estruturação de uma aplicação, mais numerosas e simples serão as suas tarefas componentes e mais intensa será a interação entre elas"*

Um modelo de carga computacional, como o modelo de execução de APD, pode ser **determinístico** ou **estocástico**. Um modelo é determinístico se todos os valores assumidos por suas variáveis podem ser determinados. Em contraposição, em um modelo estocástico, pelo menos uma variável assume valores probabilísticos, que portanto não podem ser determinados.

Como conseqüência da regra geral das aplicações paralelas, um modelo de execução de APD apresenta um grande número de variáveis e de valores que elas assumem. Esta complexidade impõe a escolha de um método de modelamento que conduza a uma descrição compacta da carga computacional. Normalmente a quantidade de informações necessárias para representar adequadamente os vários parâmetros da carga computacional e suas flutuações pode ser drasticamente reduzido se for adotado um modelo estocástico. Ou seja, os modelos estocásticos geralmente permitem uma descrição mais compacta da carga computacional. Por isto, um modelo estocástico de execução de APD é adotado neste trabalho.

Uma outra conseqüência da regra geral das aplicações paralelas, é a maior significância dos custos de escalonamento para as aplicações intensamente paralelas,

como as APD. Portanto deve-se considerar que os escalonadores disputam recursos computacionais com a APD em execução em um sistema MAD. Assim, o modelo da carga computacional deve ser descrito em termos do consumo de recursos computacionais e os escalonadores devem ter os seus gastos de recursos computacionais contabilizados.

Um modelo estocástico para a execução de APD deve modelar as suas principais características: *paralelismo, sincronização* e *comunicação interna*. Deve ainda prever que partes da aplicação interagem mais intensamente. Neste trabalho, a concorrência é modelada pela decomposição da aplicação em tarefas que podem executar simultaneamente, enquanto a sincronização e a comunicação interna são modeladas pela troca de mensagens entre as tarefas.

IV.5.2 Descrição do Modelo

Durante a sua existência, uma tarefa executa uma seqüência de atividades. Uma atividade pode ser um processamento (execução contínua de uma seqüência de instruções), a criação de uma nova tarefa, uma transmissão de mensagem, ou uma recepção de mensagem. A seqüência mínima de atividades de uma tarefa é constituída por um processamento. A primeira e a última atividades de uma tarefa são necessariamente processamentos.

As quantidades de instruções dos processamentos obedecem a uma distribuição exponencial de média constante. O último processamento de uma tarefa é o primeiro gerado, cuja duração seja inferior a um limite mínimo. O início da execução de cada processamento de uma tarefa, à exceção do primeiro, é precedido pela recepção, se houver, das mensagens a ela dirigidas. E ao final de cada processamento, à exceção do último, é decidido o envio de uma mensagem pela tarefa em execução.

O processador envolvido na execução de uma tarefa é obviamente o pertencente ao computador ao qual ela está alocada. Já a determinação dos canais de comunicação envolvidos em uma troca de mensagens entre duas tarefas da APD depende da localização destas tarefas e do mecanismo que controla esta comunicação. Para que uma tarefa possa ser localizada, ela tem que ser identificada. Assim é necessário que cada tarefa da APD tenha uma identidade própria.

Uma mensagem entre duas tarefas tem as seguintes características: identidade da tarefa emissora, identidade da tarefa destinatária e tamanho da mensagem. O tamanho da mensagem obedece a uma distribuição exponencial de média constante. São consideradas válidas somente as mensagens cujo tamanho é superior a um limite mínimo. A validade, ou não, de uma mensagem decide o envio, ou não, de uma

mensagem pela tarefa em execução. A ocorrência de uma mensagem válida acrescenta uma atividade de transmissão de mensagem à tarefa emissora e de recepção de mensagem à destinatária.

Como as atividades de uma tarefa são executadas seqüencialmente, sempre que há o envio de uma mensagem há uma sincronização explícita. A tarefa destinatária só executa atividades posteriores à atividade de recepção de uma mensagem, após a sua efetiva recepção. A transmissão, no entanto, é assíncrona. A tarefa emissora prossegue as suas atividades, sem aguardar a confirmação da recepção da mensagem.

As tarefas são separadas em grupos disjuntos para fins de comunicação. As tarefas de um mesmo grupo apresentam uma afinidade maior entre si e portanto uma maior interação. Uma mensagem pode ser enviada para uma outra tarefa do mesmo grupo, ou para uma tarefa dos demais grupos, determinando dois conjuntos de possíveis destinatárias. O valor de privilégio é definido como a probabilidade da tarefa destinatária pertencer ao mesmo grupo que a emissora. As tarefas do conjunto escolhido têm a mesma probabilidade de serem escolhidas.

A inclusão de uma tarefa em um grupo é realizada no momento de sua criação. Há uma probabilidade constante que determina se a tarefa em criação iniciará um novo grupo, ou se será incluída no grupo de sua criadora. Esta forma de agrupamento prevê a criação de novos grupos, prevê uma provável afinidade entre as tarefas criadora e criada e independe da localização das tarefas e portanto do escalonador em uso.

A criação de uma tarefa é determinada pela sua tarefa criadora e depende exclusivamente da computação já realizada por ela. O computador de criação é o mesmo que está executando a criadora. O momento de criação das tarefas é função da quantidade de instruções já executada pela tarefa criadora. Uma tarefa cria uma nova sempre que executa a quantidade de instruções requerida para a sua próxima criação. As quantidades de instruções necessárias à criação das tarefas obedecem a uma distribuição exponencial de média constante e são limitadas a um máximo, que diminui com o aumento do número de tarefas já criadas e a criar. Sempre que este máximo é ultrapassado, a tarefa correspondente perde a capacidade de criar novas tarefas. O sistema é iniciado com uma tarefa criada espontaneamente no início da simulação. Assim se preserva a independência entre a carga computacional e o sistema computacional.

O custo de migração de uma tarefa refere-se à quantidade de palavras que devem ser transferidas para realizá-la e depende exclusivamente das características da tarefa e da computação já realizada por ela. Para tarefas que ainda não executaram seu

custo é mínimo. Para as demais o custo obedece a uma distribuição exponencial de média constante.

IV.6 Modelo de Funcionamento dos Sistemas MAD

Um sistema MAD conforme a definição no capítulo I, é uma rede escalável de computadores conectados por canais de comunicação bidirecionais. Os computadores e os canais de comunicação podem ser representados por um grafo, onde os nós referem-se aos computadores e os arcos aos canais de comunicação.

Cada computador de um MAD tem a sua própria identidade e é caracterizado pela velocidade de execução do seu processador de instruções. Esta velocidade é dada em palavras por segundo, onde uma palavra refere-se ao conteúdo de uma posição de memória, que pode representar uma instrução ou o valor de um dado.

Os canais de comunicação são caracterizados pelo seu tempo de ciclo, pela sua velocidade de transmissão e pela identidade dos computadores que conecta. O tempo de ciclo de um canal de comunicação representa o tempo necessário para que um dado enviado seja recebido. O tempo de ciclo é dado em segundos e a velocidade de transmissão em palavras por segundo.

Capítulo V

Sistema de Comunicação

O desempenho do mecanismo que realiza a comunicação entre tarefas em qualquer multicomputador é muito importante quando o grão de paralelismo é pequeno ou médio [43], como é o caso das APD.

Este capítulo aborda o sistema de comunicação desenvolvido para os sistemas MAD. Inicialmente (seções V.1 e V.2) são revistas as propriedades fundamentais requeridas e as medidas de desempenho dos sistemas de comunicação para multicomputadores. A seção V.3 apresenta a estrutura geral dos sistemas de comunicação para multicomputadores, a partir de uma análise dos principais problemas relacionados ao seu projeto, com ênfase para o problema da realização da comunicação entre tarefas migrantes. A subseção V.3.5 analisa as soluções anteriores para o problema da realização da comunicação entre tarefas migrantes. A seguir a seção V.4, apresenta as soluções adotadas neste trabalho, com ênfase especial para o novo protocolo proposto para a comunicação entre tarefas migrantes. Nas subseções seguintes são apresentadas a definição deste protocolo (V.4.1), a verificação da sua correção (V.4.2) e a verificação da sua qualidade com relação a garantia das propriedades definidas na seção V.1.

V.1 Propriedades Requeridas

Um sistema de comunicações para um sistema MAD deve implementar a troca de mensagens entre as tarefas que constituem uma APD, observando as seguintes propriedades:

1. **Transparência da localização das tarefas.** A localização das tarefas, ou seja, os computadores em que elas estão executando, deve ser totalmente transparente à aplicação.

2. **Integridade das mensagens.** A transmissão deve ser realizada sem alterações nas mensagens.
3. **Unicidade das mensagens.** A transmissão deve ser realizada sem duplicação de mensagens.
4. **Ordenamento parcial das mensagens.** Para uma transmissão entre um par qualquer de tarefas, as mensagens devem ser recebidas na ordem em que foram enviadas.
5. **Eficácia.** O atraso para a entrega de uma mensagem tem que ser finito, mesmo que não seja determinístico. Isto significa que todas as mensagens tem que ser entregues.

A maioria dos algoritmos distribuídos, a serem implementados pelas APD, supõem algumas ou todas estas propriedades, sejam elas proporcionadas pelo sistema a nível de hardware ou de software de controle.

V.2 Medidas de Desempenho

Quanto ao desempenho, os objetivos de um sistema de comunicação são minimizar a latência e maximizar a vazão de mensagens entre as unidades de execução.

Latência é o tempo médio necessário a entrega de uma mensagem pelo sistema de comunicação. É medida pelo intervalo de tempo entre o pedido de transmissão da mensagem, a partir do computador fonte, até a recepção da mensagem no computador destino.

A **vazão** é a taxa máxima de mensagens entre tarefas que pode ser atendida pelo sistema de comunicação. Se a taxa de mensagens submetida ao sistema de comunicação for superior a sua vazão, o sistema de comunicação satura-se e a latência aumenta indefinidamente.

V.3 Estrutura Geral

De acordo com a seção V.1 o sistema de comunicação de um sistema MAD proporciona à APD a abstração da troca de mensagens entre tarefas. No caso da comunicação entre duas tarefas localizadas em computadores distintos, a troca de mensagens entre as tarefas corresponde à transmissão de mensagens entre os computadores.

Considerando-se que os sistemas MAD normalmente não são completamente interconectados, dois computadores que se comunicam não estão necessariamente diretamente conectados. As mensagens entre os computadores de um sistema MAD têm então, em geral, que trafegar por vários canais de comunicação e eventualmente serem processadas por vários computadores intermediários até alcançar o seu destino final.

O projeto de um sistema de comunicação para um multicomputador depende das decisões quanto à solução de uma série de problemas relacionados hierarquicamente. Há desde problemas que envolvem características físicas e construtivas de um multicomputador até problemas lógicos. A definição da forma de funcionamento dos canais de comunicação e da topologia do multicomputador são exemplos de problemas do primeiro grupo. Este trabalho aborda os seguintes problemas lógicos:

- **Comutação dos canais de comunicação.** Estabelece a forma de transmissão de mensagens por um canal de comunicação. Implementa a comunicação entre computadores vizinhos.
- **Roteamento.** Método para a determinação das possíveis seqüências de canais de comunicação entre cada par de computadores. Implementa a comunicação entre um par qualquer de computadores, transformando a rede de conexão em uma rede de comunicação.
- **Controle de fluxo.** Método para o controle do tráfego na rede de comunicação. Implementa uma política de utilização dos canais de comunicação, gerenciando os conflitos para o acesso aos canais de comunicação e garantindo o progresso das mensagens.
- **Comunicação entre tarefas migrantes.** Protocolo para o roteamento de mensagens entre tarefas migrantes. Implementa a comunicação entre tarefas que podem migrar a qualquer momento.

As subseções seguintes abordam cada um destes problemas.

V.3.1 Comutação dos Canais de Comunicação

Um dos problemas fundamentais do projeto de uma rede qualquer de computadores é a escolha de um método de comutação dos canais de comunicação. A escolha do método implica na definição da unidade de informação que pode ser enviada e a forma de transmissão desta unidade. Há três métodos básicos:

- **Comutação de circuitos.** Inspirado no método de comutação da rede telefônica. A unidade de informação enviada é uma mensagem. A transmissão

de uma mensagem é realizada em três passos. Inicialmente é estabelecido um caminho completo que conecte o computador emissor ao computador destinatário. Este caminho pode ser composto por mais de um canal de comunicação e implica na alocação simultânea, exclusiva e estática de todos os recursos da rede de comunicação necessários à transmissão da mensagem. Após o estabelecimento deste caminho, a mensagem é enviada diretamente ao computador destinatário. A seguir, o caminho é desfeito, com a liberação dos recursos alocados para a transmissão.

- **Comutação de mensagens.** A unidade de informação enviada também é uma mensagem. Este método, também denominado *"datagram"*, tem por objetivo tornar desnecessária a alocação simultânea de mais de um canal de comunicação para a transmissão de uma mensagem. A mensagem sofre transmissões sucessivas a computadores cada vez mais próximos do destinatário, até alcançá-lo. Se não há um canal de comunicação que conecte diretamente o computador emissor ao destinatário, um canal de comunicação até um computador mais próximo do computador destinatário é alocado dinamicamente e a mensagem inteira trafega por este canal. Este processo se repete sucessivamente até que a mensagem alcance o seu destino. Este método aumenta o tempo disponível dos canais de comunicação, no entanto também aumenta o tempo mínimo de latência de uma mensagem, devido a três fatores: (1) a sua transmissão pelo próximo canal de comunicação só se inicia após ela ter sido inteiramente recebida pelo computador intermediário, (2) a quantidade de dados de uma mensagem tipicamente é muito maior que o necessário ao estabelecimento de um circuito e (3) muito maior que os dados em transmissão por um canal a um dado instante.

- **Comutação de pacotes.** A motivação deste método é a diminuição do tempo de latência de uma mensagem. As mensagens são divididas em pacotes. Os pacotes têm endereçamento próprio e são transmitidos independentemente e da mesma forma que as mensagens no método de comutação de mensagens. Apesar de terem a mesma origem e destino, os caminhos percorridos pelos pacotes de uma mesma mensagem podem ser diferentes. A latência diminui porque em cada computador intermediário só é necessária a espera de um pacote, ao invés de toda a mensagem e porque os pacotes podem utilizar caminhos paralelos. Como contrapartida, os pacotes ao serem recebidos precisam ser reordenados e são necessárias mais decisões de roteamento, uma por pacote em cada computador intermediário.

Os multicomputadores normalmente utilizam a comutação de mensagens. Estas no entanto são divididas em pacotes ou em partes menores, que trafegam seqüencialmente pelo mesmo caminho.

V.3.2 Roteamento

Um método de roteamento define o próximo canal de comunicação a ser percorrido por uma mensagem a fim alcançar o seu destino. Os métodos de roteamento podem ser classificados como determinísticos, oblívios, ou adaptativos, de acordo com o tipo de informações utilizadas para a realização da decisão de roteamento.

Em um método de roteamento **determinístico** o caminho percorrido por um pacote depende apenas dos computadores de origem e de destino da mensagem. A utilização de um método determinístico pode apresentar como vantagem a ordenação dos canais de comunicação utilizados, que é útil para se evitar *"deadlocks"*.

Um método **oblívio** pode escolher um dentre vários caminhos diferentes para a realização de uma comunicação. Esta escolha, no entanto não se baseia no estado de utilização dos canais de comunicação. O caminho escolhido independe de qualquer outro tráfego de comunicação.

Em um método de roteamento **adaptativo**, a escolha do caminho percorrido por um pacote ou mensagem é função das informações referentes ao estado corrente do sistema de comunicação. Este tipo de método apresenta duas vantagens:

- Se um caminho está sobrecarregado com o tráfego de mensagens, um outro caminho pode ser escolhido reduzindo a latência da mensagem.
- Se um caminho tem um canal defeituoso, outro caminho pode ser utilizado preservando a continuidade das comunicações.

V.3.3 Controle de Fluxo

A política de controle de fluxo resolve as disputas eventuais entre as mensagens em trânsito, quanto à utilização dos recursos do sistema de comunicação. O controle de fluxo é realizado por um método de gerenciamento dos recursos do sistema de comunicação, que aloca estes recursos aos dados a serem transmitidos, e tem por objetivo garantir o progresso das mensagens em direção ao seu destino. Assim, o método de controle de fluxo é o responsável pelo gerenciamento da comunicação entre os computadores de um multicomputador.

A política adotada para o controle de fluxo de um sistema de comunicação é fundamental para o seu desempenho geral, determinando a vazão e a latência de

um sistema de comunicação. Esta subseção aborda os principais métodos para o controle de fluxo de um sistema de comunicações.

Quando há uma disputa pela utilização de um mesmo canal de comunicação por mais de uma mensagem em trânsito ocorrem três questões:

1. Alguma das mensagens consegue utilizar o canal ?
2. Em caso positivo, qual das mensagens ?
3. O que acontece com as outras mensagens ?

Quanto as duas primeiras questões a solução é simples. A mensagem que primeiro solicitar um canal de comunicação disponível o obtém e o utiliza até a conclusão da sua transmissão.

Com relação ao que acontece com cada mensagem que não consegue utilizar o canal de comunicação que precisa, há quatro estratégias básicas:

- **Armazenamento.** A mensagem pode ser armazenada no computador de origem do canal de comunicação solicitado, enquanto aguarda pelo direito à sua utilização. A vantagem é o aproveitamento integral da capacidade dos canais de comunicação. Em compensação, esta estratégia requer uma grande capacidade de armazenamento e cuidados especiais no armazenamento para se evitar eventuais *"deadlocks"* decorrentes da limitação da capacidade de armazenamento.
- **Bloqueio.** Ao invés de ser armazenada, a mensagem tem a sua recepção suspensa até que ela consiga o direito de utilizar o canal de comunicação que precisa. Deste modo, o canal de comunicação de onde ela provém é bloqueado, havendo uma perda no aproveitamento da capacidade do canal. Há ainda a possibilidade de "deadlock" se mais de um canal está bloqueado simultaneamente. A vantagem é a limitação da capacidade de armazenamento necessária, possibilitando a construção de um controlador de comunicações mais simples e rápido.
- **Eliminação.** A mensagem pode ser simplesmente eliminada. Naturalmente o computador de origem da mensagem é informado do ocorrido. O resultado pode ser um grande desperdício da capacidade dos canais e a instabilidade do sistema. Outra desvantagem desta estratégia é a necessidade de confirmação do progresso da mensagem.
- **Roteamento errado.** A mensagem é roteada para um outro canal de comunicação que esteja disponível, assumindo-se que a mensagem retornará algum tempo depois, quando então novamente tentará utilizar o canal de comunicação originalmente solicitado, ou assumindo-se que a mensagem conseguirá

alcançar o seu destino através de um outro caminho alternativo. Esta estratégia desperdiça a capacidade dos canais ao afastar as mensagens do seu destino.

Estas estratégias estão relacionadas aos principais métodos utilizados em multicomputadores para o controle de fluxo. Há quatro métodos principais: *"roteamento desesperado"*, *"store and forward"*, *"virtual cut-through"* e *"wormhole"*.

O método conhecido por **"roteamento desesperado"** baseia-se na estratégia de roteamento errado. Este método pode resultar em *"livelocks"*, quando mensagens não conseguem progredir até o seu destino. Ele foi utilizado em alguns dos primeiros multicomputadores como o Denelcor HEP e a Connection Machine [16]. Este método não deve ser mais considerado para os multicomputadores atuais, já que não é eficiente nem eficaz, pois desperdiça a banda passante do sistema de comunicação e mesmo assim não garante o progresso das mensagens e portanto a entrega delas.

O método **"store and forward"** se baseia na estratégia de armazenamento e na comutação de pacotes ou de mensagens. Cada mensagem ou pacote é recebido integralmente e armazenado antes de ser transmitido pelo próximo canal. Este método pode apresentar problemas de *"deadlock"*.

O método denominado **"virtual cut-through"**, criado por Kermani e Kleinrock [36], tem por objetivo minimizar o tempo de latência das mensagens. Se o próximo canal a ser percorrido por uma mensagem estiver disponível a mensagem é enviada diretamente. Em caso contrário ela é recebida e armazenada, como no método *"store and forward"*.

Os métodos que utilizam o armazenamento das mensagens, como *"store and forward"* e *"virtual cut-through"*, apresentam como vantagem a possibilidade de aproveitamento total da banda passante dos canais de comunicação. Esta vantagem tem como custo a necessidade de armazenamento das mensagens que não podem prosseguir. Como a capacidade de armazenamento não é ilimitada é possível a ocorrência de *"deadlocks"* quando uma mensagem não puder prosseguir, o que liberaria os buffers onde está armazenada, porque não há buffers disponíveis no próximo computador, que também não são liberados pela mesma razão. Ocorrerá uma situação de "deadlock" se computadores nesta situação formarem pelo menos um círculo de dependência.

A ocorrência de "deadlock" nestes métodos pode ser evitada através da estruturação dos buffers de armazeamento em níveis. Nestas soluções, no entanto, a demanda por buffers tende a crescer com o número de computadores interligados, o que não é desejável para um multicomputador escalável.

O método **"wormhole"**, apresentado por Seitz et al [63], está relacionado à estratégia de bloqueamento. Uma mensagem é decomposta em **flits** (*"flow control digits"*), que é a menor unidade de dados reconhecida pelo mecanismo de controle de fluxo. Normalmente, uma mensagem proveniente de um canal de entrada é transmitida a medida que é recebida, flit a flit, para o próximo computador, através de um canal de saída, e assim sucessivamente até o computador destinatário. Os flits de uma mensagem podem assim estar distribuídos por vários computadores ao longo da trajetória percorrida pela mensagem.

Os canais físicos de comunicação são compartilhados por canais virtuais de comunicação. Enquanto um canal virtual está bloqueado, um outro canal virtual, que compartilha do mesmo canal físico, pode estar ativo utilizando o canal físico comum. Assim, é possível evitar a perda da banda passante dos canais de comunicação. Um canal virtual mantém-se alocado a transmissão de uma mensagem até que todos os flits da mensagem tenham fluído por ele.

A implementação dos canais virtuais é bem simples. A um canal físico é acrescido um circuito de comutação e para cada canal virtual implementado, é acrescido um buffer capaz de armazenar apenas o próximo flit a ser enviado, ou o flit recebido mais recentemente. Assim a demanda por capacidade de armazenamento em cada computador é proporcional ao seu número de canais de comunicação, e não ao número de computadores em todo o sistema.

