Universidade Estadual de Campinas
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

UNICAMP

Aplicação de Modelos Microeconômicos na Alocação de Recursos Computacionais em Ambiente de Processamento Paralelo Virtual Baseado na Internet

Autor: Pedro Wilmer Chávez Chiclayo

Orientador: Prof. Dr. Marco A. Amaral Henriques

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos necessários para a obtenção do título de Mestre em Engenharia Elétrica.

Dissertação de Mestrado

Campinas
Dezembro, 1999

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA – BAE – UNICAMP

C398a

Chávez Chiclayo, Pedro Wilmer
Aplicação de modelos microeconômicos na alocação de recursos computacionais em ambiente de processamento paralelo virtual baseado na internet / Pedro Wilmer Chávez Chiclayo.—Campinas, SP: [s.n.], 1999.

Orientador: Marco A. Amaral Henriques.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Alocação de recursos. 2. Internet (Redes de computação). 3. Algoritmos de computador. 4. Redes de computação – Carga de trabalho. I. Henriques, Marco A. Amaral. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Universidade Estadual de Campinas
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Aplicação de Modelos Microeconômicos na Alocação de Recursos Computacionais em Ambiente de Processamento Paralelo Virtual Baseado na Internet

Autor: Pedro Wilmer Chávez Chiclayo

Banca Examinadora:

Prof. Dr. Marco A. Amaral Henriques
(DCA/FEEC/UNICAMP)

Profa. Dra. Maria Beatriz Felgar de Toledo
(IC/UNICAMP)

Prof. Dr. Mario Jino
(DCA/FEEC/UNICAMP)

Agradecimentos

A Deus por ter me guiado e cuidado para chegar até aqui e por me permitir concluir este trabalho com saúde.

Ao meu pai, por ter me ensinado a lutar e sacrificar-me para atingir meus objetivos.

Ao meu orientador, Marco Aurélio, por ser um orientador preocupado com os problemas de seus orientados, por ter sempre um tempo para discutir aspectos técnicos, por ser paciente na hora de ensinar conceitos que não eram claros. E, principalmente, por ser mais que um mestre, um amigo.

Aos meus irmãos: Jacqueline, Haydee, Cesy, Pablo, Noemi, Maritza, Edwin, Miguel e Elena, que sempre me apoiaram e continuam fazendo-o, porque, apesar da distância, me levam sempre nos pensamentos e pedem a Deus por mim.

À minha noiva e futura esposa Fresia, pela compreensão, carinho e amor. Pelo apoio emocional, sem o qual teria sido difícil esta jornada.

Ao meu colega e amigo Edwin pelas palavras de conforto e otimismo. Por ser o "friend forever" do qual estou orgulhoso.

Ao meu amigo Carlos Solano que dividiu a infância e a juventude comigo, por sempre estar presente nos momentos importantes de minha vida. Aprendemos juntos a viver.

Aos meus grandes amigos Cosme, Vitaly, Guillermo, Luis Alberto, Carlos, Alex, Miguel, Pedro, Herbert, Armando, Bogar, Omar por terem cuidado de mim e pelos conselhos sempre tão oportunos.

Às minhas amigas Rocio, Mercedes, Rossie, Grissel, Anelli, Lizet, Flor, Yndra, pelos conselhos sempre tão bem-vindos e pelas palavras de apoio nos momentos difíceis.

Aos meus amigos de trabalho Eduardo, Juan Carlos, Liana, Saulo, Míriam, Fabiano e Arnaldo que contribuíram com sugestões e idéias sempre muito interessantes para este trabalho.

À minha família e a todas as pessoas amigas que tanto ajudaram no decorrer da minha vida e cujos nomes não esqueço, mas que são tantas que não caberiam nesta página.

À agência CAPES e ao FAEP/UNICAMP por financiar este trabalho de pesquisa e dar-me a possibilidade de aperfeiçoar meus conhecimentos.

Finalmente quero agradecer à pessoa mais importante da minha vida: minha mãe. Por ter me dado vida e me ensinado a vivê-la. Obrigado mãe por esse amor infinito que você tem; eu nada poderia ter feito sem ele. Dedico-lhe este trabalho, porque pensar em você me ajudou a terminá-lo.