A ocorrência de *"deadlocks"* na utilização do método de *"wormhole"* pode ser prevenida. Basta restringir as possíveis combinações entre os canais virtuais de entrada e de saída, de modo a estabelecer uma ordem entre os canais virtuais e assim garantir a inexistência de ciclos com os canais virtuais. Esta solução também restringe os caminhos alternativos entre os computadores, o que diminui a adaptabilidade do sistema de comunicação. Há alternativas para a implementação de *"wormhole"* adaptativo [40].

Como descrito na seção V.2 o desempenho de um sistema de comunicação é dado pela sua latência e pela sua vazão. O método de controle de fluxo determina estas características quanto a comunicação entre computadores. Assim, a escolha do método de controle de fluxo a adotar deve ser baseada na análise destas medidas relativas a estes métodos.

Uma análise completa da latência de um método de controle de fluxo é muito complexa pois envolve muitas variáveis e pouco precisa pois os valores atribuídos a estas variáveis podem ser mal avaliados ou pouco representativos. Este trabalho apresenta uma análise mais simples, mas que já é bastante ilustrativa.

A análise da latência destes métodos pode ser realizada a partir da definição de alguns parâmetros independentes. Seja L o número de bits de uma mensagem; T_C o tempo de um ciclo do canal de comunicação, ou seja, o tempo necessário a transmissão de um flit; W o número de bits de um flit; e N o número de canais que uma mensagem deve percorrer. Como simplificação, assuma-se que os canais estão sempre disponíveis.

No método *"store and forward"* uma mensagem deve ser recebida integralmente antes de ser enviada pelo próximo canal. Assim para este método, a latência é dada pelo produto do tempo necessário para que a mensagem percorra integralmente um canal, pelo número de canais a serem percorridos, ou seja:

$$T_{SF} = \left(\frac{L}{W} \times T_C\right) \times N \quad ou \quad T_{SF} = T_C \times \left(\frac{L}{W} \times N\right)$$

Assumindo-se que os canais permanecem disponíveis, a latência do método *"virtual cut-through"* é igual a do método *"wormhole"*. Nestes métodos uma mensagem é enviada integralmente, sob a forma de uma seqüência de flits e cada flit recebido da mensagem já pode ser enviado pelo próximo canal, sem aguardar pela recepção dos posteriores. Para estes métodos, que funcionam como um *"pipeline"*, a latência é dada pela soma do tempo necessário a que um flit percorra todos os canais e alcance o seu destino, mais o tempo necessário a que todos os flits iniciem sua transmissão, ou seja:

$$T_{WH} = N \times T_C + \frac{L}{W} \times T_C \quad ou \quad T_{WH} = T_C \times \left(\frac{L}{W} + N\right)$$

Comparando-se as expressões constata-se que os métodos *"wormhole"* e *"virtual cut-trough"* são muito superiores ao método *"store and forward"* quanto ao tempo de latência. Quanto a vazão, os métodos se equivalem, pois em todos eles é possível o total aproveitamento da banda passante dos canais de comunicação.

V.3.4 Comunicação entre Tarefas Migrantes

O objetivo primordial de um sistema de comunicação é propiciar a comunicação entre as diversas entidades do sistema. Nos multicomputadores onde ocorre a criação dinâmica de tarefas, é fundamental a comunicação entre as tarefas e estas podem migrar. Assim é necessária a obediência a um protocolo que garanta a troca de mensagens, não apenas entre computadores distintos, mas entre tarefas migrantes.

Os métodos para controle de fluxo têm por objetivo realizar a comunicação entre cada par de computadores de um multicomputador. A capacidade de migração das tarefas distingue o problema da comunicação entre computadores de um multicomputador, do problema da comunicação entre tarefas migrantes, em razão das seguintes diferenças:

- *Comunicação entre computadores de um sistema* MAD. Os computadores e os canais de comunicação de um sistema MAD são definidos estaticamente. Assim os caminhos entre cada par de computadores permanecem inalterados durante toda a execução da APD.
- *Comunicação entre as tarefas de uma* APD. As tarefas são criadas dinamicamente de acordo com a evolução da execução da APD e podem migrar a qualquer instante. Conseqüentemente, os caminhos para comunicação entre tarefas se alteram durante a execução da APD e, portanto, precisam ser redefinidos dinamicamente.

Um dos muitos problemas a ser resolvido é como garantir as propriedades definidas na seção V.1, quando as tarefas comunicantes estão em migração. Neste caso, as mensagens são endereçadas a alvos em movimento. Um sistema de comunicação deve prover um esquema de migração que realize o roteamento das mensagens com a recuperação e o rerroteamento das mensagens perdidas.

Uma solução óbvia para o problema é fazer com que a localização atual de todas as tarefas em execução seja conhecida por cada computador do sistema MAD. Isto exige que cada criação, migração e término de todas as tarefas seja comunicada por difusão para todo o sistema, de modo a que este conhecimento se mantenha correto em todos os computadores. A difusão de mensagens em redes ponto a ponto de computadores como os sistemas MAD não é trivial, nem eficiente [49]. Como o custo de difusão de mensagens em um sistema MAD pode ser muito alto, esta solução é inadequada por não ser escalável.

Este trabalho propõe na subseção V.4.1 um novo protocolo para a comunicação entre tarefas migrantes em um multicomputador.

V.3.5 Trabalhos Anteriores

Esta subseção apresenta os principais trabalhos anteriores quanto a protocolos para a comunicação entre tarefas migrantes.

Um protocolo para a migração de processos e o encaminhamento de mensagens foi implementado pela primeira vez em um sistema distribuído no sistema operacional DEMOS/MP [53]. Esta implementação não previa a execução concorrente de

várias instâncias do protocolo no mesmo computador. Isto significa que um computador somente poderia realizar uma migração por vez. Esta característica é uma desvantagem, porque um protocolo ideal não deve impor restrições à realização de migrações.

Fowler [28][29] propôs o protocolo *"forwarding address"* para localizar objetos que se moviam em um sistema distribuído. Este protocolo assume que cada computador tem uma tabela completa com um endereço recente ou não, de cada tarefa no sistema. Nos protocolos do tipo *"forwarding address"* as mensagens são enviadas em direção ao provável endereço da tarefa destinatária. Se quando a mensagem chegar, a tarefa já houver migrado, a mensagem é enviada a um endereço mais atual, e assim sucessivamente até alcançar a tarefa destinatária. Os endereços das tarefas nos computadores são atualizados passivamente a medida que a mensagem trafega. Este protocolo não prevê uma atualização ativa dos endereços das tarefas que migram.

A principal vantagem da abordagem de Fowler é que as entradas de roteamento não são atualizadas nos computadores não relacionados às trajetórias percorridas pelas mensagens. Uma desvantagem é que não há um mecanismo que garanta a atualização dos endereços das tarefas que migraram. Estes endereços podem ficar muito desatualizados, prolongando os caminhos percorridos pelas mensagens. Outra desvantagem, é a necessidade do conhecimento prévio do endereço inicial de todas as tarefas por todos os computadores.

No sistema Emerald [35] foram implementadas tanto a mobilidade de tarefas como de dados. Para possibilitar a localização dos objetos pelos computadores, o sistema incorpora o protoloco *"forwarding address"* de Fowler. Quando não se conhece o endereço de um objeto, o sistema Emerald difunde uma mensagem por todo o sistema solicitando o endereço do objeto. A principal desvantagem deste esquema é a necessidade da difusão de mensagens por todo sistema, o que o torna não eficiente.

Nos sistemas operacionais Locus [52] e Sprite [18][51] as tarefas interagem através de chamadas ao sistema. A cada tarefa é atribuído um computador especial, para o qual são enviadas todas as chamadas remotas relativas à tarefa. Este computador é encarregado de enviar as chamadas remotas para as tarefas a partir de uma estimativa para a localização atual da tarefa. Esta estratégia apresenta vários problemas:

- Há uma dependência da tarefa em relação ao seu computador especial, mesmo após ter migrado.
- Em sistemas distribuídos com um grande número de computadores o com-

primento adicional do caminho a ser percorrido pelas mensagens, necessário à passagem pelo seu computador especial, pode ser muito maior do que a distância efetiva entre as tarefas envolvidas na chamada remota.

- A passagem obrigatória por um computador especial, estaticamente associado a uma tarefa, de todas as mensagens endereçadas a esta tarefa, torna a migração de tarefas ineficaz para reduzir as distâncias de comunicação.

O protocolo de roteamento de mensagens do sistema V [70] utiliza um mecanismo de cache para a localização de tarefas e o protocolo de *"forwarding address"* para o envio das mensagens. Se a localização correta de uma tarefa não está disponível no cache de um computador, uma mensagem é difundida por todo o sistema requerendo a localização atual da tarefa. A grande desvantagem deste esquema é a realização de um número indeterminável de difusões de mensagens por todo o sistema.

O mecanismo de migração de tarefas proposto por Lu, Chen e Liu [42] assume que cada tarefa se comunica apenas com um número limitado de outras tarefas, conhecidas a priori, denominadas tarefas adjacentes. Cada computador tem uma tabela com a localização das tarefas adjacentes às tarefas localizadas no computador. A migração de uma tarefa é precedida pelo bloqueio da transmissão de mensagens endereçadas a esta tarefa; após a migração, o novo endereço da tarefa é difundido para os computadores com tarefas adjacentes e as transmissões para a tarefa são desbloqueadas. As desvantagens deste esquema são a impossibilidade do envio de mensagens para as tarefas em migração, e a restrição ao grupo de tarefas com que uma tarefa pode se comunicar.

Ravi [55] apresenta uma especificação formal de um protocolo para a comunicação e migração de tarefas, em sistemas distribuídos ponto a ponto. A correção do protocolo também é apresentada. Este protocolo, no entanto, não garante uma das propriedades definidas na seção V.1. O protocolo não garante que a ordem de recepção das mensagens entre duas tarefas será a mesma ocorrida na transmissão.

Não há assim uma solução perfeita para o problema. Na seção seguinte é proposta mais uma solução, que não impõe restrições a comunicação entre as tarefas, e que não requer a difusão de mensagens por todo o sistema, o que é especialmente importante para um melhor desempenho em sistemas com muitos computadores.

V.4 Sistema de Comunicação Proposto

Esta seção explica o funcionamento do sistema de comunicação de um sistema MAD. São descritos e explicados as estruturas de dados utilizadas e os algoritmos execu-

tados para o roteamento das mensagens pelo sistema.

Um objetivo fundamental do sistema de comunicação de um sistema MAD é propiciar a comunicação entre tarefas de forma eficaz e eficiente. Como uma tarefa pode migrar a qualquer momento, inclusive quando há mensagens já transmitidas em sua direção, e como o conhecimento da ocorrência de uma migração não é imediato em todos os computadores de um sistema distribuído, então o roteamento de mensagens entre tarefas tem que se basear em informações sobre a localização das tarefas que podem estar desatualizadas.

Para ser eficiente, um protocolo para a comunicação entre tarefas migrantes deve ser capaz de redefinir a trajetória de uma mensagem ao longo da sua transmissão, sempre que se conheça uma trajetória melhor que a prevista. O método de controle de fluxo a ser adotado deve permitir a redefinição da trajetória de uma mensagem antes que ela alcance o computador ao qual ela inicialmente se destinava.

A livre redefinição da trajetória de uma mensagem em trânsito não é possível no método de controle de fluxo *"wormhole"*. A não ocorrência de "deadlocks" neste método depende do ordenamento dos canais da trajetória completa da mensagem. A trajetória estabelecida não pode ser alterada sob o risco de desobedecer a este ordenamento. Assim, este método para a comunicação entre computadores não é adequado para a realização da comunicação entre tarefas migrantes.

O método de controle de fluxo a ser adotado é o *"virtual cut-through"*. Este método independe da trajetória efetivamente percorrida pela mensagem e apresenta as melhores características de desempenho, que são equivalentes as do método *"wormhole"*. O método de comutação dos canais de comunicação é o *"message switch"*, que é adequado a comunicação intermitente que se espera das tarefas de uma APD e ao método *"virtual cut-through"*. O método de roteamento a ser utilizado decorre da topologia do sistema MAD utilizado, mas é adaptativo.

Toda comunicação entre tarefas é realizada por processadores de comunicação (PC). Quando uma tarefa decide enviar uma mensagem a outra tarefa, a tarefa emissora requer este serviço ao PC do computador no qual ela está localizada. A transmissão de uma mensagem entre tarefas pode utilizar canais de comunicação e envolver mais de um PC.

V.4.1 Comunicação entre Tarefas Migrantes

Esta subseção apresenta o protocolo desenvolvido para realizar a comunicação entre tarefas migrantes em um sistema MAD.

As mensagens podem ser classificadas em mensagens do escalonamento global, mensagens do mecanismo de roteamento e mensagens da aplicação. A primeira palavra de uma mensagem identifica a sua classe. Esta seção aborda as mensagens do mecanismo de roteamento e as mensagens da aplicação.

As mensagens entre os PC são:

- INTER: transmite uma mensagem entre tarefas.
- LOCAL: transmite a atual localização de uma tarefa.

Estas mensagens contêm um bloco de controle incluindo a identificação, localização e grau de atualidade da localização, tanto da tarefa emissora e como da destinatária. O grau de atualidade de uma localização é igual a um mais o número de migrações já ocorridas com a tarefa, quando ela apresentava esta localização.

O mecanismo de roteamento de mensagens entre tarefas utiliza o conhecimento aproximado da localização da tarefa destinatária. O PC da tarefa emissora envia a mensagem em direção ao computador que segundo o seu conhecimento é a localização mais recente da tarefa destinatária.

A primeira aproximação para a localização de uma tarefa é definida estaticamente, de acordo com a identificação da tarefa. Há um conjunto de computadores, espalhados uniformemente pelo sistema MAD, associado a cada tarefa. A escolha dos membros deste conjunto depende da topologia particular do sistema MAD disponível. Esta escolha, no entanto, é definida de tal forma que a partir da identidade de uma tarefa seja possível, a qualquer PC, determinar o computador mais próximo associado a tarefa, que será conhecido como a localização da tarefa com grau de atualidade igual a zero.

Quando uma tarefa é criada o conjunto de computadores associado a ela é informado da sua localização através de uma mensagem LOCAL. Os PC armazenam as mensagens endereçadas às tarefas associadas ao seu computador até a recepção de uma mensagem LOCAL, informando a criação e a localização de uma destas tarefas, quando então, as mensagens relativas a tarefa cuja localização foi comunicada são enviadas.

A atualização do conhecimento da localização de uma tarefa só pode ocorrer de duas formas:

- quando um computador for o destinatário, ou pertencer ao caminho percorrido por uma mensagem que contenha uma localização mais atual da tarefa que a disponível neste computador e
- quando um computador participa da migração de uma tarefa, seja como transmissor ou como receptor.

Uma das informações de estado da tarefa é o número de migrações em que já participou. Quando ocorre a migração de uma tarefa, este número é incrementado. Um computador ao receber uma tarefa passa a conhecer o grau de atualidade desta localização da tarefa a partir deste número de migrações. O grau de atualidade é igual ao número de migrações da tarefa, mais um.

As mensagens são inicialmente enviadas em direção a localização conhecida da tarefa destinatária pelo PC da tarefa emissora. Se ao longo da trajetória de uma mensagem um dos PC intermediários conhece uma localização mais atual da tarefa destinatária, então a localização e o seu grau de atualidade presentes na mensagem são corrigidos, a mensagem é redirecionada e é indicado na mensagem que o PC da tarefa emissora não conhece a localização mais recente da tarefa destinatária.

Quando uma mensagem é recebida pelo PC do computador onde a tarefa destinatária se encontra ela é colocada na fila de mensagens da tarefa (veja a frente). Se há a indicação de que o PC da tarefa emissora não conhece a localização atual da tarefa destinatária então, é enviada uma mensagem LOCAL direcionada a tarefa emissora.

```
Primeira execução de uma tarefa:

Envia uma mensagem de atualização da localização para
os computadores definidos como a primeira aproximação
da sua localização.
```

Figura V.1: Primeira Execução de uma Tarefa

```
Pedido de transmissão de uma mensagem:

Se a tarefa destinatária estiver presente
    Coloca a mensagem na fila da tarefa destinatária.
    Fim.
Envia a mensagem em direção ao computador onde
a tarefa destinatária deve se localizar.
```

Figura V.2: Pedido de Transmissão de Mensagens

O PC participa do processo de roteamento de uma mensagem quando ocorre um dos seguintes eventos:

- Primeira execução de uma tarefa.

- Recepção de uma mensagem de atualização da localização de uma tarefa.
- Pedido de transmissão de mensagens por parte de uma tarefa em seu computador.
- Recepção de uma mensagem entre tarefas proveniente de outro PC

As figuras V.1, V.2, V.3 e V.4 explicitam as ações realizadas em cada caso.

```
Recepção de uma mensagem de atualização de localização:

Atualiza a localização da(s) tarefa(s) especificadas.
    Se há mensagens acumuladas endereçadas à(s) tarefa(s)
    cuja localização foi atualizada.
        Envia as mensagens em direção ao computador onde
        a(s) tarefa(s) deve(m) estar.
Se a mensagem for endereçada a outro computador.
    Envia a mensagem em direção ao outro computador.
```

Figura V.3: Recepção de Mensagens de Localização

Cada PC gerencia as seguintes informações:

- **Tabela das tarefas**: indica a localização das tarefas. Para cada tarefa há duas informações: endereço mais recente e atualidade deste endereço.
- **Tarefas presentes**: lista das tarefas em um computador. Para cada tarefa presente estão associadas as seguintes informações: *fila de mensagens não entregues* e as *listas de correspondentes*.

A ordem parcial das mensagens trocadas entre as tarefas é garantida por duas estruturas de dados por tarefa, *as listas de correspondentes* e a *fila das mensagens não entregues*. Há uma **lista de correspondentes** com as tarefas que receberam mensagens desta tarefa e outra com as tarefas que enviaram mensagens a esta tarefa. Cada elemento destas listas contém a identificação de uma tarefa que já se correspondeu com esta e o número de mensagens já transmitidas ou recebidas. A **fila de mensagens** contém as mensagens que ainda não foram entregues às tarefas destinatárias, seja porque as tarefas ainda não as solicitaram, ou porque mensagens anteriores ainda não chegaram.

Todas as mensagens têm um número de ordem, que é incluído pelos PC, a partir da lista de receptores da tarefa emissora. As mensagens são ordenadas por este número e permanecem na fila até que as mensagens anteriores sejam entregues. A lista de emissoras de uma tarefa têm o número de ordem da última mensagem recebida pela tarefa, proveniente de cada tarefa que já lhe enviou mensagens.

```
Recepção de uma mensagem entre tarefas:

Se a mensagem contém uma localização da tarefa emissora
ou destinatária mais atual que este processador de
comunicações.

    Atualiza a localização da(s) tarefa(s) no processador
    de comunicações.

        Se há mensagens acumuladas endereçadas à(s) tarefa(s)
        cuja localização foi atualizada.

            Envia as mensagens em direção ao computador onde
            a(s) tarefa(s) deve(m) estar.

Se a mensagem contém uma localização da tarefa destinatária
menos atual que este processador de comunicações.

    Atualiza a localização da tarefa na mensagem.

    Indica na mensagem que o processador de comunicações
    do computador de origem da mensagem não conhece
    a localização atual da tarefa destinatária.

Se a tarefa destinatária estiver presente

    Coloca a mensagem na fila da tarefa destinatária.

    Se a mensagem não estava destinada a este computador
    ou está indicado que o processador de comunicações
    do computador da tarefa emissora não conhece a
    localização atual da tarefa destinatária.

        Envia uma mensagem de atualização da
        localização da tarefa.

    Fim.

Envia a mensagem em direção ao computador onde
a tarefa destinatária deve se localizar.
```

Figura V.4: Recepção de Mensagens entre Tarefas

A migração de uma tarefa envolve a migração do seu estado de execução, incluindo o número de migrações que já sofreu, a sua fila de mensagens e as listas de correspondentes.

V.4.2 Verificação da Correção do Protocolo

O protocolo proposto é correto se não produz *"deadlocks"* e se as mensagens que são roteadas de acordo com o protocolo são em algum momento entregues as tarefas destinatárias.

A ausência de *"deadlocks"* decorre da inexistência de bloqueios no protocolo em si e da ausência de "deadlocks" no método de controle de fluxo utilizado.

A prova de que as mensagens roteadas segundo o protocolo proposto alcançam as tarefas destinatárias se baseia nas seguintes premissas definidas no próprio protocolo:

1. Para cada tarefa há um conjunto de computadores que conhecem ou conhecerão uma localização desta tarefa. Todos os computadores reconhecem um dos membros deste conjunto como a localização da tarefa com o grau de atualidade igual a zero.
2. A criação de uma tarefa faz com que a localização e o seu grau de atualidade no computador da criação sejam alterados para respectivamente a identidade deste computador e um.
3. A criação de qualquer tarefa envolve o envio de uma mensagem com a sua localização de grau de atualidade um para todos os computadores reconhecidos como a localização de grau de atualidade zero da tarefa criada.
4. O PC de um computador armazena as mensagens recebidas, que forem dirigidas a tarefas cuja localização prevista na mensagem tenha um grau de atualidade igual a zero e que seja o próprio computador. Quando este PC passar a conhecer uma localização de uma destas tarefas com o grau de atualidade superior a zero as mensagens armazenadas endereçadas a esta tarefa continuaram o seu trajeto.
5. A migração de uma tarefa faz com que a localização e o seu grau de atualidade nos dois computadores envolvidos sejam atualizados respectivamente para o computador receptor e para um mais o número de migrações (presente no estado da tarefa).
6. As informações em um computador sobre a localização e o seu grau de atualidade de uma tarefa só são alterados pela recepção ou passagem de uma mensagem com uma localização da tarefa com um grau maior de atualidade, ou pela recepção da própria tarefa.
7. A localização de uma tarefa e o seu grau de atualidade presentes em uma mensagem sempre se originam de informações idênticas do computador de origem da mensagem ou de algum computador pelo qual a mensagem passou, que apresentava uma localização com um grau de atualidade maior.
8. As mensagens sempre são enviadas em direção a localização mais atual da sua tarefa destinatária.

As premissas 1, 2, 3 e 4 têm como conseqüência o seguinte corolário:

- *Qualquer mensagem enviada para um computador que é a localização com grau de atualidade zero da sua tarefa destinatária, será enviada para o computador onde esta tarefa destinatária foi criada.*

Para mostrar a correção do protocolo proposto é necessário provar o seguinte teorema:

- *Mesmo que a informação em um computador sobre a localização de uma tarefa não esteja necessariamente atualizada, ela sempre indica uma localização correta da tarefa no passado ou no presente.*

  A premissa 1 garante a correção da primeira informação quanto a localização de tarefas em todos os computadores. Resta verificar se estas informações podem se tornar incorretas após alguma atualização.

  Segundo a premissa 6 as informações em um computador sobre a localização de uma tarefa são atualizadas pela recepção de uma mensagem com uma informação mais atual ou pela participação em uma migração. Elas estarão erradas se em algum destes casos o computador receber uma informação errada quanto a localização de uma tarefa.

  De acordo com a premissa 5 a participação de um computador na migração de uma tarefa garante a este computador uma informação correta sobre a localização da tarefa. Assim esta descartada a possibilidade de um computador receber uma informação errada, quanto a localização de uma tarefa, através da participação em uma migração.

  De acordo com a premissa 7, as informações sobre a localização de uma tarefa em uma mensagem se originam de um dos computadores, portanto as mensagens não introduzem informações erradas no sistema e se todas as informações anteriores estavam corretas, as mensagens não as tornarão erradas. Assim esta descartada a possibilidade de um computador receber uma informação errada, quanto a localização de uma tarefa, através de uma mensagem.

A partir do corolário, deste teorema e das premissas definidas no protocolo proposto pode-se provar a correção do protocolo proposto.

O corolário e a premissa 5 garantem que a trajetória mais longa percorrida por uma mensagem será constituída pelas seguintes trajetórias:

1. do computador de origem da mensagem até um computador que conhece a localização da tarefa com grau zero,
2. deste computador até o computador de criação da tarefa destinatária,
3. a trajetória percorrida pela tarefa em suas migrações.

Como de acordo com o teorema todas as informações quanto a localização e o seu grau de atualidade de uma tarefa estão sempre corretas e como pelas premissas 7 e 8 a continuação da trajetória de uma mensagem é sempre em direção ao computador

conhecido pelos computadores por onde a mensagem passou como a localização mais atual da tarefa naquele momento, então qualquer outra trajetória percorrida pela mensagem será mais curta.

No pior caso então, a trajetória percorrida por uma mensagem é maior que a trajetória da sua tarefa destinatária por um número constante de computadores intermediários (trajetória 1 e 2 acima). Supondo-se que uma mensagem trafega com uma velocidade maior do que a velocidade de migração de uma tarefa, então todas as mensagens alcancaçam suas tarefas destinatárias.

Esta suposição é razoável pois normalmente as mensagens entre tarefas são muito menores que o somatório de mensagens para realizar uma migração. Qualquer que seja o protocolo para a comunicação entre tarefas migrantes, se as migrações tiverem uma maior velocidade que as transferências de mensagens, então com o passar do tempo, as mensagens poderão estar cada vez mais distantes das suas tarefas destinatárias. Assim, sem a suposição acima não se pode garantir que as mensagens alcancaçaram as suas tarefas destinatárias qualquer que seja o protocolo.

V.4.3 Qualidade do Protocolo Proposto

A verificação da qualidade do protocolo proposto pode ser realizada verificando-se se o protocolo proposto garante as propriedades definidas na seção V.1.

A propriedade 1, relativa a transparência do processo de migração para as tarefas, é garantida pela utilização de um identificador distinto para cada tarefa, que independe da localização da tarefa, e cujo reconhecimento em todo o sistema é realizado totalmente pelos PC.

As propriedades 2, relativa a integridade das mensagens, e 3, relativa a unicidade das mensagens, são garantidas pela concepção do protocolo. As mensagens não são nem particionadas, nem duplicadas.

A propriedade 4, relativa ao ordenamento parcial das mensagens, é garantida pela gerência da fila de mensagens recebidas e das listas de tarefas correspondentes de cada tarefa, que é realizada pelo PC do computador onde a tarefa se encontra.

A propriedade 5, relativa a eficácia, é decorrência da correção do protocolo, que foi provada na subseção V.4.2.

Capítulo VI

O Simulador DDSS

A simulação foi escolhida, no capítulo IV, como o método de avaliação a ser empregado neste trabalho. Este capítulo aborda o processo de simulação adotado, os componentes do simulador, a sua organização e as medidas de desempenho coletadas.

O simulador de escalonadores dinâmicos distribuídos DDSS tem por finalidade cumprir o objetivo da avaliação definido no capítulo IV. O DDSS é um simulador híbrido, pois além de implementar modelos de simulação, permite o acoplamento de módulos funcionais. O DDSS é um simulador discreto orientado a eventos, conforme o capítulo IV, programado na linguagem de programação C.

O DDSS mimetiza o funcionamento completo de um sistema MAD incluindo o sistema computacional, o sistema operacional e a APD em execução. O sistema computacional é mimetizado pelo emulador de sistemas MAD. O sistema operacional constitui-se dos módulos de comunicação, de escalonamento local e de escalonamento global. A APD é definida pelas atividades a serem realizadas pelas suas tarefas. Estas atividades são lidas de um arquivo de entrada do DDSS, o *arquivo de atividades*. A preparação para a avaliação de um escalonador global é realizada através da simples substituição do módulo de escalonamento global.

VI.1 Processo de Simulação

A simulação de qualquer sistema demanda a existência de uma caracterização deste, de acordo com os objetivos da simulação. Neste trabalho esta caracterização é realizada pelo modelo de execução das APD e pelo modelo de funcionamento dos sistemas MAD (ambos abordados no capítulo IV).

Os sistemas reais são normalmente muito complexos para que seja viável a sua

caracterização e simulação precisa. A viabilidade da simulação de um sistema exige que a sua caracterização se faça através da simplificação e sistematização da sua estrutura e do seu funcionamento. Como conseqüência dessa exigência, os resultados obtidos por um processo de simulação são uma estimativa dos resultados reais. A qualidade desta estimativa depende da qualidade da caracterização do sistema e do processo de simulação, em relação aos índices de desempenho selecionados para estudo.

Se a caracterização de um sistema utiliza um modelo estocástico, os resultados da sua simulação são probabilísticos e portanto requerem um tratamento estatístico. O controle e diminuição da margem de erro das estimativas de resultados obtidos por simulação, neste caso, requerem um método de inferência estatística.

Este trabalho utiliza o método de replicações independentes para o cálculo do intervalo de confiança e do seu nível de confiança. Neste método são realizados vários exercícios de simulação, denominados replicações, todos com o mesmo estado inicial, mas cada um de acordo com a sua seqüência de números aleatórios. Cada replicação refere-se a uma instância de uma carga computacional. Assim, cada exercício de simulação corresponde a uma execução de uma APD em particular.

A aplicação do método de replicações independentes a este trabalho, requer a realização de muitos exercícios de simulação. A avaliação de um escalonador em um sistema MAD particular requer um exercício de simulação para cada instância de cada carga computacional. Há um único conjunto de sistema MAD que é utilizado na avaliação de todos os escalonadores selecionados. O número total de exercícios de simulação é então igual ao número de escalonadores, multiplicado pelo número de cargas computacionais e pelo número de instâncias de uma carga computacional.

O custo total da simulação pode ser diminuído se for possível a reutilização das instâncias das cargas computacionais. O número total de instâncias das cargas computacionais pode ser igual ao número de replicações necessárias à avaliação de um escalonador em um sistema MAD. Um único conjunto de instâncias de cargas computacionais pode então ser utilizado na avaliação de todos os escalonadores selecionados em todos os sistemas MAD selecionados.

A reutilização das instâncias das cargas computacionais é possível se a sua geração for independente da realização dos exercícios de simulação. Neste trabalho, esta independência é obtida através do uso do **gerador de atividades**, que produz a cada ativação, uma nova instância da carga computacional caracterizada. Assim não é necessário que uma instância de carga computacional seja gerada a cada exercício de simulação.

A independência entre a geração das instâncias de cargas computacionais e a realização dos exercícios de simulação apresenta ainda como vantagens:

- O conhecimento direto de estatísticas importantes sobre a instância de carga computacional, como o tempo mínimo de execução (cadeia mais longa de tarefas), o grau máximo de paralelismo (número máximo de tarefas paralelas), a carga total de comunicação entre as tarefas e a carga total de processamento das tarefas.
- Uma maior facilidade para a garantia da submissão de todos os escalonadores as mesmas condições de operação.
- Uma menor complexidade da construção do simulador.

O processo de simulação é realizado por dois programas, o gerador de atividades e o simulador que as executa. O simulador resulta da incorporação do emulador de sistemas MAD, do mecanismo de comunicação, do mecanismo de escalonamento local e do escalonador global em avaliação. A interação do gerador de atividades com o simulador é realizada através do arquivo de atividades.

VI.2 Gerador de Atividades

O Gerador de Atividades implementa o modelo de execução das APD proposto na seção IV.5. Os valores atribuídos aos parâmetros do modelo determinam as características da carga computacional, da qual uma instância é gerada. O gerador de atividades gera as atividades das tarefas que compõem uma APD e as armazena em um *arquivo de atividades*.

As atividades que podem ser realizadas por uma tarefa são:

- **Processamento**: execução de uma seqüência de instruções.
- **Criação de Tarefas**.
- **Transmissão**: Transmissão de uma mensagem para uma outra tarefa.
- **Recepção**: Recepção de uma mensagem proveniente de outra tarefa.

O processo de geração de uma APD ocorre através da emulação da sua execução em um sistema MAD ideal. As atividades são geradas a medida que a APD vai evoluindo. O processo se inicia com a criação espontânea de uma tarefa. As atividades desta tarefa inicial são geradas até que esta tarefa termine. Ao longo da sua existência, a tarefa inicial provavelmente cria outras tarefas, que criam mais outras, e assim sucessivamente.

A evolução da APD se dá em um ambiente ideal, de modo que todo o paralelismo intrínseco à aplicação se desenvolva. Assim há a aproximação de um número ilimitado de computadores disponíveis, um para cada tarefa ativa, as trocas de mensagem entre tarefas são imediatas e o gasto computacional se restringe à realização das atividades da própria aplicação.

Os parâmetros que definem uma carga computacional são:

- MEDINST: Número médio de instruções de uma atividade de processamento.
- MININST: Número mínimo de instruções de uma atividade de processamento.
- MEDCHILD: Número médio de instruções executadas necessárias a criação de uma tarefa "filha".
- MIGCOST: Número médio de palavras a serem transferidas em uma migração de uma tarefa.
- MEDMESG: Número médio de palavras de uma mensagem entre tarefas.
- MINMESG: Número mínimo de palavras de uma mensagem entre tarefas.
- PRIV: Privilégio de recepção das tarefas do mesmo grupo da tarefa emissora.
- NEWGROUP: Probabilidade de criação de um novo grupo de tarefas.
- MAXTASKS: Número máximo de tarefas.

A seqüência de valores pseudoaleatórios utilizada pelo gerador de atividades é produzida através do método congruente e é obtida por chamadas sucessivas de uma rotina da linguagem de programação utilizada. As amostras aleatórias das distribuições de probabilidade, previstas pelo modelo de execução das APD, são geradas pelo método da transformada inversa sobre os valores pseudoaleatórios gerados.

O gerador de atividades produz as seguintes estatísticas da APD gerada (algumas opcionais):

- Seqüência máxima de atividades de processamento, que determina o tempo mínimo de execução da APD em um sistema MAD.
- Número máximo de tarefas ativas simultaneamente.
- Número total de instruções da APD.
- Número total de atividades de processamento.
- Número de atividades de processamento por tarefa.
- Número total de palavras transmitidas.
- Número total de mensagens entre tarefas.

- Número de mensagens enviadas e recebidas por tarefa.
- Os valores inicial e próximo para o gerador de números pseudoaleatórios.

VI.3 Emulador de MAD

A especificação do sistema MAD a ser emulado está no *arquivo de descrição.* Um sistema MAD é especificado pelo número e características dos seus computadores e canais de comunicação. Cada computador é caracterizado pela sua velocidade de execução. Cada canal de comunicação é caracterizado pela sua velocidade de transmissão, pelos computadores que interliga e pela sua latência, que é o tempo necessário para percorrê-lo. O arquivo de descrição de um sistema MAD contém todas estas informações.

Para facilitar a especificação de sistemas MAD homogêneos com topologias regulares, foram implementados geradores de subconjuntos de sistemas MAD. Estão disponíveis o *gerador de hipercubos* e o *gerador de toróides.* Estes geradores produzem arquivos de descrição, a partir do mínimo de informações necessárias a caracterização dos sistemas MAD gerados.

As decisões do algoritmo de escalonamento realizam o mapeamento das tarefas aos computadores ao longo do tempo. Este mapeamento além de determinar a utilização dos computadores pelas tarefas, também determina se as mensagens entre tarefas passam por canais de comunicação e influenciam o funcionamento dos serviços necessários à execução da APD, que também utilizam os computadores e os canais de comunicação.

O simulador contabiliza todos os custos necessários à execução da APD. Estes custos são relativos à execução de rotinas de serviço nos computadores. As mensagens de controle podem ser provenientes do algoritmo de escalonamento, ou do mecanismo de suporte a troca de mensagens entre as tarefas. As rotinas de serviço são as rotinas do algoritmo de escalonamento em ação, as rotinas do mecanismo de escalonamento local, as rotinas de recepção e envio de mensagens e as rotinas do mecanismo de suporte a troca de mensagens entre as tarefas.

VI.4 Módulo de Escalonamento Global

Para que um algoritmo distribuído de escalonamento dinâmico possa ser incorporado ao DDSS, é necessário que ele obedeça a algumas normas. Estas normas definem

uma interface entre o DDSS e o algoritmo e determinam a forma de apresentação do algoritmo. Esta seção apresenta esta interface.

O algoritmo deve obedecer a algumas precondições básicas:

- Estar expresso na linguagem de programação C.
- Assumir que em cada computador haverá uma instância sua em execução.
- Assumir que estas instâncias se comunicam entre si através da troca de mensagens.
- Contabilizar a estimativa do número de instruções executadas.

As normas para a especificação do algoritmo compreendem a definição dos eventos que provocam a sua ativação, dos protótipos das rotinas públicas do algoritmo, das informações disponíveis e suas estruturas de dados, dos protótipos das rotinas de serviço disponíveis e dos tipos de mensagens que podem ser enviados.

Os eventos que interessam diretamente ao algoritmo de escalonamento são os relativos a alteração das tarefas na fila de execução e os relativos a recepção de mensagens provenientes de outra de suas diversas instâncias. A alteração das tarefas na fila de execução se efetua quando ocorre a criação ou término de uma tarefa. As mensagens recebidas pelas instâncias do algoritmo podem pedir uma informação, como a carga de processamento de um computador vizinho, fornecer esta informação e pedir ou trazer uma tarefa para execução. A figura VI.1 exibe estes eventos.

```
EV_CREATE:      Criação de uma tarefa.
EV_TERMINATE:   Término de uma tarefa.
EV_GETINFO:     Recepção de uma informação.
EV_PUTINFO:     Pedido para a transmissão de uma informação.
EV_GETTASK:     Recepção de uma tarefa.
EV_PUTTASK:     Pedido para a transmissão de uma tarefa.
EV_CONTENT:     Contenção em um canal de comunicação.
```

Figura VI.1: Eventos que ativam o escalonador

A ocorrência de um dos eventos da figura VI.1 provoca a invocação de uma rotina pública do algoritmo de escalonamento. É através da execução dessa rotina, que o algoritmo realiza as ações de escalonamento. O algoritmo tem que apresentar ainda uma outra rotina pública para a sua iniciação. A figura a seguir mostra o protótipo dessas rotinas. O primeiro parâmetro em ambas corresponde à identificação do computador, os outros parâmetros correspondem à identificação do evento que ocorreu e ao canal de comunicação associado, se o evento refere-se a recepção de uma mensagem dirigida ao escalonador.

```
void    schedinit       (short p);
short   sched           (short p, short ev, CHAN *source);
```

Figura VI.2: Protótipos das rotinas públicas do escalonador

As estruturas de dados das informações que os algoritmos de escalonamento global podem consultar e manipular estão na figura VI.4.

```
void    sendtask        (TASK *tp, CHAN *cp, short p);
void    sendmesg        (CHAN *cp, MESG *mp);
short   chfree          (CHAN *c);
short   chbusy          (CHAN *c);
```

Figura VI.3: Protótipos das rotinas de serviço disponíveis ao escalonador

As rotinas de serviço disponíveis são:

- *sendmesg*: envia uma mensagem pelo canal de comunicação especificado.
- *sendtask*: pede a transferência de uma tarefa pelo canal de comunicação especificado. Esta rotina tem critérios próprios para a escolha da tarefa a ser transferida.
- *chfree*: verifica se um canal de comunicação está livre.
- *chbusy*: verifica se um canal de comunicação está muito ocupado.

Mensagens que podem ser enviadas às outras instâncias do escalonador:

- GETINFO: transmite uma informação a instância do escalonador global de um computador vizinho.
- PUTINFO: pede uma informação a instância do escalonador global de um computador vizinho.
- GETTASK: migra uma tarefa para a instância do escalonador global de um computador vizinho.
- PUTTASK: pede a migração de uma tarefa à instância do escalonador global de um computador vizinho.

VI.5 Módulo de Escalonamento Local

A interação do módulo de escalonamento local com o resto do simulador envolve muitos detalhes específicos referentes a programação dos demais componentes do simulador. Assim, a disponibilidade de um escalonador local padrão facilita a programação dos algoritmos de escalonamento a serem avaliados.

O mecanismo de escalonamento local determina a execução das tarefas presentes em cada um dos computadores. Ele segue a uma política "*FCFS*" ("primeiro a chegar, primeiro servido"). Uma tarefa executa até terminar, ou até que haja uma alguma restrição de sincronização, quando então é substituída por outra.

VI.6 Medidas de Desempenho

Esta seção apresenta os dados coletados da simulação, referentes a cada execução do DDSS.

A principal medida de desempenho coletada pelo DDSS é o tempo total de execução, que corresponde ao tempo de resposta da aplicação. Para cada computador são fornecidas os seguintes dados sobre o seu processador computacional:

- Taxa de utilização pela APD.
- Taxa de utilização pelo escalonador global.
- Taxa de ociosidade.

Para cada canal de comunicação são fornecidas os seguintes dados:

- Somatório dos tempos de espera das mensagens. Indica a contenção do canal de comunicação.
- Taxa de utilização pela APD.
- Taxa de utilização pelo escalonador global.
- Taxa de utilização pelo PC.

Para o sistema como um todo são fornecidas as médias e os desvios padrões dos dados acima relativos aos processadores computacionais e aos canais de comunicação.

VI.7 Seleção das Cargas Computacionais

Na seção II.3 as APD foram definidas como aplicações com um grau elevado de paralelismo de grão médio. Esta definição permite uma grande variação entre os valores que podem ser atribuídos a cada um dos parâmetros que no modelo das APD as caracterizam. A viabilidade do processo de simulação e a maior significância dos seus resultados dependem, respectivamente da limitação do número de diferentes combinações de valores atribuídos aos parâmetros e da representatividade destes valores.

Alguns dos parâmetros que caracterizam uma APD estão diretamente relacionados ao grau de paralelismo, ao grão de paralelismo e à demanda por capacidade de processamento e de comunicação. Como conseqüência da importância e abrangência destes parâmetros, o processo de seleção de valores se restringirá a eles. Estes parâmetros se referem ao número de instruções dos processamentos, ao número de instruções necessárias a criação de uma nova tarefa, ao número de palavras das mensagens e ao privilégio para a troca de mensagens entre tarefas do mesmo grupo.

O processo de seleção das características das cargas computacionais se realiza com base nos resultados da execução de um dos algoritmos de escalonamento sobre as diversas combinações de valores dos parâmetros a serem analisadas. O algoritmo utilizado deve apresentar um bom desempenho e ser representativo dos demais. A representatividade decorre da utilização de uma estratégia que seja abrangente em relação as operações de escalonamento que realiza.

Os parâmetros são organizados sob a forma de conjuntos de parâmetros relacionados. Há um conjunto referente as seqüências de instruções, outro referente as mensagens entre tarefas e outro com o privilégio para troca de mensagens entre tarefas do mesmo grupo. O primeiro conjunto constitúi-se dos seguintes parâmetros: MININST, MEDINST e MEDCHILD. O segundo conjunto constitúi-se dos seguintes parâmetros: MINMESG e MEDMESG. O terceiro conjunto constitúi-se apenas do parâmetro PRIV. Os valores utilizados devem cobrir um largo espectro do que possa se considerar como valores razoaveis para as APD.

Os valores a serem adotados para os parâmetros da cargas computacionais a que serão submetidos todos os demais algoritmos são selecionados a partir dos resultados das simulações com as diversas combinações de valores. São eliminadas as combinações de valores que se mostrem conflitantes com a faixa de operação do MAD; por exemplo porque as aplicações resultantes utilizam as capacidades de processamento e de comunicação desproporcionalmente às capacidades disponíveis. As combinações escolhidas devem abranger uma gama de APD que sejam adequadas

aos sistemas MAD, pois assim abrangerão qualquer APD cuja execução seja eficiente em um sistema MAD.

```
/*
 ****************************************************************
 *      Estrutura de uma tarefa                                 *
 ****************************************************************
 */
typedef struct  t       TASK;

struct  t
{
        short   t_id;           /* Identificação da tarefa */
        short   t_move;         /* Última transferência */
        short   t_nmesgs;       /* Número de mensagens já enviadas */
        MESG    *t_mesg;        /* Lista de mensagens a receber */
        TASK    *t_next;        /* Próxima tarefa no processador */
        ...

};
/*
 ****************************************************************
 *      Estrutura de um canal.                                  *
 ****************************************************************
 */
type    struct  c       CHAN;

struct  c
{
        short   c_dest;         /* Destino do canal */
        CHAN    *c_dual;        /* Canal no sentido contrário */
        CHAN    *c_next;        /* Próximo canal */
        float   c_busy;         /* Término do último envio */
        long    c_neighinfo;    /* Informação enviada pelo vizinho */
        long    c_owninfo;      /* Última informação enviada */
        ...

};
/*
 ****************************************************************
 *      Estrutura de um processador.                            *
 ****************************************************************
 */
type    struct  p       PROC;

struct  p
{
        int     p_load;         /* Número de tarefas */
        bool    p_idle;         /* idle ?  */
        CHAN    *p_chan;        /* Lista de canais de comunicação */
        TASK    *p_task;        /* Círculo de tarefas já iniciadas */
        ...

};

#define NOTASK  (TASK *)0
#define NOCHAN  (CHAN *)0
#define NOPROC  (PROC *)0
```

Figura VI.4: Estruturas dos dados disponíveis ao escalonador

Capítulo VII

Simulações Realizadas

Este capítulo descreve as simulações realizadas. São especificados os algoritmos escolhidos para avaliação, os parâmetros das cargas computacionais empregadas e os sistemas MAD selecionados. Por último são apresentados os resultados encontrados, uma avaliação do processo de simulação realizado e uma interpretação destes resultados.