Resumo

Um conjunto de computadores que estão conectados pela Internet pode ser visto como um Computador Massivamente Paralelo Virtual (MPVC) com memória distribuída. Há sistemas de processamento que se propõem a explorar estes recursos como um computador virtual que possa executar aplicações paralelas compostas por uma grande quantidade de tarefas. A execução destas aplicações introduz problemas de alocação de recursos e balanceamento da carga, isto é, a distribuição eficiente dos computadores do MPVC entre as aplicações.

Este trabalho propõe o algoritmo Resource Allocation Algorithm using Prices (RAAP) baseado em modelos microeconômicos que auxilia na alocação de recursos da Internet às aplicações submetidas nestes computadores virtuais. Este algoritmo faz uso do conceito de preço dos recursos para permitir uma alocação mais eficiente e implementa um mercado onde os recursos são ofertados e consumidos. São mostrados os resultados da simulação de um mercado e é feita uma análise dos parâmetros que influenciam no sucesso da alocação de recursos computacionais em um mercado microeconômico. Os resultados obtidos mostram a viabilidade e o potencial de se usar conceitos de economia na solução de problemas de alocação eficiente de recursos computacionais em sistemas heterogêneos como um MPVC baseado na Internet.

Abstract

A set of computers connected by the Internet can be considered as a Massivelly Parallel Virtual Computer (MPVC) with distributed memory. There are systems which propose the use of these resources as a virtual computer that can execute parallel applications consisting of a large number of tasks. The execution of such applications introduces problems of resource allocation and load balancing, i.e., how to distribute the MPVC computers among the applications in a more efficient way.

This work proposes the Resource Allocation Algorithm using Prices (RAAP) based on an economic approach to help the resource allocation process needed by applications submitted to an MPVC in the Internet. This algorithm uses the concept of prices to allow a more efficient allocation, and implements a market where the resources are offered and consumed. The results of a market simulation are show, as well as an analysis of the parameters that influence the allocation of computacional resources in a microeconomic market. The results obtained show the potential and viability of using concepts from economics to solve problems of efficient allocation of computational resources in heterogeneous systems as an MPVC based on the Internet.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Visão Geral

O crescimento acelerado que a Internet está tendo faz desta rede uma grande coleção de recursos de computação e comunicação. Segundo as pesquisas realizadas por MIDS (Matrix and Information Directory Service, Inc) [MIDS99], o número de computadores que podem usar os serviços da Internet ultrapassou os 16 milhões em 1997. Nesta mesma pesquisa se indica que este número está dobrando a cada ano desde 1986 e vai continuar crescendo nos próximos anos. Fox e Furmansky [FF97] estimaram que no ano 2007 a performance conjunta dos computadores conectados pela Internet chegará aos Exaops (10^{18} operações por segundo), enquanto os computadores mais avançados estarão na ordem dos Petaops (10^{15} operações por segundo). Baseado nestes dados vislumbra-se a possibilidade de se usar este conjunto de computadores conectados pela Internet para a resolução de problemas que requeiram um grande poder computacional. Este conjunto de computadores pode ser visto como um Computador Massivamente Paralelo Virtual (MPVC[1]) com memória distribuída [EH98]. Um MPVC tem diferenças marcantes com respeito a computadores paralelos reais como: processadores independentes e heterogêneos, distribuição geográfica ampla, alta latência de comunicação entre processadores, topologia de interconexão irregular e disponibilidade imprevisível dos computadores. Por sua parte, os computadores paralelos reais têm processadores homogêneos normalmente disponíveis, topologias de interconexão bem definidas e enlaces de comunicação dedicados e de alto desempenho.

[1]Massivelly Parallel Virtual Computer

Mesmo com algumas possíveis desvantagens frente aos sistemas paralelos reais, MPVCs se propõem a tirar proveito do grande poder de computação agregado dos computadores disponíveis na Internet. Além disso eles têm um baixo custo quando comparados a computadores paralelos reais. Hoje há um conjunto de sistemas [LE96, PO97, CO96, EH98] que estão focados na exploração das vantagens deste grande poder computacional, procurando oferecer uma alternativa de baixo custo, que seja simples de usar, que dê a segurança que as aplicações requerem e que seja acessível para qualquer pessoa que tenha um problema computacional paralelizável.

Na execução de aplicações, com uma grande quantidade de unidades de trabalho ou tarefas, uma das preocupações é a alocação dos recursos e o balanceamento da carga, isto é, distribuição eficiente dos computadores do MPVC entre as tarefas das aplicações. A alocação de recursos em sistemas massivamente paralelos baseados na Internet é complexa e ocorre geralmente em ambientes dinâmicos, isto é, os computadores entram e saem do sistema de maneira imprevisível e os serviços demandados pelas tarefas e suas interações também não são previsíveis.

Princípios econômicos têm sido usados em ciências da computação com sucesso, especialmente a microeconomia. A idéia de usar abordagens microeconômicas para solucionar problemas não econômicos e, em particular, problemas de alocação de recursos computacionais não é nova. Ela surgiu em 1968 em um artigo de Sutherland [IES68]. Mas quais são as vantagens de se usar princípios econômicos em sistemas computacionais? Esta parece ser a primeira pergunta a surgir quando introduzimos este tema. A teoria econômica tem sido usada em ciência da computação para solucionar alguns problemas de alocação de recursos [IES68, WM93, FE96, PO97]. Tais problemas são complexos pelo fato de se precisar alocar recursos entre componentes com grande interação mútua. Estes componentes são autônomos e geralmente têm interesses conflitantes. Este problema é de natureza distribuída e para a solução precisa-se de comunicação para troca de informações. A alocação de recursos pode ser formulada como um problema econômico, no qual vários agentes estão competindo pelos recursos disponíveis e fazendo uso de princípios e mecanismos econômicos. Em economia, esta maneira de interação é chamada de mercado. Os mercados têm mecanismos muito bem conhecidos para solucionar este tipo de competição e seus resultados parecem ser bons, apesar de estarem sujeitos a certas suposições que às vezes não são factíveis.

Se um computador pode formar parte do MPVC, então também pode abandoná-lo

quando quiser ou quando falhar, problema que os algoritmos de alocação de recursos devem tratar. Os computadores que formam parte destes sistemas baseados na Internet não são propriedade dos sistemas, mas sim de pessoas ou instituições que precisam de um estímulo ou recompensa para manter a sua participação.

Com exceção dos sistemas JoiN [EH98, EM99] e PopCorn [PO97], a grande maioria dos MPVCs não prevêem mecanismos de recompensa aos proprietários dos computadores que são colocados à disposição do computador virtual. Tais mecanismos são interessantes não só porque servem de incentivo à participação de um grande número de computadores, mas também porque fornecem dados que podem ser utilizados na alocação de recursos e distribuição de carga pelo sistema. Esta preocupação é importante, já que a não existência da recompensa reduz sensivelmente o interesse em participar, o que é comprovado pelo fato que até agora nenhum dos sistemas existentes ter conseguido crescer até reunir milhares de computadores, apesar de alguns já terem alcançado a maturidade. Um dos sistemas com maior sucesso até o momento é o Globus que realizou experimentos com 330 computadores e 3600 processadores[FK98].

No projeto PopCorn a recompensa é baseada em intercâmbio de serviços, ou seja, o proprietário de um computador pode colaborar com o sistema enquanto executa algum jogo do Servidor Web do projeto, ou enquanto armazena créditos (chamados de PopCoins) como pagamento que poderão ser usados posteriormente. A alocação de recursos em PopCorn é feita procurando pelo primeiro recurso disponível no mercado e que pode ser pago pela aplicação, o que não garante o balanceamento de carga no sistema. Além disso, este sistema não provê nenhum mecanismo que o torne escalável a um grande número de computadores.

Na abordagem proposta por Ferguson *et. al.* [FE96] é usado o conceito de preços para a alocação em sistemas distribuídos. A alocação é feita através de leilões, com ofertas submetidas diretamente ao recurso (processador ou enlace de comunicação). Nesta abordagem, cada computador mantém informação sobre todos os outros participantes, o que não é viável para sistemas massivamente paralelos devido à quantidade elevada de computadores no sistema e de mensagens necessárias para manter atualizadas as informações.