VII.1 Algoritmos Avaliados

Alguns dos algoritmos propostos em outros trabalhos foram selecionados para serem avaliados. Estes são os algoritmos que poderiam ser adaptados para o escalonamento em MAD. As tabelas VII.1 e VII.2 apresentam os algoritmos selecionados.

Classificação dos Algoritmos Avaliados				
Algoritmo	Autonomia	Distribuição	Iniciativa	Mobilidade
Buddy set	cooperativo	divisão	ambos	irrestrita
Gradient	cooperativo	divisão	receptor	irrestrita
Greedy	não cooperativo	divisão	fornecedor	irrestrita
Sender	não cooperativo	divisão	fornecedor	restrita

Tabela VII.1: Classificação dos Algoritmos Avaliados

Foram preparadas duas versões de cada algoritmo. A versão **otimizada** tem por objetivo minimizar a contenção dos canais de comunicação. A versão **simples** não tem qualquer objetivo em relação a contenção dos canais de comunicação.

Todos os algoritmos, nas suas duas versões, apresentam algumas características em comum, quanto a gerência de informações:

Política de Transferência				
Algoritmo	Ativação	Regra	Parceiro	Tarefa
Buddy set	alteração da carga	limiares	vizinhos	qualquer
Gradient	término de tarefas	limiares	menos carregado	qualquer
Greedy	criação de tarefas	limiar f(n)	menos carregado	tarefa criada
Sender	criação de tarefas	limiar	aleatório	tarefa criada

Tabela VII.2: Sumário da Política de Transferência dos Algoritmos Avaliados

- As decisões e informações de escalonamento referem-se somente ao próprio computador, ou aos computadores vizinhos.
- A gerência de informações de todos os algoritmos utiliza as informações mais recentes enviadas pelos vizinhos.
- Como regra geral, as informações de um computador só são atualizadas nos seus vizinhos se forem diferentes da última informação comunicada.

A adoção destas características tem por objetivo eliminar as mensagens redundantes entre as instâncias dos algoritmos de escalonamento e assim possibilitar um melhor desempenho geral do sistema. Uma conseqüência da adoção destas características é a redução da diferença de desempenho entre os algoritmos de escalonamento e entre suas versões.

As versões de cada algoritmo se diferenciam pelos critérios da sua política de escolha da tarefa a migrar e pelo método de avaliação da carga computacional. Na versão simples é escolhida qualquer tarefa disponível para a migração e a carga computacional em um computador é estimada pelo número de tarefas presentes neste computador. Na versão otimizada, há o reconhecimento que não são todas as tarefas que estão prontas para executar (elas podem estar esperando por mensagens). Assim todas as tarefas disponíveis para migração são analisadas. A tarefa é escolhida preferencialmente em função da sua história de comunicação, ou se não for possível, em função do seu custo de migração. Na versão otimizada a carga computacional é estimada pelo número de tarefas prontas.

A forma de contabilizar a história de comunicações de uma tarefa é muito simples. Para cada tarefa tem-se o somatório de palavras enviadas a tarefas em outros computadores por cada canal de comunicação e enviadas a tarefas no mesmo computador. Estes somatórios estão na lista de tarefas presentes no computador, que é controlada pelo processador de comunicações. A história de comunicação de uma tarefa é iniciada quando a tarefa executa pela primeira vez e após cada migração.

A seleção da tarefa a migrar a partir da sua história de comunicação, escolhe entre as tarefas que apresentam uma história mínima de comunicação (50 palavras), aquela que apresenta o maior percentual de comunicações com o canal de comunicação para o qual se quer migrar uma tarefa, cujo custo de migração não seja muito grande. Se o percentual da tarefa escolhida for inferior a um limiar mínimo (50%), é escolhida a tarefa com o menor custo de migração, entre as tarefas cujo percentual de comunicações com tarefas no mesmo computador seja inferior a um limiar máximo (30 %).

VII.2 Cargas Computacionais Empregadas

A determinação das características da cargas computacionais escolhidas foi realizada através do um processo de seleção definido na seção VI.7.

O algoritmo "greedy" foi utilizado no processo de avaliação dos valores atribuídos aos diversos parâmetros das cargas computacionais. Este algoritmo apresenta um bom desempenho geral e um baixo custo.

Conjuntos de Valores para a Demanda por Processamento				
Conjunto	Distribuição	MININST	MEDINST	MEDCHILD
0	Exponencial	4	64	96
1	Exponencial	8	128	192
2	Exponencial	16	256	384
3	Exponencial	24	384	576
4	Exponencial	32	512	768
5	Exponencial	50	800	1200

Tabela VII.3: Conjuntos de Valores para a Demanda por Processamento

Privilégios para a Comunicação Intragrupal		
Valor	Distribuição	PRIV
0	Uniforme	0.0%
1	Uniforme	50.0%
2	Uniforme	90.0%

Tabela VII.4: Privilégios para a Comunicação Intragrupal

Os valores utilizados nos conjuntos de parâmetros referentes a demanda por processamento são dados pela tabela VII.3, os valores referentes a demanda por co-

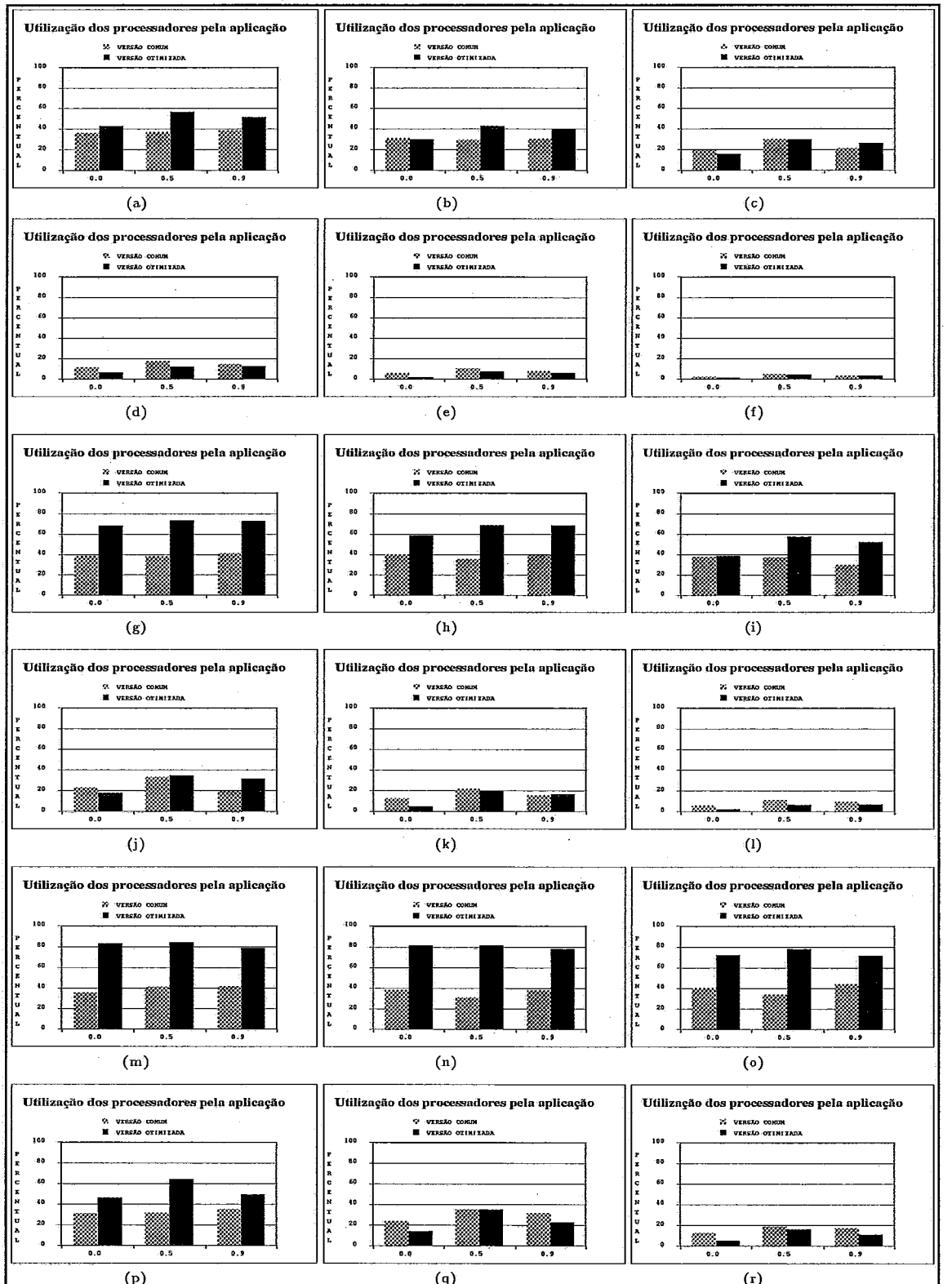
municação são dados pela tabela VII.5 e os valores para o privilégio de comunicação entre tarefas do mesmo grupo estão na tabela VII.4.

Conjuntos de Valores para a Demanda por Comunicação			
Conjunto	Distribuição	MINMESG	MEDMESG
1	Exponencial	2	16
2	Exponencial	4	32
3	Exponencial	8	64
4	Exponencial	16	128
5	Exponencial	32	256
6	Exponencial	64	512

Tabela VII.5: Conjuntos de Valores para a Demanda por Comunicação

As figuras a seguir mostram os resultados das simulações com todas as combinações de valores para os parâmetros em estudo em seus principais aspectos. Para maior clareza os resultados são apresentados sob a forma de conjuntos de histogramas. Cada histograma apresenta os valores médios de alguma medida referente aos processadores ou aos canais de comunicação. Cada conjunto de histogramas apresenta um aspecto dos resultados das simulações.

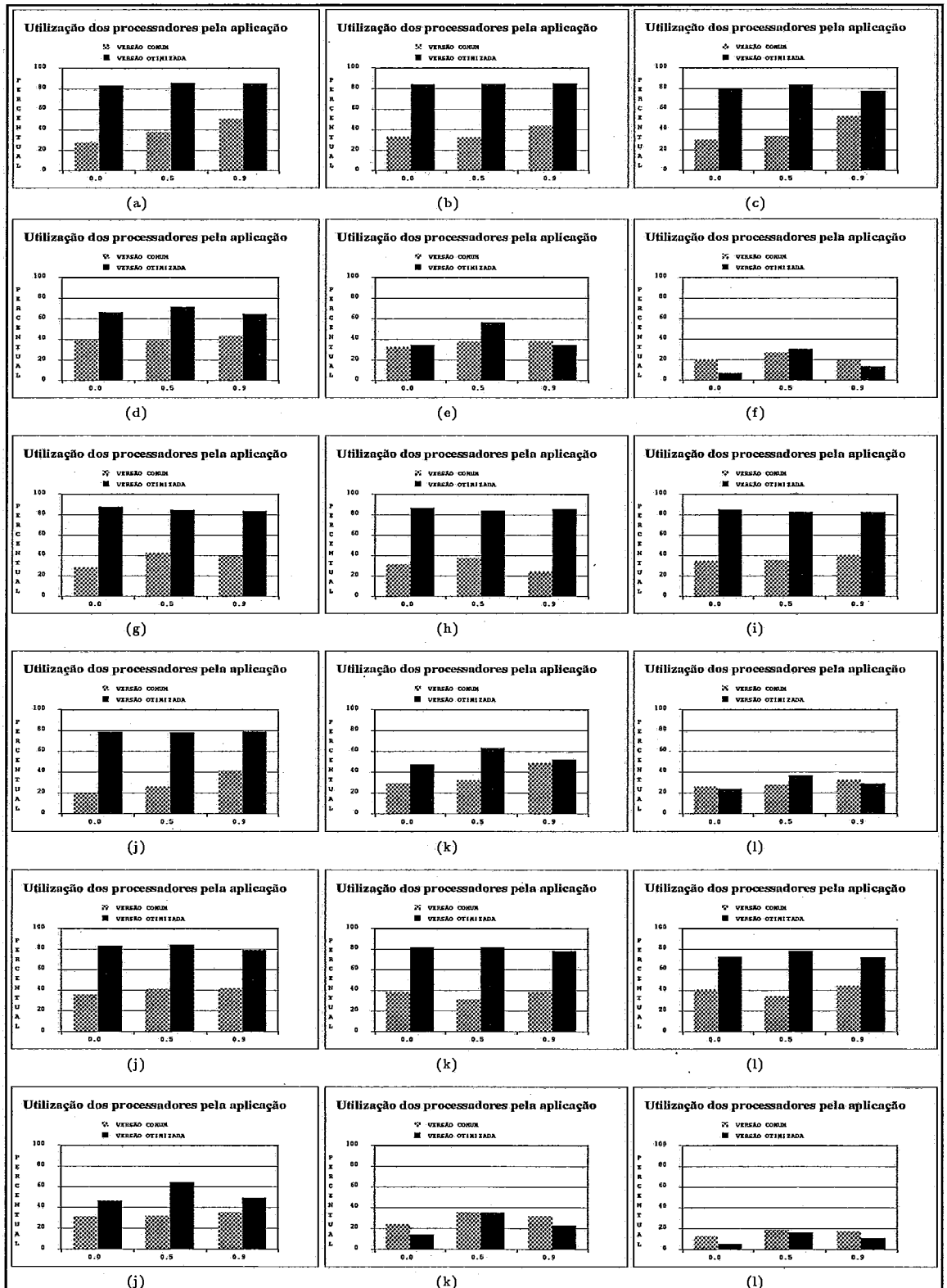
As figuras VII.1 e VII.2 apresentam a taxa de utilização dos processadores pelas aplicações. As figuras VII.3 e VII.4 apresentam a taxa de utilização dos canais de comunicação pelas aplicações. As figuras VII.5 e VII.6 apresentam a taxa de utilização dos canais de comunicação. As figuras VII.7 e VII.8 apresentam um índice da contenção dos canais de comunicação, este índice corresponde ao somatório dos tempos de espera de todas as mensagens que passaram por um canal. As figuras VII.9 e VII.10 apresentam os tempos de resposta das aplicações.



Instruções: (a) (b) (c) (d) (e) (f) média 64, mínimo 4;
(g) (h) (i) (j) (k) (l) média 128, mínimo 8; (m) (n) (o) (p) (q) (r) média 256, mínimo 16.

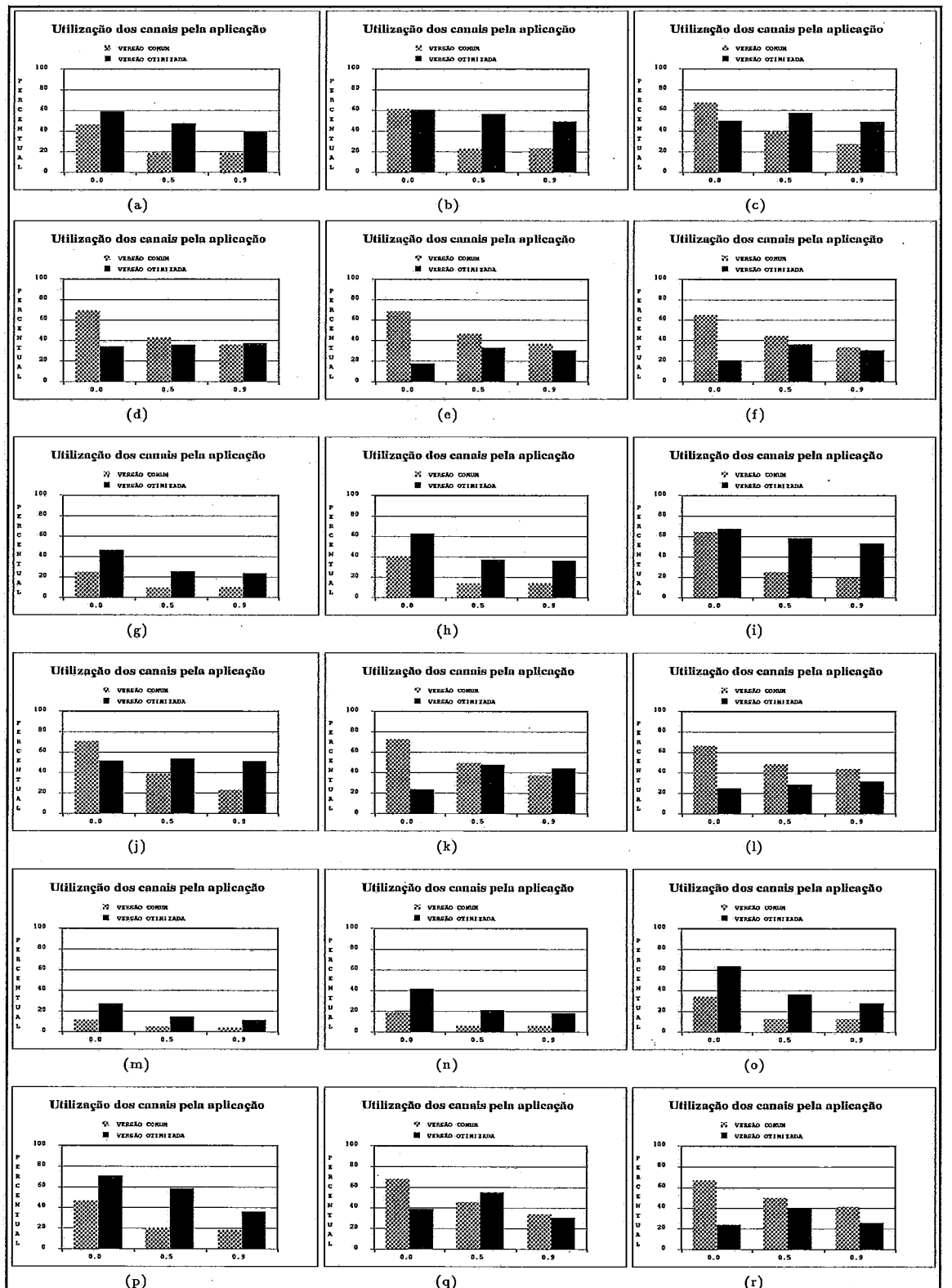
Mensagens: (a) (g) (m) média 16, mínimo 2; (b) (h) (n) média 32, mínimo 4; (c) (i) (o) média 64, mínimo 8;
(d) (j) (p) média 128, mínimo 16; (e) (k) (q) média 256, mínimo 32; (f) (l) (r) média 512, mínimo 64;

Figura VII.1: Utilização dos processadores pela aplicação



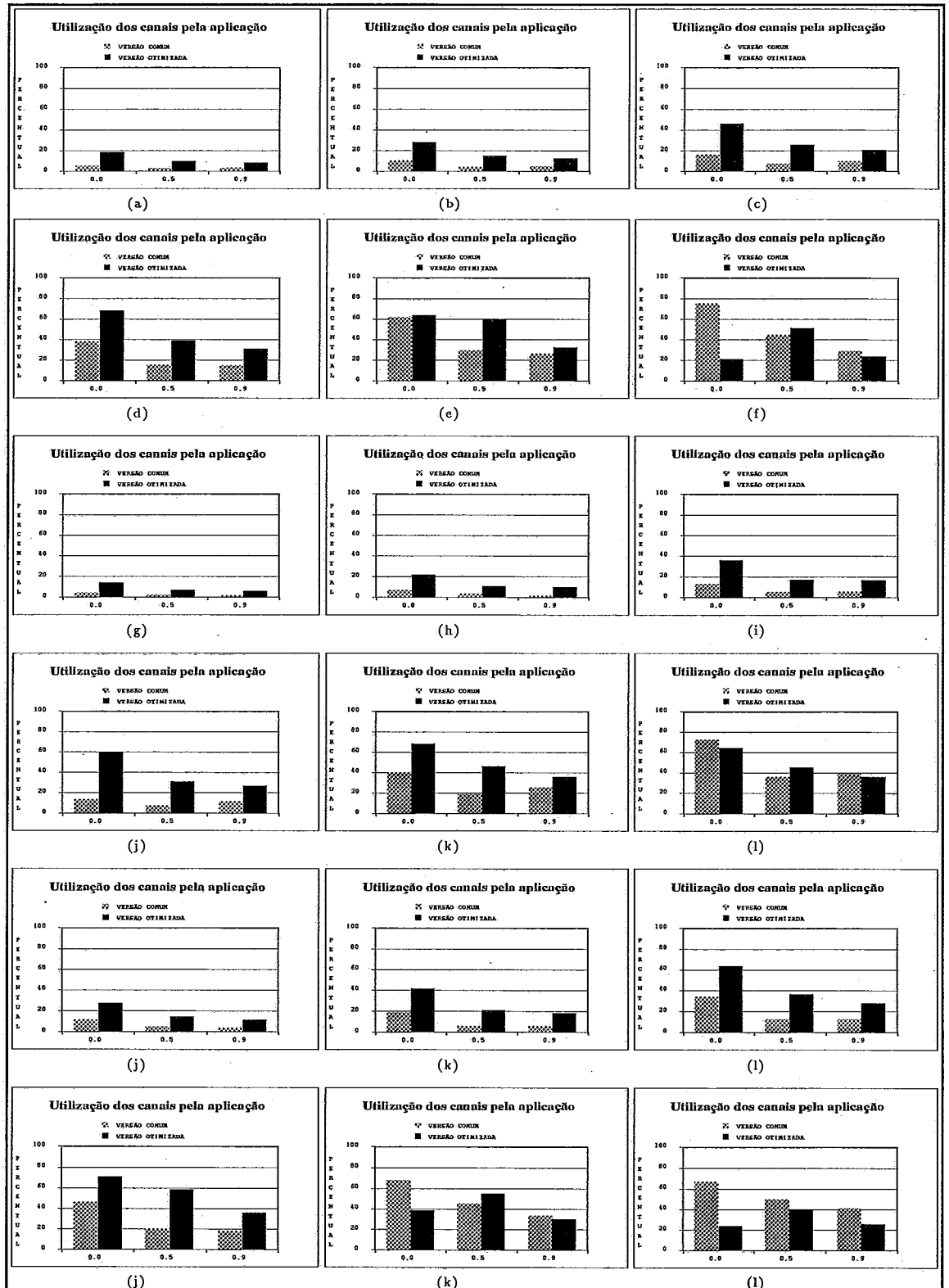
Instruções: (a) (b) (c) (d) (e) (f) média 384, mínimo 24;
(g) (h) (i) (j) (k) (l) média 512, mínimo 32 (m) (n) (o) (p) (q) (r) média 800, mínimo 50.
Mensagens: (a) (g) (m) média 16, mínimo 2; (b) (h) (n) média 32, mínimo 4; (c) (i) (o) média 64, mínimo 8;
(d) (j) (p) média 128, mínimo 16; (e) (k) (q) média 256, mínimo 32; (f) (l) (r) média 512, mínimo 64;

Figura VII.2: Utilização dos processadores pela aplicação



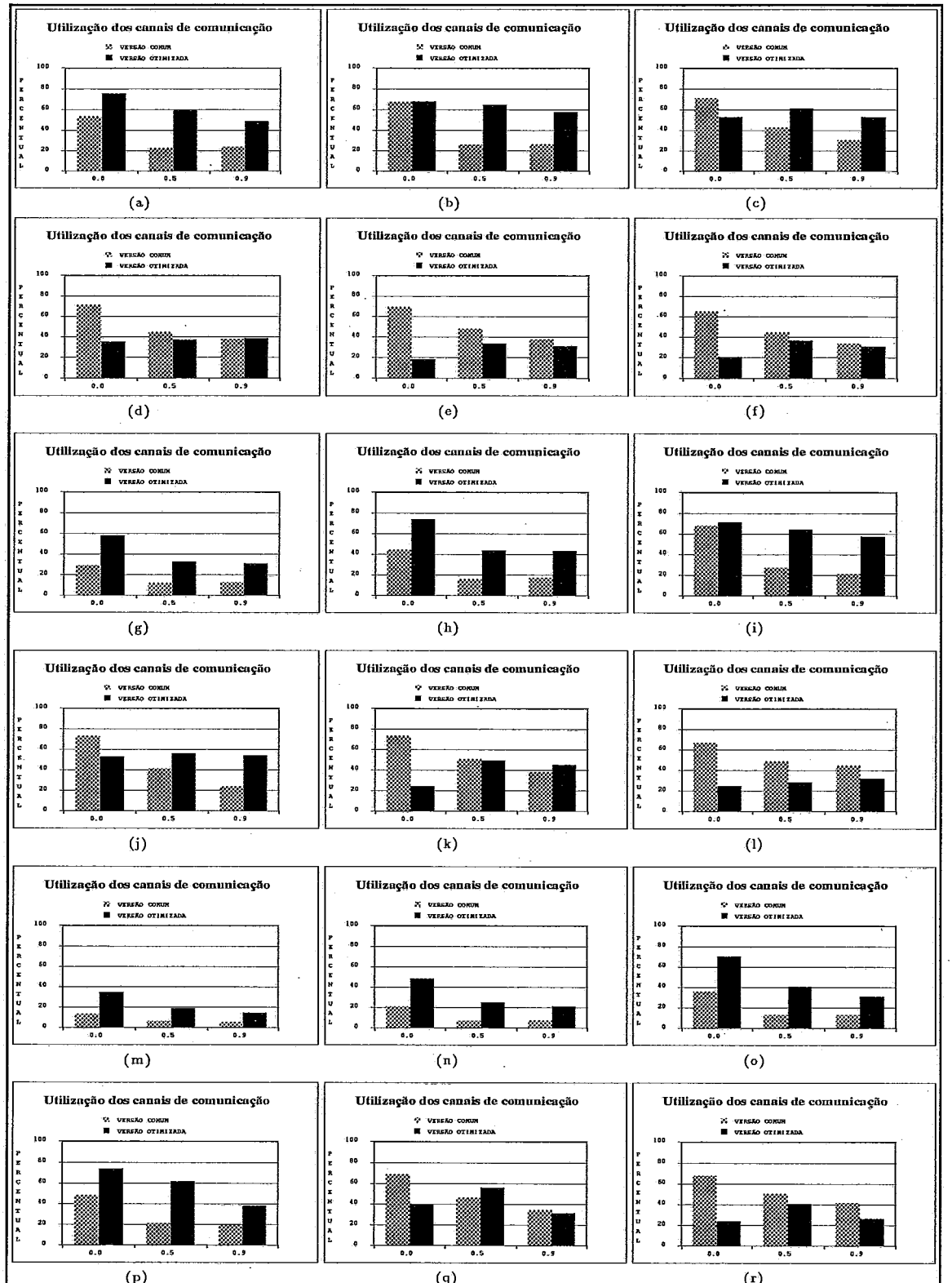
Instruções: (a) (b) (c) (d) (e) (f) média 64, mínimo 4;
(g) (h) (i) (j) (k) (l) média 128, mínimo 8; (m) (n) (o) (p) (q) (r) média 256, mínimo 16.
Mensagens: (a) (g) (m) média 16, mínimo 2; (b) (h) (n) média 32, mínimo 4; (c) (i) (o) média 64, mínimo 8;
(d) (j) (p) média 128, mínimo 16; (e) (k) (q) média 256, mínimo 32; (f) (l) (r) média 512, mínimo 64;

Figura VII.3: Utilização dos canais de comunicação pela aplicação



Instruções: (a) (b) (c) (d) (e) (f) média 384, mínimo 24;
(g) (h) (i) (j) (k) (l) média 512, mínimo 32 (m) (n) (o) (p) (q) (r) média 800, mínimo 50.
Mensagens: (a) (g) (m) média 16, mínimo 2; (b) (h) (n) média 32, mínimo 4; (c) (i) (o) média 64, mínimo 8;
(d) (j) (p) média 128, mínimo 16; (e) (k) (q) média 256, mínimo 32; (f) (l) (r) média 512, mínimo 64;

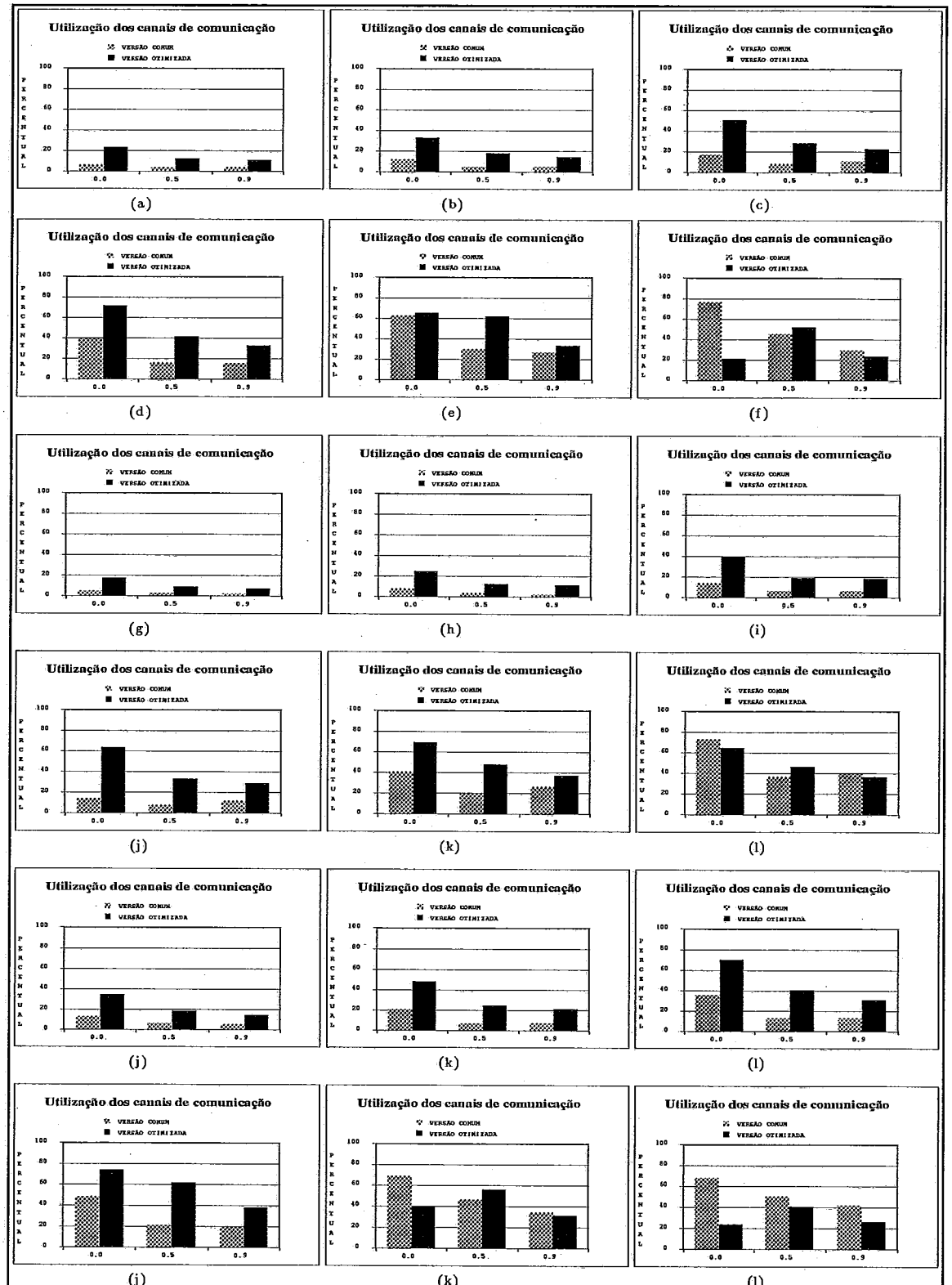
Figura VII.4: Utilização dos canais de comunicação pela aplicação



Instruções: (a) (b) (c) (d) (e) (f) média 64, mínimo 4;
(g) (h) (i) (j) (k) (l) média 128, mínimo 8; (m) (n) (o) (p) (q) (r) média 256, mínimo 16.

Mensagens: (a) (g) (m) média 16, mínimo 2; (b) (h) (n) média 32, mínimo 4; (c) (i) (o) média 64, mínimo 8;
(d) (j) (p) média 128, mínimo 16; (e) (k) (q) média 256, mínimo 32; (f) (l) (r) média 512, mínimo 64;

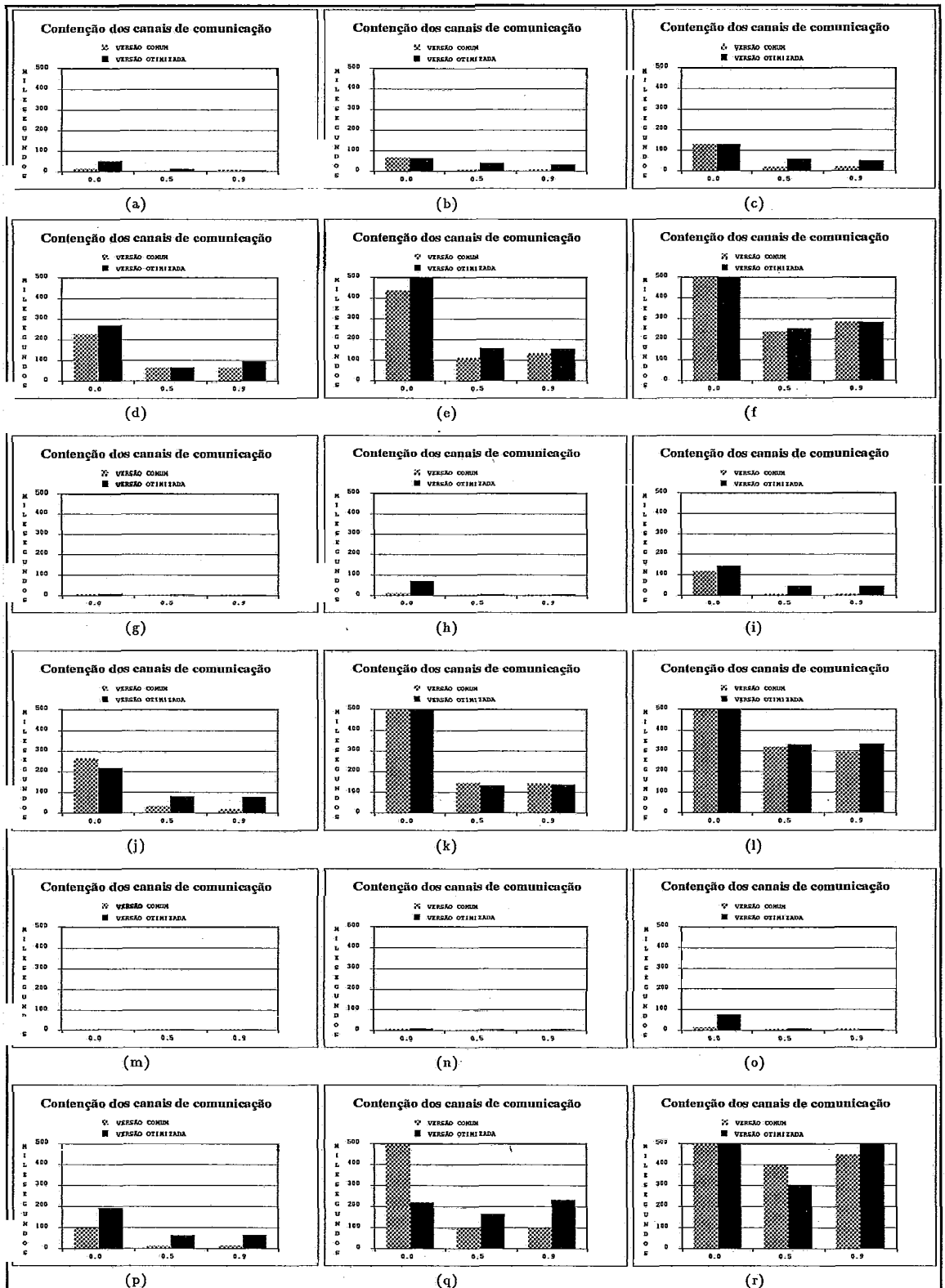
Figura VII.5: Utilização dos canais de comunicação



Instruções: (a) (b) (c) (d) (e) (f) média 384, mínimo 24;
(g) (h) (i) (j) (k) (l) média 512, mínimo 32 (m) (n) (o) (p) (q) (r) média 800, mínimo 50.

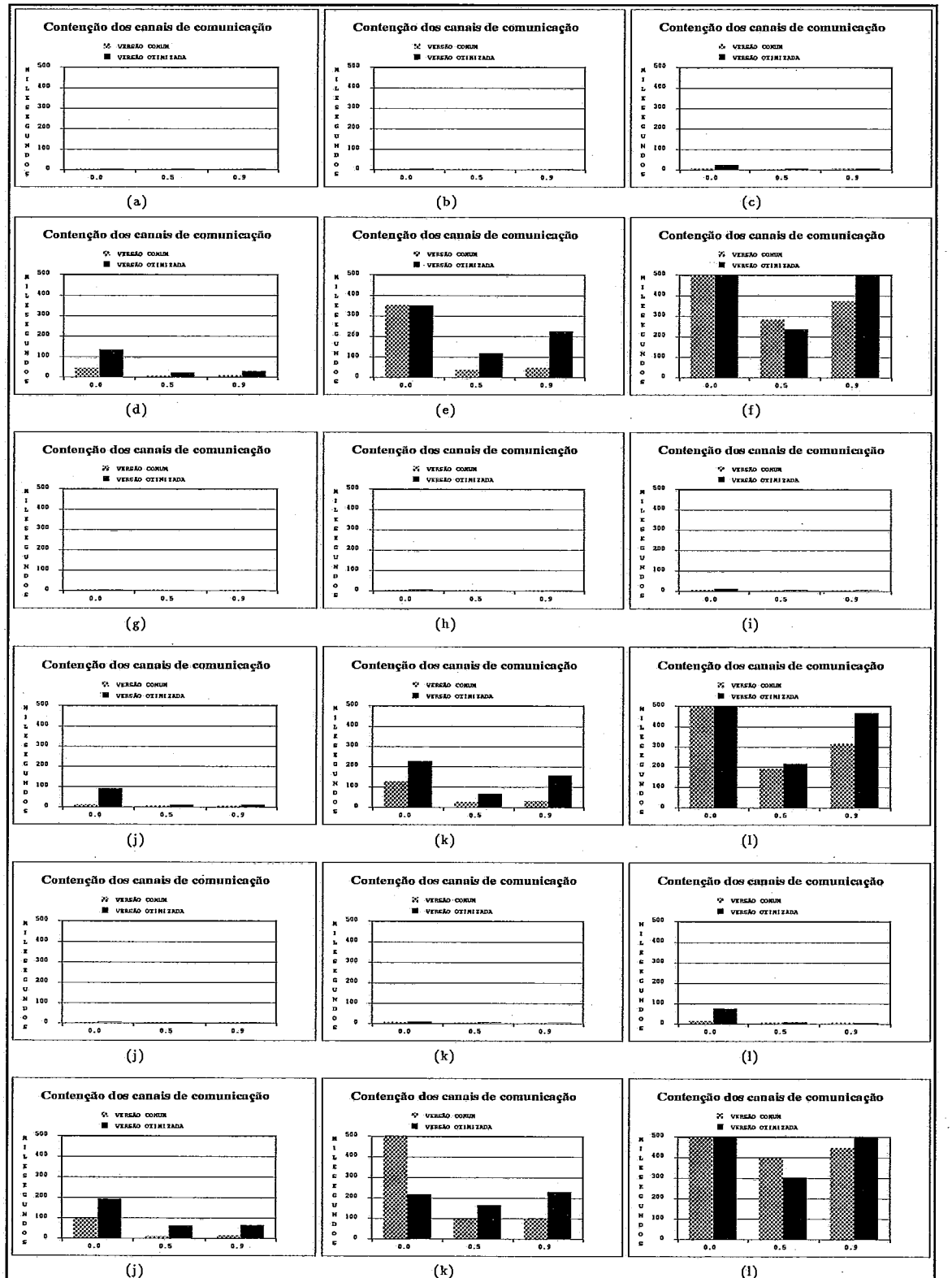
Mensagens: (a) (g) (m) média 16, mínimo 2; (b) (h) (n) média 32, mínimo 4; (c) (i) (o) média 64, mínimo 8;
(d) (j) (p) média 128, mínimo 16; (e) (k) (q) média 256, mínimo 32; (f) (l) (r) média 512, mínimo 64;

Figura VII.6: Utilização dos canais de comunicação



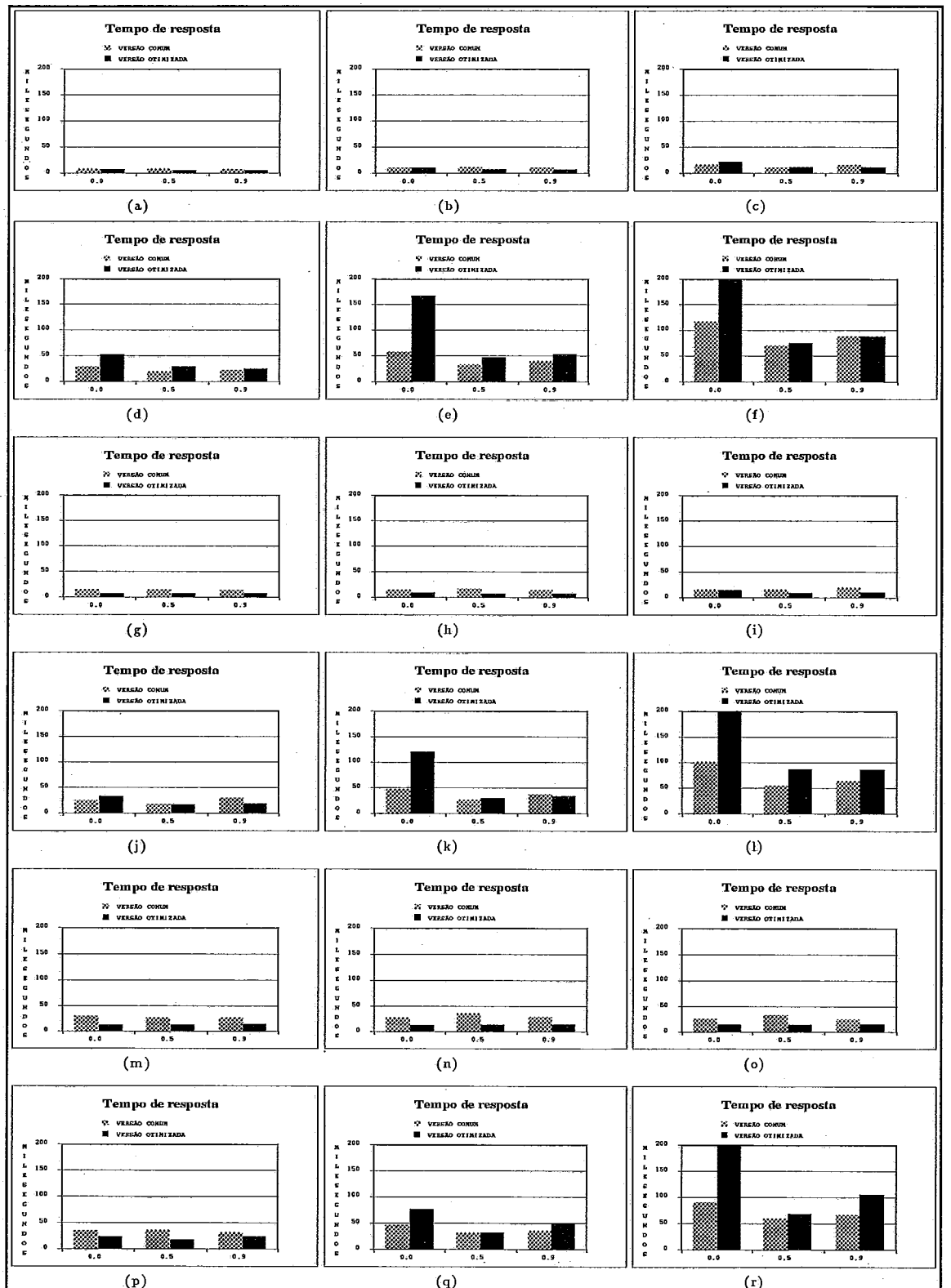
Instruções: (a) (b) (c) (d) (e) (f) média 64, mínimo 4;
(g) (h) (i) (j) (k) (l) média 128, mínimo 8; (m) (n) (o) (p) (q) (r) média 256, mínimo 16.
Mensagens: (a) (g) (m) média 16, mínimo 2; (b) (h) (n) média 32, mínimo 4; (c) (i) (o) média 64, mínimo 8;
(d) (j) (p) média 128, mínimo 16; (e) (k) (q) média 256, mínimo 32; (f) (l) (r) média 512, mínimo 64;