Nesta dissertação é proposto o algoritmo RAAP (Resource Allocation Algorithm using Prices) para a alocação de recursos às aplicações submetidas em computadores paralelos virtuais. RAAP também pode ser usado para calcular uma recompensa para os proprietários dos computadores que coloquem os mesmos à disposição do sistema virtual.

O algoritmo usa o conceito de preços como em [FE96], mas procura evitar suas deficiências de escalabilidade. Isto é feito dividindo os computadores participantes em grupos como faz o MPVC JoiN [EH98, EM99] , com um processo coordenador por grupo, e mantendo tabelas de preços dos computadores do grupo nos coordenadores. Além de minimizar as mensagens trafegadas pela rede para a alocação, outra característica deste algoritmo é que cada coordenador de grupo conhece endereços de coordenadores de outros grupos que podem ser contactados para completar uma alocação, caso o coordenador não tenha todos os recursos solicitados. Os grupos devem estar conectados uns com outros usando alguma topologia lógica de conexão. Estar conectado logicamente um com outro significa que um conhece o endereço do outro. Não há uma topologia particular de conexão, mas o ótimo seria que cada grupo estivesse conectado com a menor quantidade possível de outros grupos (para simplificar a alocação) e que, ainda assim, a topologia usada permitisse trocar informações entre todos os grupos.

Outra característica que distingue o RAAP é a possibilidade de se indicar a prioridade da aplicação para a alocação.

Em RAAP é implementado um mercado de recursos onde dois agentes (fornecedores e consumidores) negociam tempos de uso e preços de processadores. Este mercado ainda implementa um esquema de valorização e desvalorização dos recursos pela oferta e demanda como acontece na economia real.

A moeda da economia virtual do RAAP é chamada de crédito e pode ser transferida de contas das aplicações até contas dos computadores, quando um contrato para execução é combinado. Este mecanismo de transferência de créditos serve para fazer a recompensa posterior e está baseado na quantidade de trabalho efetivo feito. As aplicações possuem uma quantidade de dinheiro virtual medida em créditos, que serve para alugar os recursos para a execução de suas tarefas filhas.

Um esquema de preços nos computadores simplifica o controle da quantidade e qualidade de serviço prestado pelos mesmos, servindo o dinheiro virtual ganho por cada um como parâmetro para a recompensa devida a cada proprietário. Os créditos ganhos poderão ser enviados aos computadores por correio eletrônico, para que eles façam um controle próprio. Estes créditos poderiam ser convertidos em dinheiro virtual [CH92], em moeda eletrônica como a descrita em [DV98] ou em bilhetes contendo números para algum tipo de sorteio de bens ou serviços [GEOT]. O envio destes créditos deve ser feito por mensagem de correio

eletrônico assinada digitalmente para garantir sua integridade e autenticidade.

A alocação econômica pode ser usada como base para futuras pesquisas em outras áreas relativas aos sistemas de processamento massivamente paralelo baseados na Internet, como por exemplo, acesso ao sistema de arquivos quando há muitas aplicações tentando ter acesso ao mesmo tempo, e obtenção de informações do sistema para monitoramento do MPVC, dentre outras.

1.2 Objetivos da dissertação

Na alocação de recursos em sistemas locais, o objetivo principal dos algoritmos é alocar os recursos às aplicações de maneira a fazer com que estas tenham a execução mais rápida e segura possível, fazendo um uso eficiente dos recursos existentes. O balanceamento da carga está relacionado diretamente com a alocação. Em sistemas paralelos baseados na Internet o objetivo da alocação continua sendo o mesmo, mas dando maior ênfase à disponibilidade dos computadores participantes, por estes sistemas estarem formados por computadores que não têm disponibilidade previsível, e pelas características físicas destes sistemas paralelos virtuais que são bastante distintas daquelas dos computadores paralelos reais.

Os principais objetivos das idéias expostas nesta dissertação são:

- oferecer um algoritmo escalável para alocação de recursos e balanceamento de carga em uma plataforma de computação massivamente paralela virtual;
- servir como base para o cálculo da recompensa dos proprietários dos computadores participantes, a fim de manter e aumentar o interesse em participar;
- usar conceitos microeconômicos para fazer uma alocação mais eficiente e dinâmica.