Figura VII.7: Contenção dos canais de comunicação



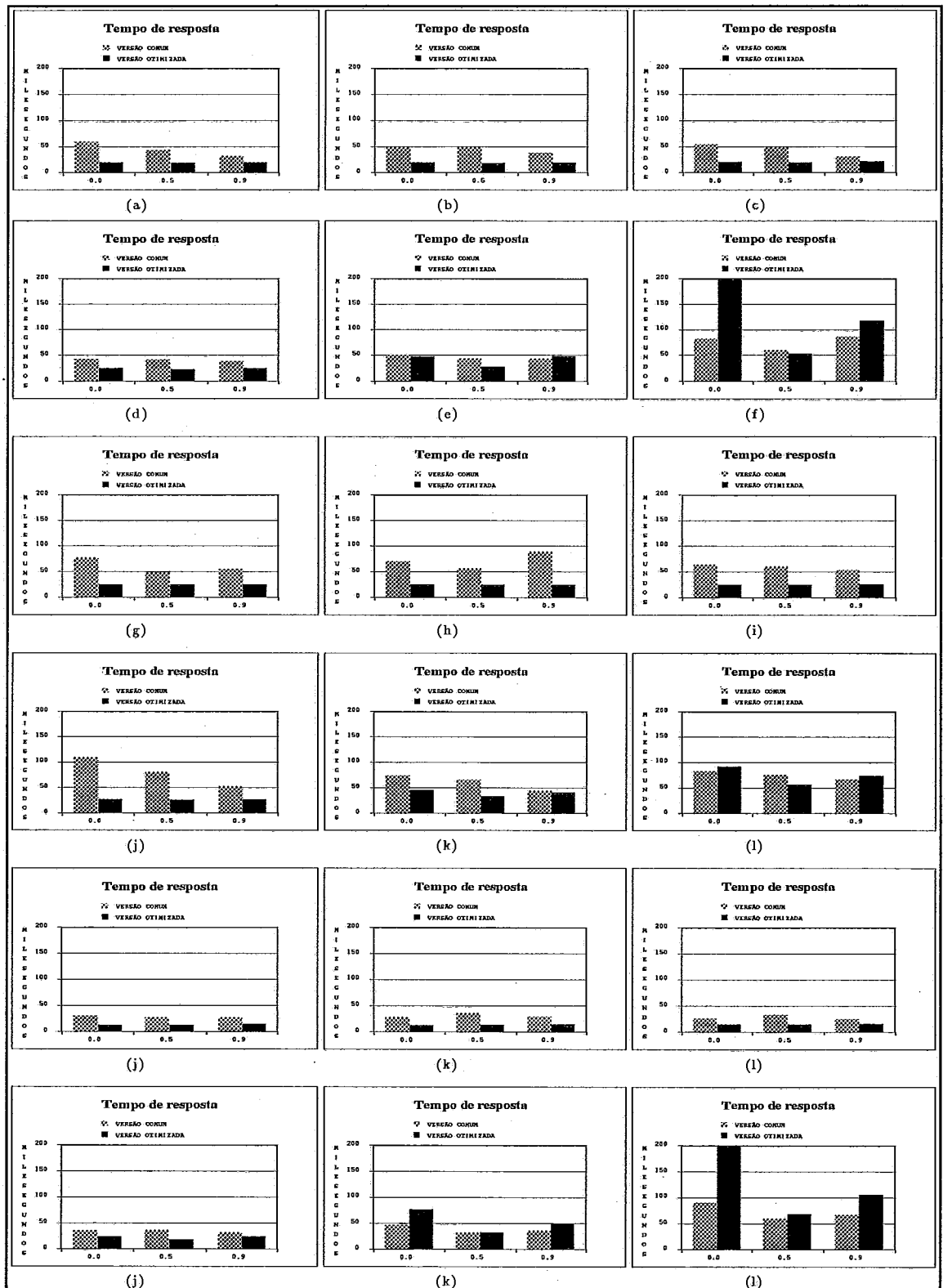
Instruções: (a) (b) (c) (d) (e) (f) média 384, mínimo 24;
(g) (h) (i) (j) (k) (l) média 512, mínimo 32 (m) (n) (o) (p) (q) (r) média 800, mínimo 50.
Mensagens: (a) (g) (m) média 16, mínimo 2; (b) (h) (n) média 32, mínimo 4; (c) (i) (o) média 64, mínimo 8;
(d) (j) (p) média 128, mínimo 16; (e) (k) (q) média 256, mínimo 32; (f) (l) (r) média 512, mínimo 64;

Figura VII.8: Contenção dos canais de comunicação



Instruções: (a) (b) (c) (d) (e) (f) média 64, mínimo 4;
(g) (h) (i) (j) (k) (l) média 128, mínimo 8; (m) (n) (o) (p) (q) (r) média 256, mínimo 16.
Mensagens: (a) (g) (m) média 16, mínimo 2; (b) (h) (n) média 32, mínimo 4; (c) (i) (o) média 64, mínimo 8;
(d) (j) (p) média 128, mínimo 16; (e) (k) (q) média 256, mínimo 32; (f) (l) (r) média 512, mínimo 64;

Figura VII.9: Tempo de resposta da aplicação



Instruções: (a) (b) (c) (d) (e) (f) média 384, mínimo 24;
(g) (h) (i) (j) (k) (l) média 512, mínimo 32 (m) (n) (o) (p) (q) (r) média 800, mínimo 50.
Mensagens: (a) (g) (m) média 16, mínimo 2; (b) (h) (n) média 32, mínimo 4; (c) (i) (o) média 64, mínimo 8;
(d) (j) (p) média 128, mínimo 16; (e) (k) (q) média 256, mínimo 32; (f) (l) (r) média 512, mínimo 64;

Figura VII.10: Tempo de resposta da aplicação

Foram selecionados dois conjuntos de valores para cada conjunto de parâmetros. Como conseqüência foram escolhidas oito combinações para os valores dos parâmetros, gerando oito tipos de cargas computacionais. A tabela VII.6 mostra as combinações escolhidas.

Parâmetros das Cargas Computacionais Empregadas			
Parâmetro	Distribuição	Média	Corte
Instruções de um Processamento 1	Exponencial	256	16
Instruções de um Processamento 2	Exponencial	512	32
Tamanho das Mensagens 1	Exponencial	32	4
Tamanho das Mensagens 2	Exponencial	256	32
Privilégio do Grupo 1	Uniforme	.50	—
Privilégio do Grupo 2	Uniforme	.90	—
Criação de Grupo	Uniforme	.05	—
Custo médio de Migração	Exponencial	100	0
Total de Tarefas	—	—	6000

Tabela VII.6: Parâmetros das Cargas Computacionais Empregadas

VII.3 Sistemas MAD Adotados

Foram adotados para simulação uma topologia freqüentemente empregada em sistemas paralelos de alto desempenho: hipercubos. Foi emulado um hipercubo de dimensão 5, com 32 computadores e 160 canais de comunicação. Cada computador do hipercubo tem a capacidade de processar cinquenta milhões de instruções por segundo, e os canais de comunicação tem a capacidade de transmitir vinte milhões de palavras por segundo, com um tempo de ciclo igual a cinquenta nanosegundos. Estes dados relativos as características de velocidade dos computadores e dos canais de comunicação são semelhantes as mesmas capacidades dos sistemas MAD atuais, como a *"maquina J"* [17] em desenvolvimento no MIT.

VII.4 Resultados e Avaliação

Do mesmo modo que na seção VII.2 os resultados das simulações são apresentados sob a forma de conjuntos de histogramas. Cada conjunto refere-se ao desempenho dos algoritmos quanto a um aspecto. As figuras a seguir apresentam os histogramas.

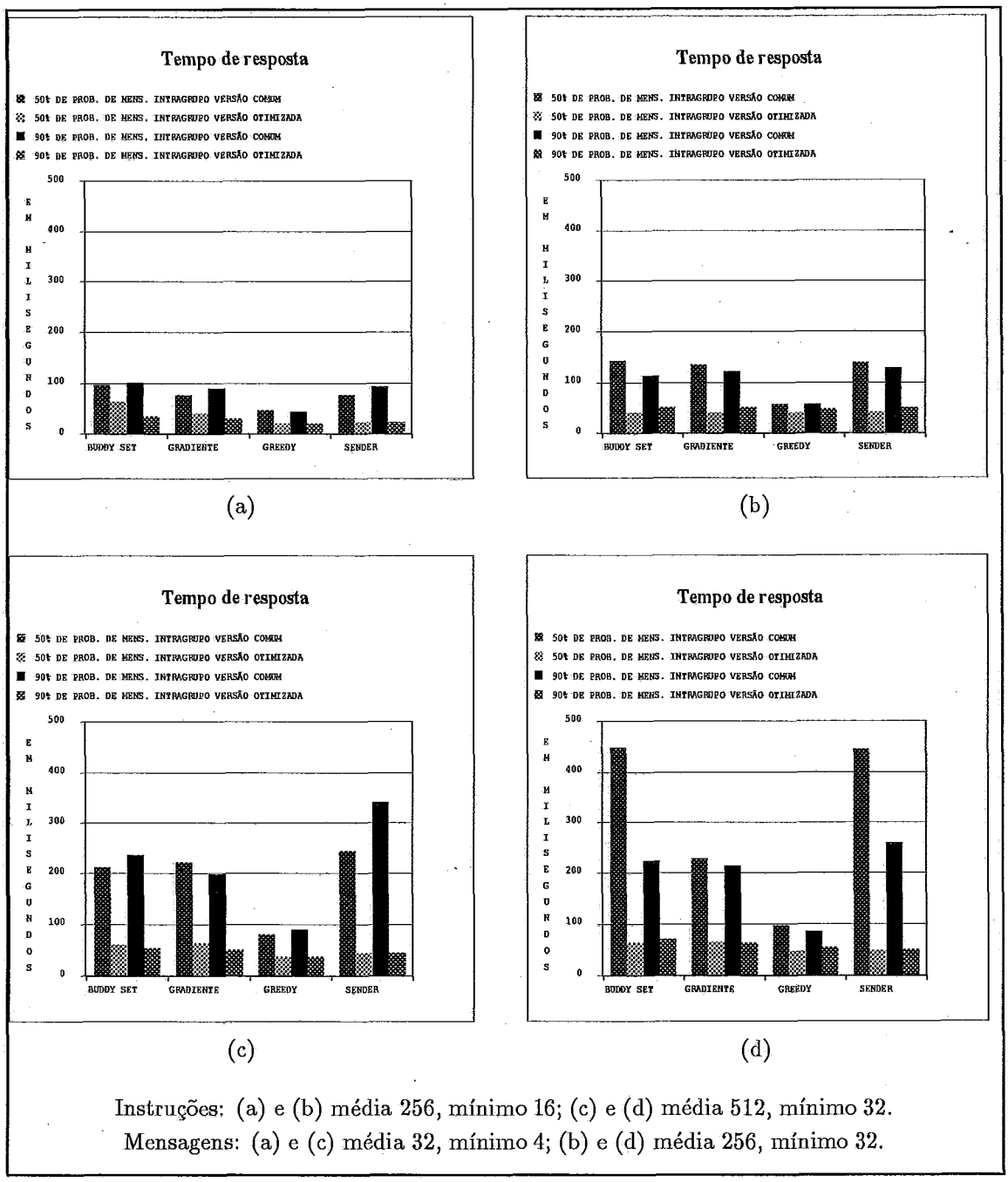
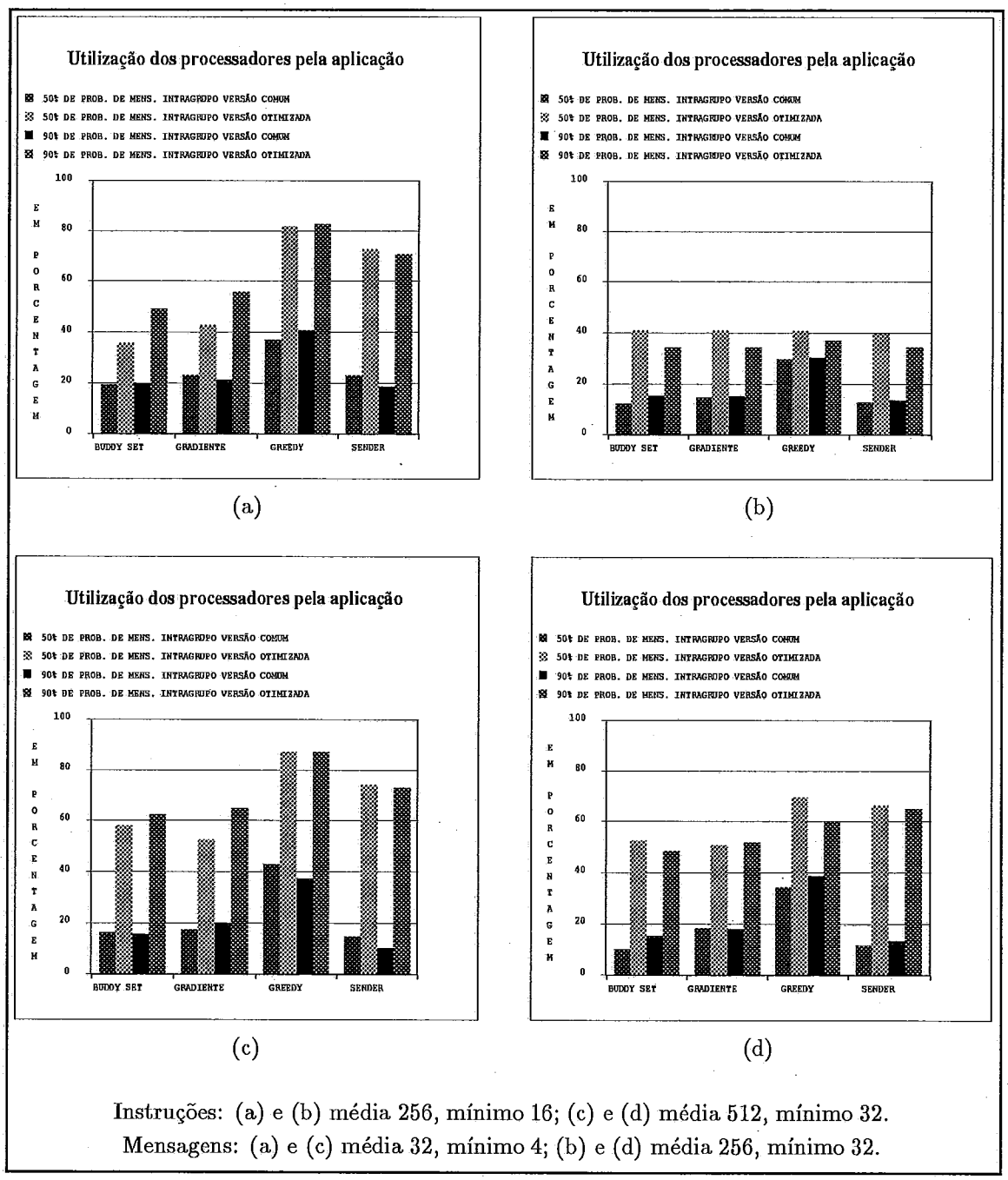


Figura VII.11: Tempo de resposta da aplicação

O tempo de resposta da aplicação é a medida que efetivamente avalia a qualidade dos escalonadores. Os histogramas mostram que em todos algoritmos avaliados a versão otimizada apresenta um menor tempo de resposta e portanto apresenta um melhor desempenho.



Instruções: (a) e (b) média 256, mínimo 16; (c) e (d) média 512, mínimo 32.
Mensagens: (a) e (c) média 32, mínimo 4; (b) e (d) média 256, mínimo 32.

Figura VII.12: Utilização dos processadores pela aplicação

A taxa de utilização dos processadores pela aplicação indica a eficiência dos escalonadores quanto ao aproveitamento da capacidade de processamento do sistema. As versões otimizadas também neste aspecto apresentam um desempenho superior as versões comuns.

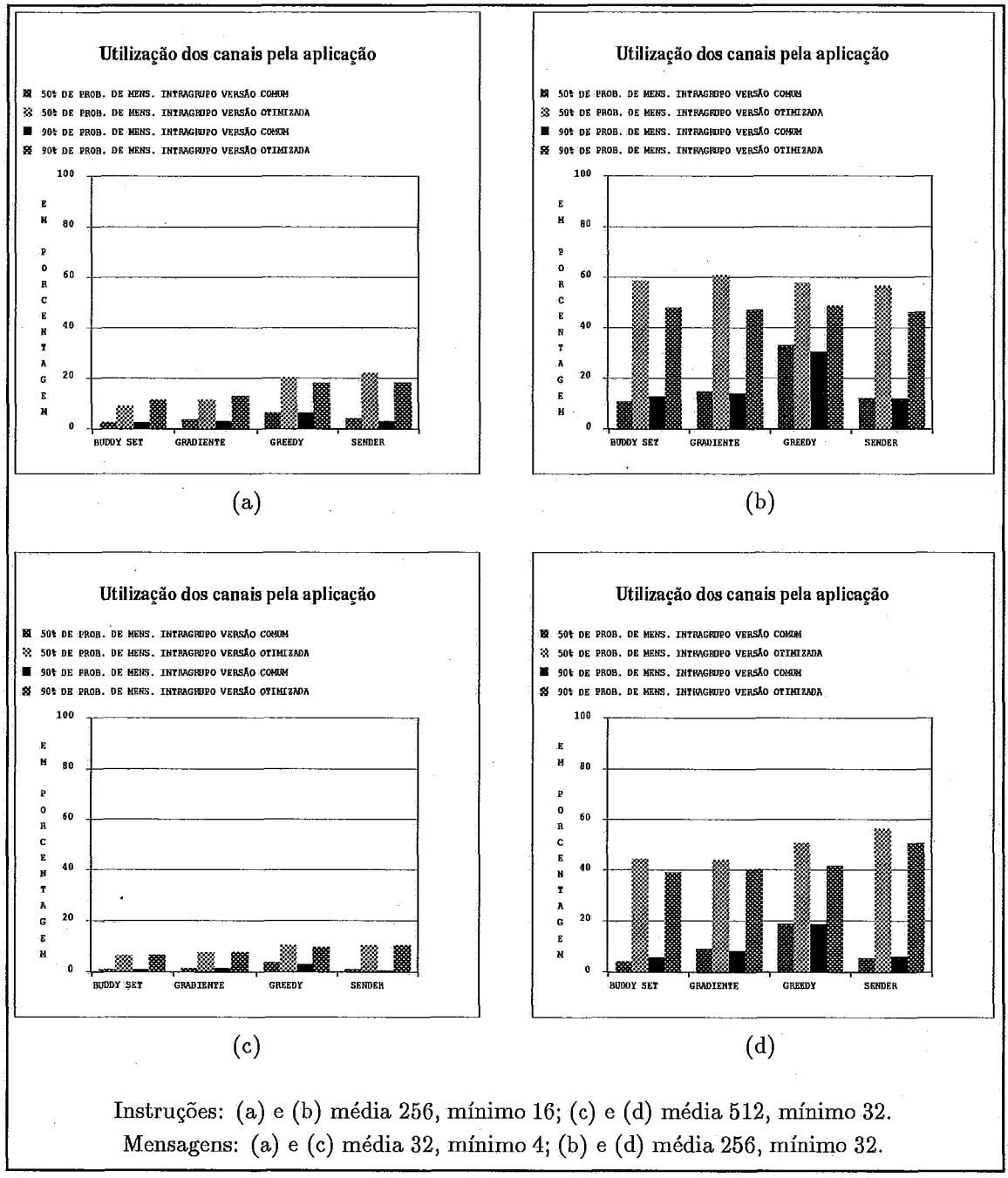
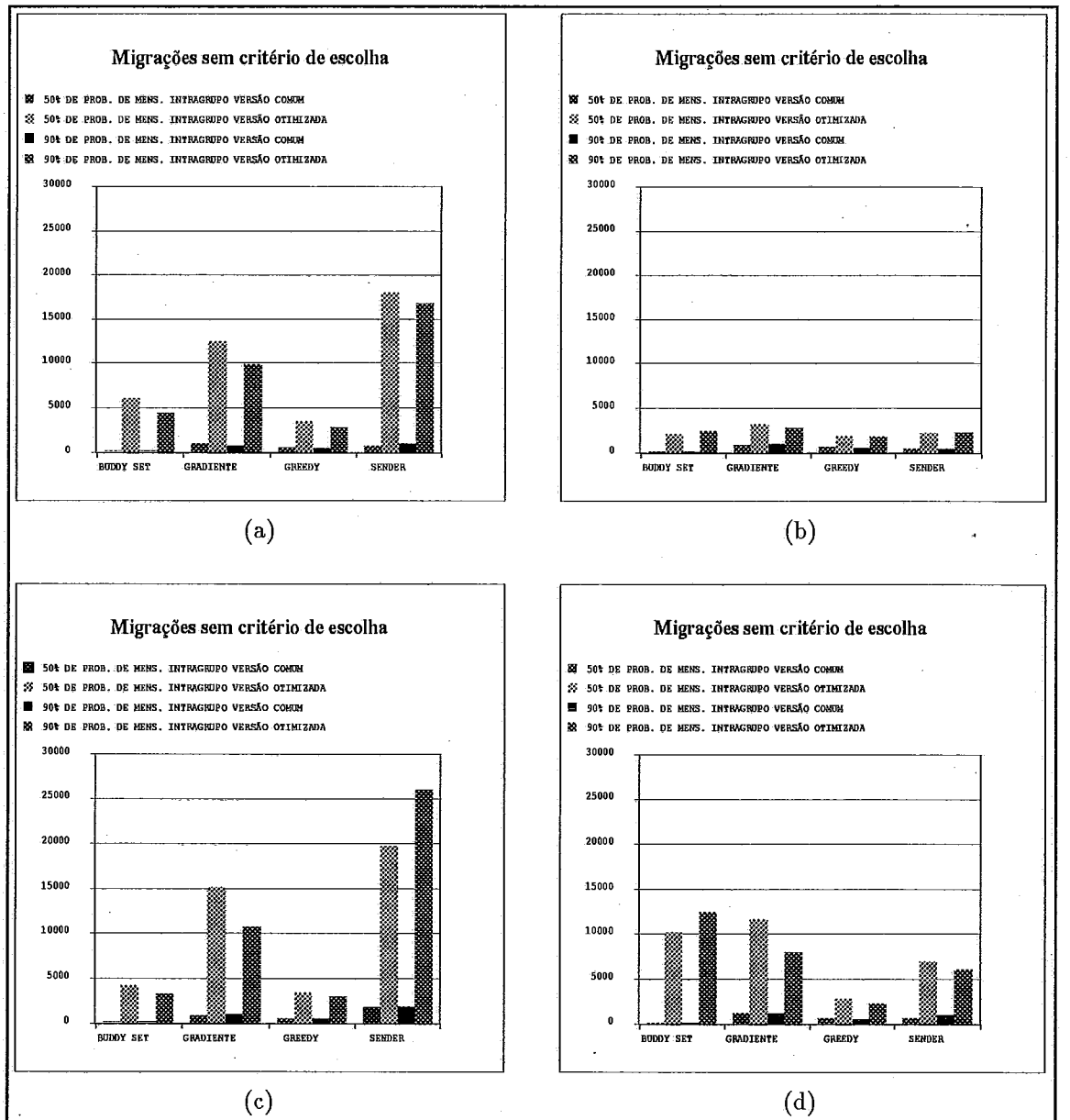