1.3 Organização da dissertação

A dissertação esta organizada como segue.

No Capítulo 2 são mostradas as motivações para este trabalho, assim como algumas abordagens prévias sobre o uso de economia em sistemas computacionais.

O Capítulo 3 explica os mecanismos de teste e classificação dos computadores participantes, que são feitos usando um conjunto de testes de desempenho (benchmarks).

O Capítulo 4 apresenta o algoritmo de alocação de recursos usando preços (Resource Allocation Algorithm using Prices - RAAP) assim como sua implementação e testes.

O Capítulo 5 apresenta a análise dos resultados obtidos pelo algoritmo desenvolvido e faz algumas recomendações sobre seu uso.

Finalmente, o Capítulo 6 apresenta as conclusões e recomendações finais, assim como um resumo das principais aplicações e trabalhos futuros de pesquisa que podem ser realizados nesta área, usando as contribuições desta dissertação.

Capítulo 2

Alocação de Recursos e Balanceamento de carga

2.1 Introdução

A idéia de usar economia em ciências da computação não é completamente nova. Princípios econômicos têm sido usados implicitamente em Inteligência Artificial por muitos anos. Recentemente a ciência da economia está sendo usada diretamente como um guia para pesquisas na área da ciência da computação.

Neste Capítulo abordaremos o assunto de como a economia tem sido usada em ciência da computação e as motivações que nos levaram a desenvolver uma nova maneira de alocação de recursos computacionais, assim como revisaremos alguns trabalhos afins.

2.2 Motivações para a alocação de recursos

Atualmente existem milhões de computadores conectados à Internet e, em muitos momentos, uma grande quantidade destes permanecem ociosos. Uma idéia que surge naturalmente é usar estes computadores como um grande computador paralelo para executar aplicações que precisem de muito poder computacional. Idéias similares no contexto de redes locais são conhecidas devido ao trabalho sobre redes de estações de trabalho [ACP95] e ao desenvolvi-

mento de software para configurar, programar e administrar este tipo de redes, as quais têm dado resultados satisfatórios em desempenho [GBD+94, SOH+96, JBA, DL, SUN97b, NL]. Apesar destes sistemas parecerem muito promissores, são muitas as dificuldades encontradas na sua implementação e funcionamento. Tais dificuldades são maiores para o caso de computação global na Internet, começando pela portabilidade do código, segurança, heterogeneidade de plataformas, e problemas de coordenação.

A Internet ainda apresenta muitas dificuldades técnicas para a implementação de um sistema de processamento paralelo sobre ela. Os principais problemas a enfrentar são a pequena largura de faixa disponível, a alta latência e a baixa confiabilidade. Por outro lado, há um grande atrativo que é o elevado número de processadores.

Uma outra dificuldade é o fato de que os possíveis computadores participantes de um computador massivamente paralelo virtual baseado na Internet (MPVC) são de pessoas ou instituições que normalmente não têm interesse em colocar seus computadores formando parte do sistema. É possível compreender então que é necessário um mecanismo de recompensa para que os proprietários dos computadores sejam motivados a colocar estes à disposição, o que nos leva a pensar em mecanismos de pagamento pelo uso dos mesmos.

No desenvolvimento deste tipo de sistema encontramos uma área que precisa de atenção especial: a área de alocação de recursos e balanceamento da carga, já que as aplicações podem exigir grandes quantidades de processamento, sendo divididas em tarefas que são executadas concorrentemente nos computadores espalhados pela Internet.

No balanceamento da carga, o sucesso depende dos algoritmos que são influenciados por vários fatores tais como o desempenho de cada computador, a velocidade de suas conexões, a carga atual de trabalho de cada computador, os recursos necessários para as aplicações executarem, dentre outros. Com estas considerações concluímos que devemos ter alguns critérios de classificação dos computadores participantes, o que nos leva a pensar num método de classificação baseado no poder computacional dos computadores que se declaram disponíveis para o sistema. Tal classificação pode ser feita por meio do cálculo de um valor associado a testes de desempenho que serão discutidos no Capítulo 3.