Figura VII.13: Utilização dos canais de comunicação pela aplicação

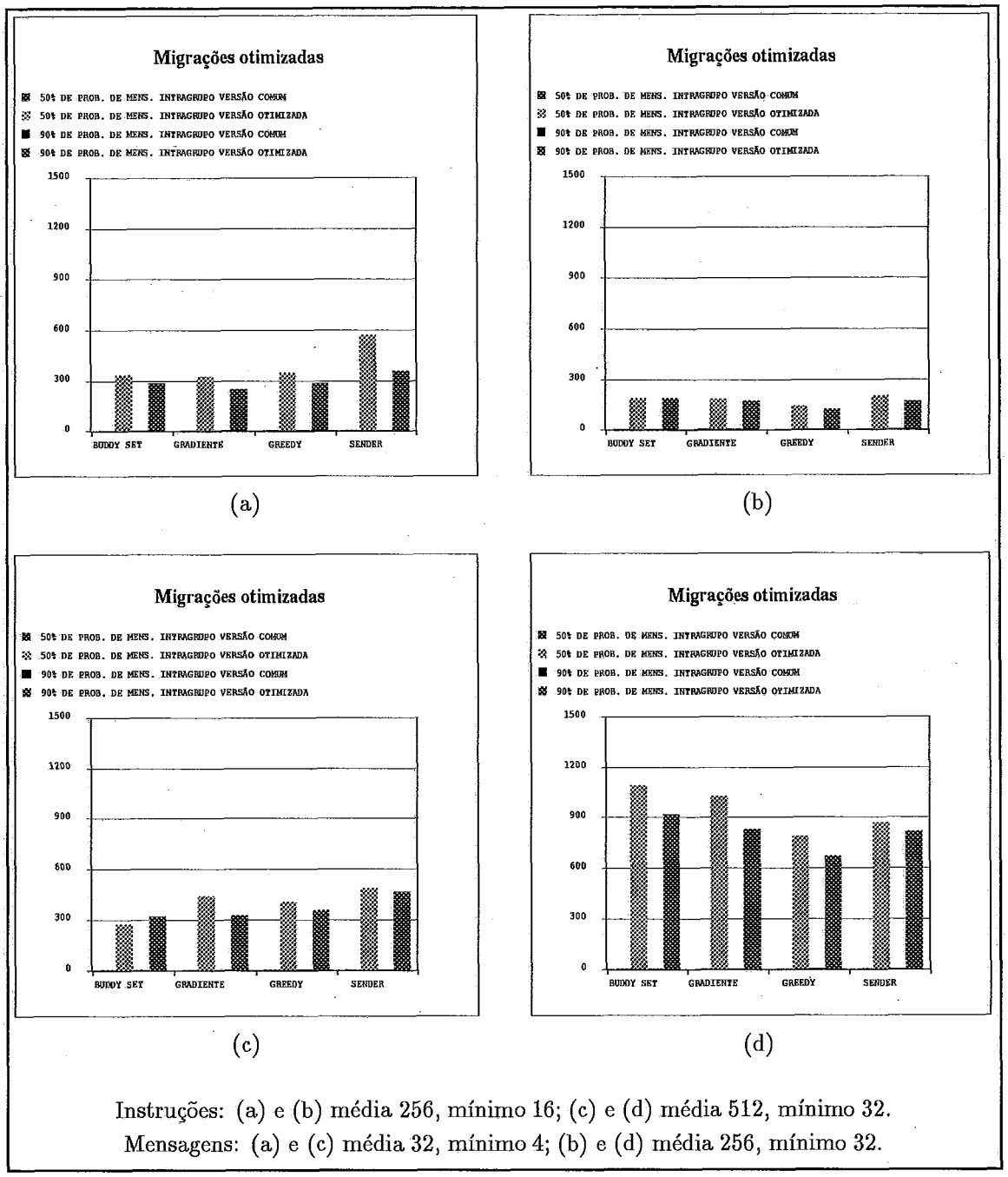
A taxa de utilização dos canais de comunicação pela aplicação indica uma maior eficiência do escalonamento. As versões otimizadas apresentam índices mais elevados para esta taxa, o que é mais um indicador da sua superioridade. Entre as versões otimizadas, os resultados relativos às aplicações com 50% de privilégio de grupo são maiores, o que pode ser atribuído a trajetos maiores percorridos pelas mensagens.



Instruções: (a) e (b) média 256, mínimo 16; (c) e (d) média 512, mínimo 32.
Mensagens: (a) e (c) média 32, mínimo 4; (b) e (d) média 256, mínimo 32.

Figura VII.14: Migrações sem critério de escolha

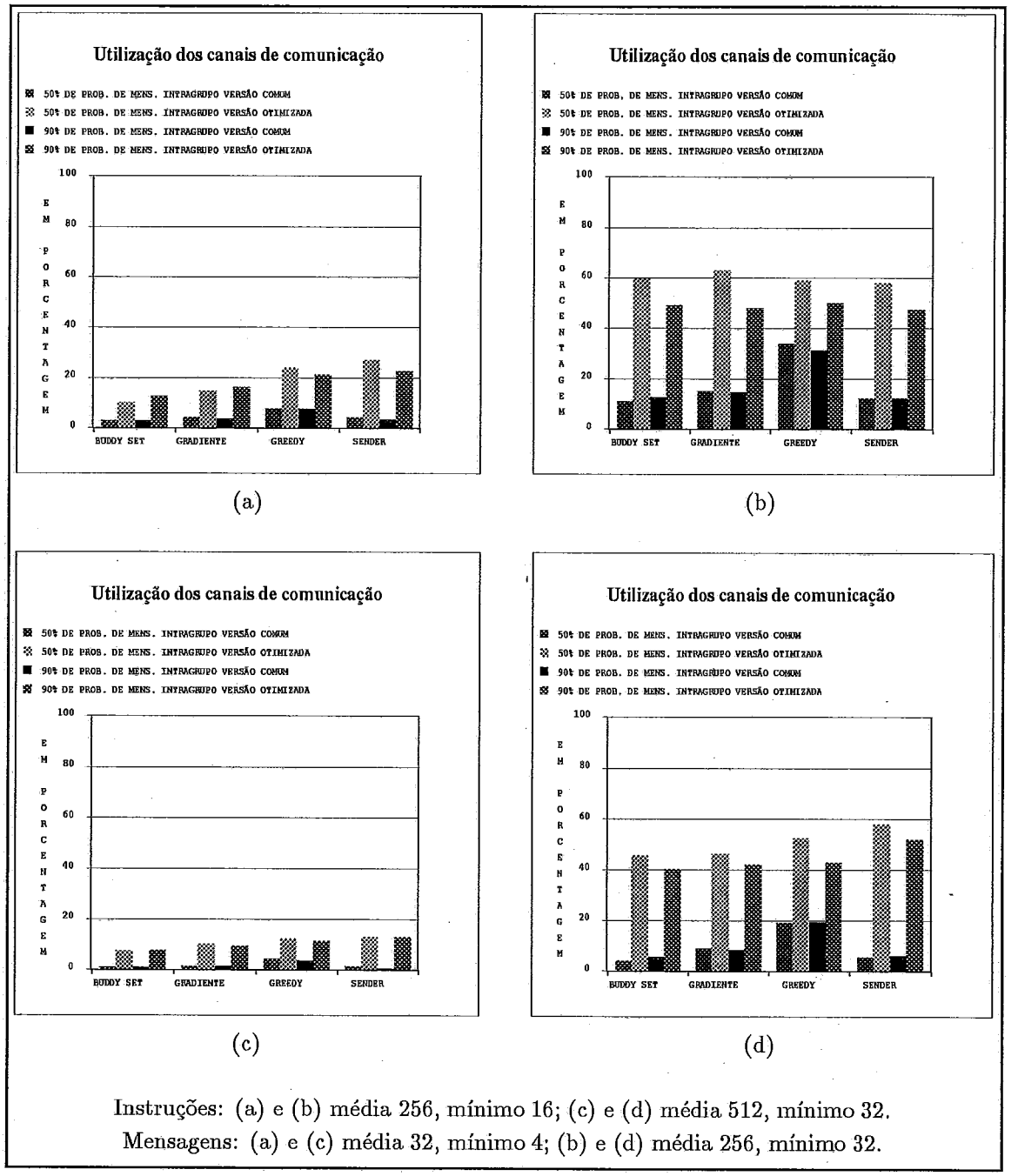
No caso das versões otimizadas, estas são as migrações realizadas quando não foi possível encontrar uma tarefa que se apresentasse como ideal para a migração. No caso das versões comuns são todas as migrações realizadas. Os números mais elevados apresentados pelas versões otimizadas são decorrentes de uma melhor política para a avaliação da carga computacional dos processadores, que considera apenas as tarefas prontas para execução.



Instruções: (a) e (b) média 256, mínimo 16; (c) e (d) média 512, mínimo 32.
Mensagens: (a) e (c) média 32, mínimo 4; (b) e (d) média 256, mínimo 32.

Figura VII.15: Migrações das tarefas mais comunicativas

Estas migrações são as decorrentes de uma escolha mais criteriosa da tarefa a migrar. Os critérios estabelecidos têm por objetivo aproximar as tarefas que se comunicam mais intensamente entre si e minimizar os custos das migrações de tarefas.



Instruções: (a) e (b) média 256, mínimo 16; (c) e (d) média 512, mínimo 32.
Mensagens: (a) e (c) média 32, mínimo 4; (b) e (d) média 256, mínimo 32.

Figura VII.16: Utilização dos canais de comunicação

A utilização dos canais de comunicação indica o aproveitamento da capacidade de comunicação do sistema. Taxas muito altas ou muito baixas indicam uma inadequação do escalonamento ou uma desproporção entre as demandas por comunicação e processamento da aplicação. As versões otimizadas apresentam uma maior taxa, o que pode ser conseqüência de uma maior utilização dos processadores pela aplicação.

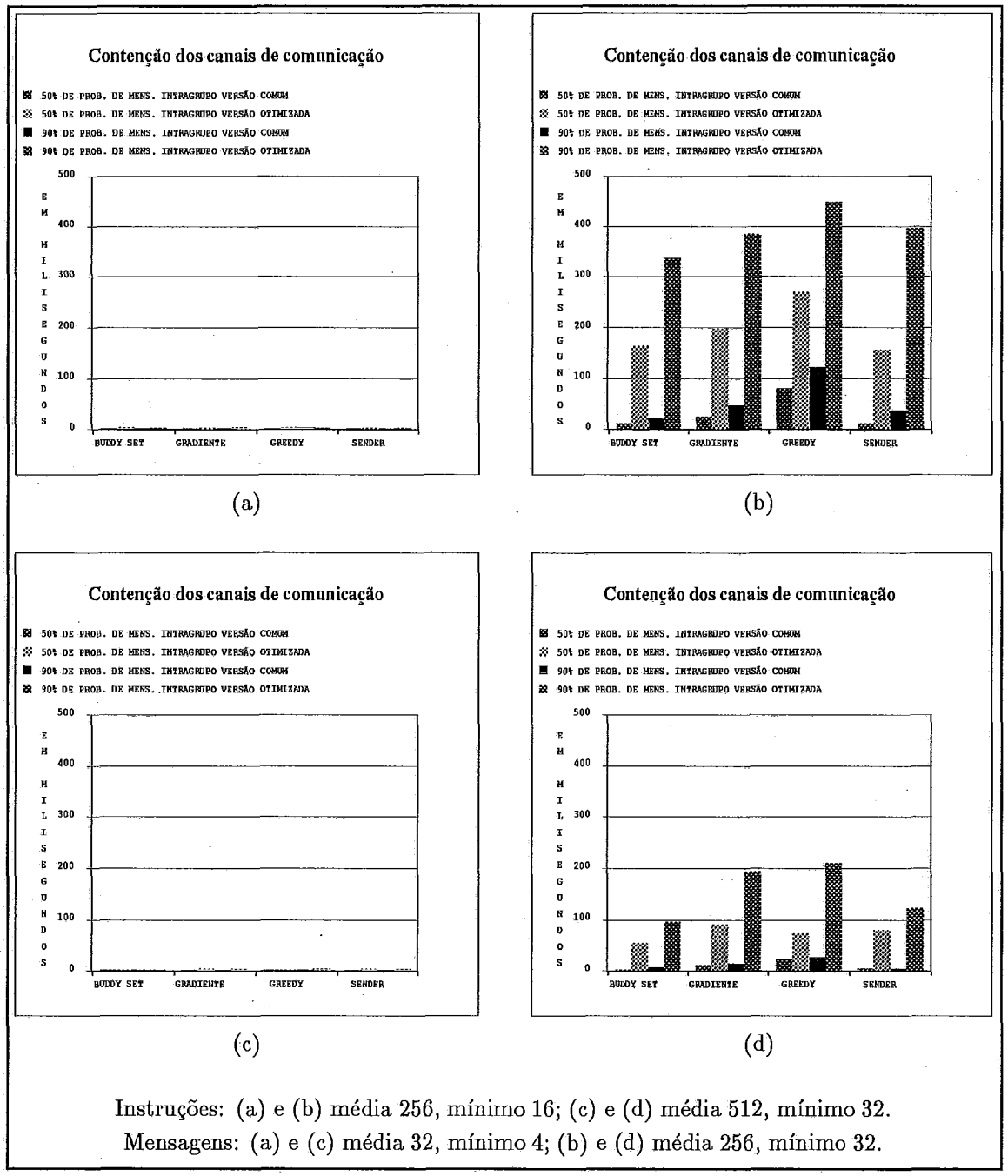


Figura VII.17: Contenção dos canais de comunicação

A contenção dos canais de comunicação é dada pelo somatório dos tempos de espera das mensagens que passaram por um canal. Um índice muito alto de contenção pode indicar a saturação do sistema de comunicação, ou que ele está operando próximo ao seu limite. As versões otimizadas apresentam uma contenção maior, o que pode ser um reflexo da maior densidade de comunicações, decorrente da maior utilização dos processadores pela aplicação nestas versões.

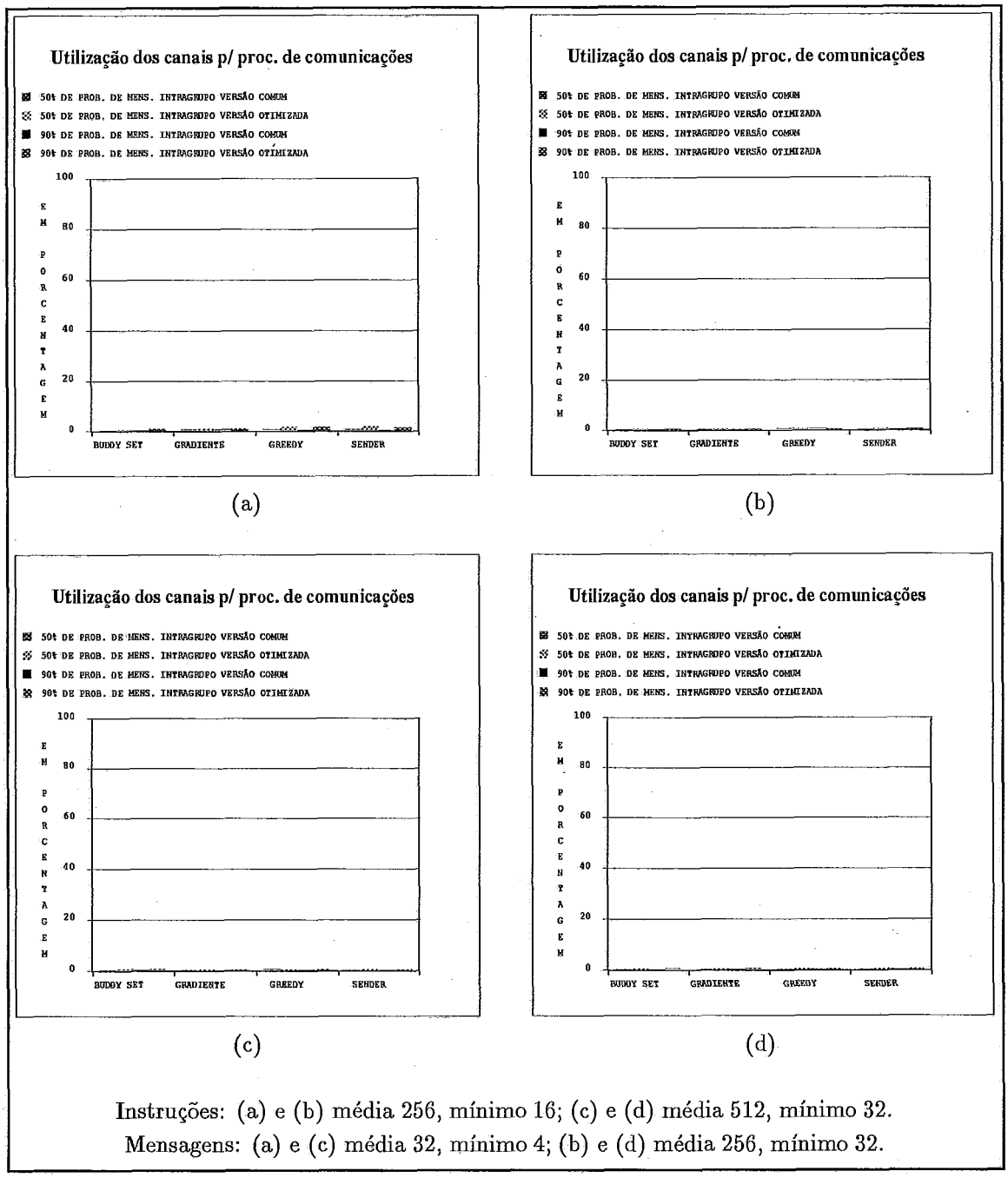
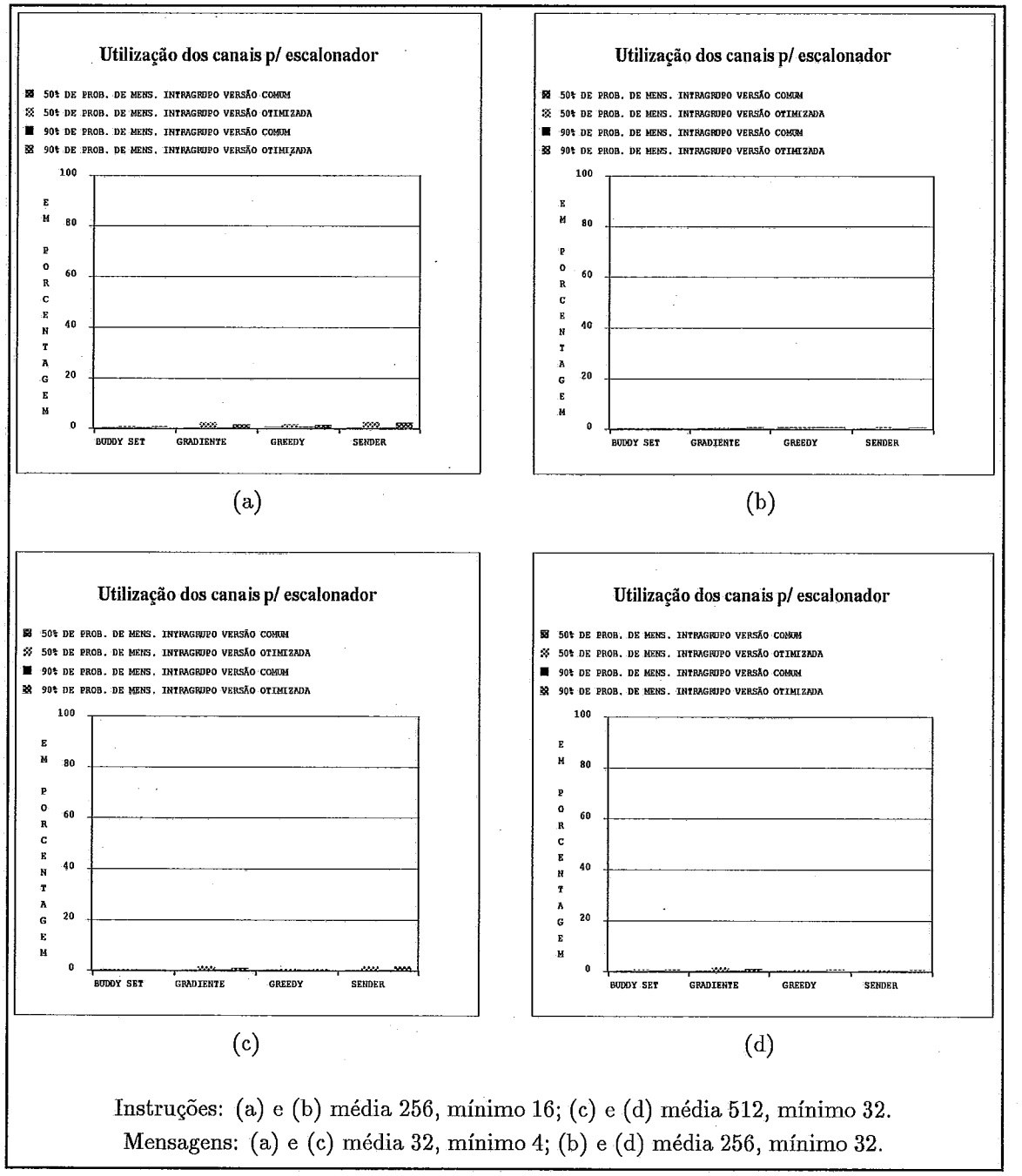


Figura VII.18: Utilização dos canais de comunicação p/ processadores de comunicação

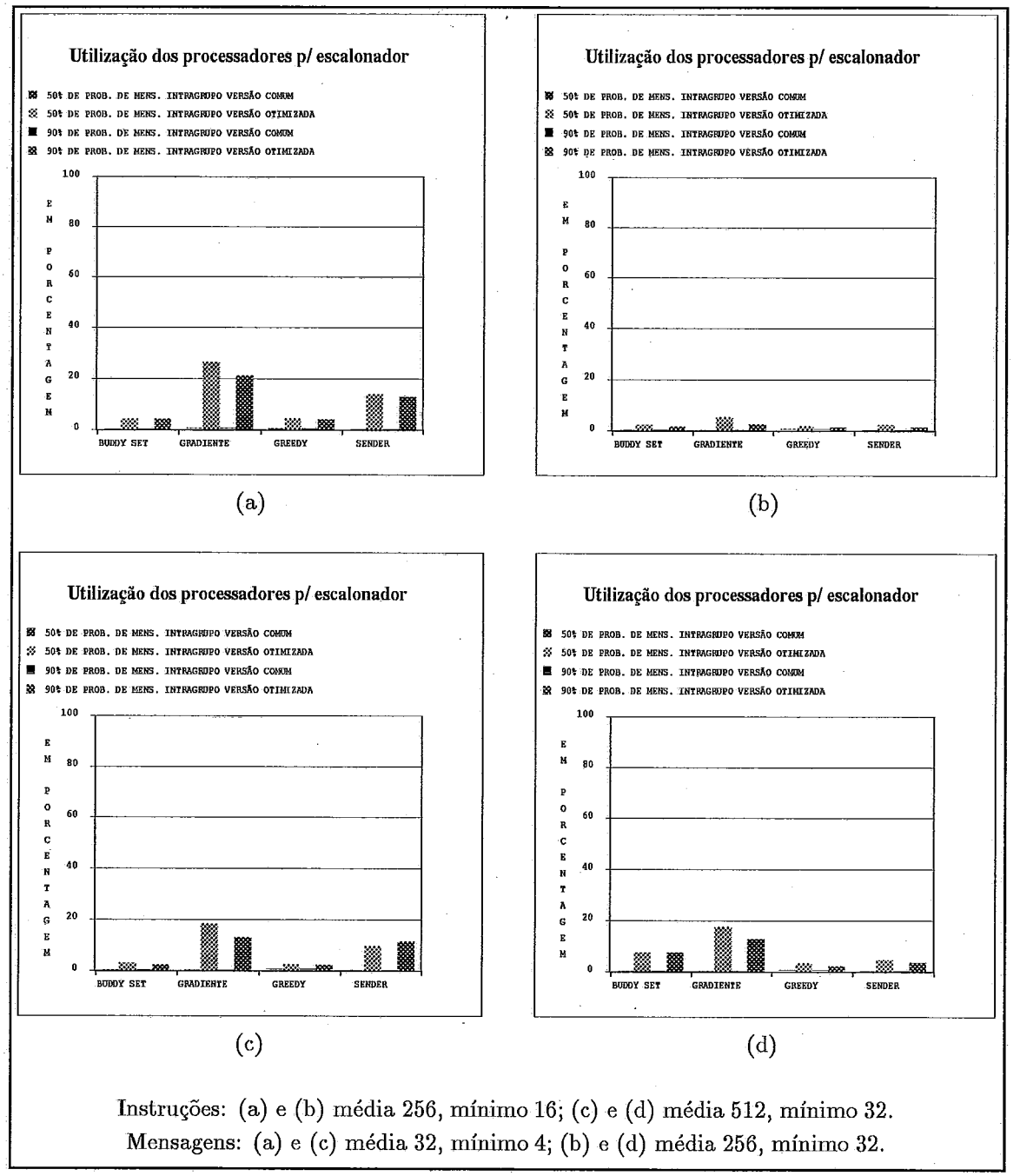
Esta taxa indica a interferência produzida pelo protocolo de comunicação entre tarefas migrantes. As baixas taxas, próximas de zero, são um indício de que o protocolo proposto é muito eficiente.



Instruções: (a) e (b) média 256, mínimo 16; (c) e (d) média 512, mínimo 32.
Mensagens: (a) e (c) média 32, mínimo 4; (b) e (d) média 256, mínimo 32.

Figura VII.19: Utilização dos canais de comunicação pelo escalonador

Esta taxa indica a eficiência dos escalonadores quanto a utilização dos canais de comunicação. Todos os escalonadores em ambas versões apresentam taxas próximas a zero, isto pode ser conseqüência do melhoramento introduzido em todos algoritmos e versões quanto a política de informações.



Instruções: (a) e (b) média 256, mínimo 16; (c) e (d) média 512, mínimo 32.
Mensagens: (a) e (c) média 32, mínimo 4; (b) e (d) média 256, mínimo 32.

Figura VII.20: Utilização dos processadores pelo escalonador

A maior utilização dos processadores pelo escalonador está relacionada a um maior número de ativações do escalonador, ou a uma maior complexidade do escalonador. As versões otimizadas dos escalonadores apresentam taxas de utilização dos processadores maiores. Como a otimização realizada nos algoritmos de escalonamento não aumenta significativamente a complexidade destes algoritmos, as versões otimizadas apresentam um maior número de ativações.

Cada algoritmo foi simulado treze vezes nas suas duas versões, para cada um dos oito tipos de cargas computacionais. Nesta fase de obtenção de resultados foram realizados um total de 832 exercícios de simulação.

Conforme pode ser observado na figura VII.11, referente ao tempo de resposta da aplicação, as versões otimizadas apresentam um desempenho superior ao desempenho das versões comuns. Os reflexos deste melhor desempenho podem ser observados nas figuras VII.12 e VII.13, que exibem respectivamente as taxas de utilização pela aplicação dos processadores e dos canais de comunicação.

O melhor desempenho das versões otimizadas pode ser explicado pela aproximação das tarefas que mais se comunicam entre si e pelo maior número de ativações pertinentes do escalonador global.

As versões otimizadas sempre tentam realizar migrações otimizadas. A otimização tem por objetivo aproximar as tarefas que interagem mais freqüentemente entre si, e minimizar os custos de migração. Ao aproximar as tarefas que mais se comunicam entre si, as versões otimizadas diminuem a distância percorrida pelas mensagens, o que resulta em um menor tempo de espera por mensagens pelas tarefas, assim em um maior número de tarefas prontas e logo provavelmente uma maior taxa de utilização dos processadores pelas aplicações.

O maior número de ativações do escalonador global resulta de dois fatores: uma melhor política de avaliação da carga computacional e o reconhecimento da contenção de um canal de comunicação como um evento relevante para a realização de uma decisão de escalonamento.

De acordo com o seu princípio fundamental de considerar a comunicação entre as tarefas da aplicação em execução, as versões otimizadas distinguem as tarefas prontas das tarefas presentes. A política de avaliação da carga computacional das versões otimizadas consideram como índice da carga computacional em um computador o número de tarefas prontas para executar, ao invés do número de tarefas presentes no computador, que é utilizado nas versões comuns.

Capítulo VIII

Conclusões Finais e Avaliação do Trabalho

Este capítulo apresenta conclusões gerais do trabalho, avalia o trabalho em si e aponta direções para a continuação da pesquisa.

Este trabalho tem uma conclusão principal com relação aos escalonadores dinâmicos distribuídos para sistemas MAD:

- A utilização da capacidade de comunicação de um MAD deve ser considerada nas decisões de escalonamento.

O DDSS além de ser uma ferramenta para a avaliação de escalonadores dinâmicos distribuídos, é na realidade, uma ferramenta mais abrangente. O DDSS pode ser utilizado para a avaliação de diversos aspectos dos multicomputadores. Como por exemplo:

- Protocolos de comunicação para multicomputadores.
- Topologias para a organização dos multicomputadores.
- Adequação da configuração de um multicomputador em relação as características de uma aplicação paralela distribuída.
- Métodos de avaliação da carga computacional.
- Políticas para a gerência de informações para os escalonadores dinâmicos distribuídos.
- Políticas de transferências para os escalonadores dinâmicos distribuídos.
- Utilização de novos eventos para a ativação dos escalonadores dinâmicos distribuídos.

Esta abrangência do DDSS permite a realização uma pesquisa mais completa sobre os multicomputadores.

Bibliografia

[1] *Amorim C. L., Barbosa V. C. e Fernandes E. S. T.;* **Uma introdução à computação paralela e distribuída**; *VI Escola de Computação, 1988.*

[2] *Artsy Y. e Finkel R.;* **Designing a process migration facility**; *IEEE Computer Magazine, 22(9), pp. 47-56, September 1989.*

[3] *Baccelli F. e Liu Z.;* **On the execution of parallel programs on multiprocessor systems – A queuing theory approach**; *Journal of the ACM, Vol 37, No 2, pp. 373-414, April 1990.*

[4] *Badr H. G., Gelernter D. e Podar S.;* **An adaptive communications protocol for network computers**; *Performance Evaluation, Vol 6, No 1, pp. 53-68, March 1986.*

[5] *Barak A. e Shiloh A.;* **A distributed load-balancing policy for a multicomputer**; *Software–Practice and Experience, Vol 15(9), pp. 901-913, September 1985.*

[6] *Barbosa V. C. e Huang H. K.;* **Static task allocation in heterogeneous distributed systems**; *Relatórios Técnicos do Programa de Engenharia de Sistemas e Computação, ES-149/88, Junho 1988.*

[7] *Baumgartner K. M. e Wah B. W.;* **A global load balancing strategy for a distributed computer system**; *IEEE Workshop on Future Trends of Distributed Computing Systems in the 1990s, pp. 93-102, September 1988.*

[8] *Baumgartner K. M. e Wah B. W.;* **GAMMON: A load balancing strategy for local computer systems with multiaccess networks**; *IEEE Transactions on Computers, Vol 38, No 8, pp. 1098-1108, August 1989.*

[9] *Bryant R. M. e Finkel R. A.;* **A stable distributed scheduling algorithm**; *Proceedings of the 2nd International Conference on Distributed Systems, pp. 314-323, 1981.*

[10] *Casavant T. L. e Kuhl J. G.;* **Analysis of three dynamic distributed load-balancing strategies with varing global information requirements;** *IEEE Proceedings of The 7th International Conference on Distributed Systems, pp. 185-192, September 1987.*

[11] *Casavant T. L. e Kuhl J. G.;* **A taxonomy of scheduling in general-purpose distributed computing systems;** *IEEE Transactions on Software Engeneering, Vol 14, No 2, pp. 141-154, February 1988.*

[12] *Chowdhury S.;* **The greedy load sharing algorithm;** *Journal of Parallel and Distributed Computing, Vol 9(1), pp. 93-99, May 1990.*

[13] *Chowkwanyun R. e Hwang K.;* **Hybrid dynamic load balancing for distributed-memory multicomputers;** *IEEE Workshop on Future Trends of Distributed Computing Systems in the 1990s, pp. 391-399, September 1988.*

[14] *Cybenko G.;* **Dynamic load balancing for distributed memory multiprocessors;** *Journal of Parallel and Distributed Computing, Vol 7(2), pp. 279-301, October 1989.*

[15] *Dally W. e Seitz C. L.;* **Deadlock-free message routing in multiprocessor interconnection networks;** *IEEE Transactions on computers, Vol C-36, No 5, pp. 547-553, May 1987.*

[16] *Dally W.;* **Network and processor architecture for message driven computers;** *VLSI and Parallel Computation, capítulo 3, 1989.*

[17] *Dally W.;* **Express cubes: improving the performace of k-ary n-cube interconnection networks;** *IEEE Transactions on computers, Vol 40, No 9, pp. 1016-1023, September 1991.*

[18] *Douglis, F. e J. K. Ousterhout;* **Process Migration in the Sprite Operating System;** *Seventh International Conference on Distributed Computing, pp. 18-25, Sept. 1987.*

[19] *Eager D. L., E. D. Lazowska e J. Zahorjan;* **A comparison of receiver-initiated and sender-initiated adaptive load sharing;** *Performance Evaluation Vol 6(1), pp. 53-68, March 1986.*

[20] *Eager D. L., Lazowska E. D. e Zahorjan J.;* **Adaptive load Sharing in homogeneous distributed systems;** *IEEE Transactions on Software Engeneering, Vol SE-12, pp. 662-675, May 1986.*

[21] *Eager D. L., Lazowska E. D. e Zahorjan J.;* **The limited performance benefits of migrating active processes for load sharing;** *ACM Performance Evalaluation, Vol 16, pp. 63-72, 1989.*

[22] *Efe K. e Groselj B.;* **Minimizing control overheads in adaptive load sharing;** *IEEE 1989, pp. 307-315.*

[23] *Eskicioglu M. R. e Cabrera L. F.;* **Process migration: an annotated bibliography;** *Technical Report RJ 7935 (72918), IBM Almaden Ressearch Center, January 1991.*

[24] *Farber D. J.;* **The distributed computing system;** *Proceedings of the Compcon Spring 73, pp. 31-34, 1973.*

[25] *Ferguson D., Yemini Y. e Nikolaou C.;* **Microeconomic algorithms for load balancing in distributed computer systems;** *IEEE Proceedings of The 8th International Conference on Distributed Systems, pp. 491-499, June 1988.*

[26] *Ferrari D.;* **Computer Systems Performance Evaluation;** *Prentice-Hall, 1978.*

[27] *Ferrari D.;* **Workload characterization for tightly-coupled and loosely-coupled systems;** *ACM Performance Evalaluation Review, Vol 17, #1, pp. 210, May 1989.*

[28] *Fowler, R. J.;* **Decentralized Object Finding Using Forwarding Addresses;** *PhD thesis, University of Washington, Seatle, Washington, December 1985.*

[29] *Fowler, R. J.;* **The Complexity of Using Forwarding Addresses for Decentralized Object Finding;** *Proceedings of the Fifth ACM Symposium on the Principles of Distributed Computation, Calgary, Canada, August 1986.*

[30] *Garey M. R. e Johnson D. S.;* **Computers and Intractability: a guide to the theory of NP-Completeness;** *W. H. Freeman, 1979.*

[31] *Hác A., Jin X.;* **Dynamic load balancing in a distributed system using a descentralized algorithm;** *IEEE Proceedings of The 7th International Conference on Distributed Systems, pp. 170-177, September 1987.*

[32] *Hsu C.-Y. H. e Liu W.-S.;* **Dynamic load balancing algorithms in homogeneous distributed systems;** *IEEE Proceedings of The 6th International Conference on Distributed Systems, pp. 216-223, 1986.*

[33] *Joosen W., Berbers,Verbaeten P.;* **Dynamic load balancing in transputer applications with geometric parallelism;** *North-Holland, Microprocessing and Microprogramming, 30, pp. 77-84, 1990.*

[34] *Joosen W., Verbaeten P. e Leuven K. U.;* **On the use of process migration in distributed systems;** *North-Holland, Microprocessing and Microprogramming, 28(1-5), pp. 49-52, March 1990.*

[35] *Jul, E., H. Levy, N. Hutchinson e A. Black;* **Fine-Grain Mobility in the Emerald System;** *ACM Transactions on Computer Systems, Vol. 6, No. 1, February 1988.*

[36] *Kermani P. e Kleinrock L.;* **Virtual cut-through: a nnew computer communication switchig technnique;** *North-Holland, Computer Networks, Vol 3, No 4, pp. 267-286, September 1979.*

[37] *Krueger P. e Livny M.;* **A comparison of preemptive and non-preemptive load distributing;** *IEEE Proceedings of The 8th International Conference on Distributed Systems, pp. 123-130, June 1988.*

[38] *Kyrimis K. e Alonso R.;* **An experimental comparison of initial placement vs. process migration for load balancing strategies;** *Technical Report CS-TR-199-88, Princeton University, December 1988.*

[39] *Lin F. C. H. e Keller R. M.;* **Gradient model: A demand-driven load balancing scheme;** *IEEE Proceedings of The 6th International Conference on Distributed Systems, pp. 329-336, 1986.*

[40] *Linder D. H e Harden J C.;* **An adaptative and fault tolerant wormhole routing strategy for k-ary n-cubes;** *IEEE Transactions on Computers, Vol 40, No 1, pp. 2-12, January 1991.*

[41] *Livney M. e Melman M.;* **Load balancing in homogeneous broadcast distributed systems;** *Proceedings of ACM Computer Network Performance Symposium, pp. 47-55, 1982.*

[42] *Lu C., Chen A. e Liu J.;* **Protocols for reliable process migration;** *In Proceedings of INFOCOM'87, San Francisco, California, March 1987.*

[43] *Macharia G. M.;* **CLD: A novel approach to dynamic load balancing;** *Microprocessing and Microprogramming 28, pp. 43-48, 1989.*

[44] *Mead C. e CONWAY L.;* **Introduction to VLSI systems;** *Addison-Wesley, 1980.*

[45] *Mirchandaney R., Towsley D. e Stankovic J. A.;* **Adaptive load sharing in heterogeneous distributed systems;** *Journal of Parallel and Distributed Computing, Vol 9(4), pp.331-347, August 1990.*

[46] *Mullender S. J., e P. M. B. Vitany;* **Distributed Match-Making for Processes in Computer Networks;** *Proceedings of the Fourth ACM Symposium on the Principles of Distributed Computation, Minacki, Canada, August 1985.*

[47] *Mullender S. J.;* **Distributed Systems;** *ACM Press Frontier Series 1989.*

[48] *Ni L. M.;* **A distributed load balancing algorithm for point-to-point local computer networks;** *IEEE CompCon Fall, pp. 116-123, 1982.*

[49] *Ni L. M., Xu C.-W. e Gendreau T. B.;* **A distributed drafting algorithm for load balancing;** *IEEE Transactions on Software Engineering SE-11(10), pp. 1153-1161, October 1985.*

[50] *Nicol D. M. e Townsend J. C.;* **Accurate modeling of parallel scientific computations;** *ACM Performance Evalaluation Review Vol 17 #1, pp. 165-170, May 1989.*

[51] *Ousterhout, J. K. et al.;* **The Sprite Network Operating System;** *Computer, February 1988.*

[52] *Popek, G. J. e B. J. Walker;* **The LOCUS Distributed System Architecture;** *Computer Systems Serires, The MIT Press, 1985.*

[53] *Powell, M. L. e B. P. Miller;* **Process Migration in DEMOS/MP;** *Proceedings of the Ninth Symposium on Operating Systems Principles, October 1983.*

[54] *Pulidas S., Towsley D., Stankovic J. A.;* **Imbedding gradient estimators in load balancing algorithms;** *IEEE Proceedings of The 8th International Conference on Distributed Systems, pp. 482-490, June 1988.*

[55] *Ravi T. M. e Jefferson D.;* **A basic protocol for routing messages to migrating processes;** *In proceedings of the International Conference on Parallel Processeing, Vol. II, Software, August 1988.*

[56] *Raynal M.;* **Distributed algorithms and protocols;** *John Wiley & Sons, Wiley series in computing, 1988.*

[57] *Reed D. A.;* **The performance of multimicrocomputer networks supporting dynamic workloads;** *IEEE Transactions on Computers, Vol C-33, No 11, pp. 1045-1048, November 1984.*

[58] *Reed D. A. e Fujimoto R. M.;* **Multicomputer networks: message based parallel processing;** *MIT Press series in scientific computation, 1987.*

[59] *Saad Y. e Schultz M. H.;* **Data communication in hipercubes;** *Journal of Parallel and Distributed Computing, 6(1), pp. 115-135, February 1989.*

[60] *Sauer C. H. e Chandy K. M.;* **Computer systems performance modeling;** *Prentice-Hall Series in Advances in Computing Science and Technology, 1981.*

[61] *Scheuermann P. e Wu G.;* **Broadcasting in point-to-point computer networks;** *In Proceedings of the 1984 International Conference on Parallel Processing, pp. 346-351, August 1984.*

[62] *Scheurich C. e M. Dubois;* **Dynamic Memory Allocation in a Mesh-Connected Multiprocessor;** *Proceedings of the 12th Annual Hawaii International Conference on System Sciences, January 1987.*

[63] *Seitz C. et al;* **Wormhole chip project report;** *Inverno 1985.*

[64] *Shin K. G. e Chang Y.;* **Load sharing in distributed real-time systems with state-change broadcasts;** *IEEE Transactions on Computers, Vol 38, No 8, pp. 1124-1142, August 1989.*

[65] Smith J. M.; **A survey of process migration mechanisms;** *ACM Operating Systems Review, 22(3), pp. 28-40, July 1988.*

[66] *Smith R. G.;* **The contract net protocol: high-level communication and control in a distributed problem solver;** *IEEE Transactions on Computers, Vol C-29, No 12, pp. 1104-1113, December 1980.*

[67] *Soares L. F. G.;* **Modelagem e simulação discreta de sistemas;** *IME-USP VII Escola de Computação, São Paulo 1990.*

[68] *Stankovic J. A. et al;* **A review of current research and critical issues in distributed system software;** *Distributed Processing Technical Commitee Newsletter, Vol 7, No 1, pp. 14-47, March 1985.*

[69] *Stout Q. e Wagar B.;* **Intensive hipercube communication - Prearranged communication in link bound machines;** *Journal of Parallel and Distributed Computing, Vol 10(2), pp. 167-181, 1990.*

[70] *Theimer, M.;* **Preemptable remote execution facilities for loosely-coupled distributed systems;** *Stanford University Technical Report STAN-CS-86-1128, June 1986.*

[71] *Towsley D. e Mirchandaney R.;* **The effect of communication delays on the performance of load balancing policies in distributed systems;** *Computer Performance and Reliability, pp. 213-226, 1988.*

[72] *Wang Y., Morris R. J. T.;* **Load sharing in distributed systems;** *IEEE Transactions on Computers, Vol C-34, No 3, pp. 204-217, March 1985.*

[73] *Weinrib A.;* **Greed is not enough: Adaptive load sharing in large heterogeneous systems;** *IEEE (Parallel) 1988, pp. 0986-0994, 1988.*