2.3 Teoria Econômica em Ciência da Computação

2.3.1 Alocação de recursos

Um dos principais tópicos em economia é a alocação de recursos. Muitas pesquisas têm sido focadas na linha de como os recursos são alocados na sociedade humana e como estes deveriam ser alocados. Especificamente os economistas estão preocupados com a alocação de recursos *escassos.* Um recurso é *escasso* se a demanda por ele excede a oferta. Exemplos de recursos escassos estão presentes virtualmente em todos os lugares. Um exemplo de recurso que não é escasso é o ar, já que em condições normais a oferta excede a demanda e por isso ele ainda é grátis. Em um computador a maioria de recursos são escassos, e esta escassez não pode ser solucionada aumentando a quantidade de recursos disponíveis. Lembremos que programadores que antes não precisavam de mais de 16k de memória para programar nos computadores antigos, hoje têm problemas para fazê-lo com computadores que passam de 32 MB.

Os economistas têm estabelecido uma boa solução para o problema de alocação de recursos escassos, fazendo uso de um mecanismo de negociação chamado mercado. Várias pesquisas mostram que os mercados podem alcançar alocações de recursos eficientes sob muitas circunstâncias com pouca ou nenhuma centralização. Cada mercado pode cuidar de si próprio, e o equilíbrio entre a oferta e a demanda será ótimo[1] ou muito proximo de ótimo. Os mercados também têm vantagens significativas sobre esquemas de alocação centralizados, alguns dos quais são mencionados nas seções a seguir.

2.3.2 Eficiência

A teoria econômica também fornece uma definição de eficiência, que em alguns casos é mais exata ou mais útil que seu significado comum em ciência da computação. Em ciência da computação o termo "eficiente" usualmente é usado para fazer comparações de rapidez de um método com respeito a outros possíveis. Esto é o significado padrão de um algoritmo eficiente em ciência da computação. Já que eficiente é definido em relação a outras opções, isto em geral tem um significado não preciso por que podem existir opções mais rápidas

[1]No sentido que esta alocação de recursos não pode ser melhorada por nenhum outro alocador de recursos.

que ainda não são conhecidas.

Os economistas generalizam o significado de eficiência para incluir não só a velocidade (o eficiente uso do tempo), com também o eficiente uso dos recursos. Eficiência econômica está relacionada à negociação que existe entre os consumidores de recursos, à competição no consumo, e as condições que implicam que todos os recursos estejam sendo usados simultaneamente para máximo benefício.

2.3.3 Tomada de decisões descentralizada

Os mercados estão baseados na tomada de decisões descentralizada. Cada participante precisa apenas de informação local com a qual possa tomar decisões a respeito de preferências, e eventualmente a respeito da valorização de seus serviços e bens. Em muitos casos pode ser muito mais fácil formular soluções a partir de decisões locais que globais.

Em qualquer situação na qual exista uma grande quantidade de informação e uma taxa muito alta de geração desta, pode ser impossível uma tomada de decisões centralizada. No caso de que exista informação incerta ou conflitante, os participantes descentralizados podem ter uma melhor opção para fazer uma escolha razoável, já que presumivelmente eles têm menos incertezas e poucas contradições para tratar.

A descentralização também pode permitir a especialização na solução de problemas locais. Indivíduos que têm as mesmas necessidades e objetivos podem desenvolver vias mais eficientes para se comunicar e trocar conhecimentos que são específicos aos seus negócios. Um tomador de decisões centralizado deve conhecer todas as "linguagens" para processar a informação disponível.

A descentralização não pode ser vista como uma rota garantida ao sucesso. Em muitos casos os tomadores de decisões centralizados podem operar mais rápido que os descentralizados e tomar melhores decisões. Isto é verdade especialmente quando as soluções dos problemas são bem conhecidas e o sistema está servindo mais como uma hierarquia de comandos que como um tomador de decisões.

2.3.4 Ajuste dinâmico

Ajuste dinâmico é a caraterística do mercado que denota a habilidade para ajustar-se rapidamente a mudanças repentinas e imprevistas. Cada participante precisa apenas ajustar seus próprios valores baseados nas mudanças e o equilíbrio vai refletir-se na nova alocação de recursos. Nenhuma autoridade é necessária para determinar uma nova solução; o mercado fornece isto automaticamente.

Isto é uma vantagem significativa em sistemas de computadores. O ambiente pode estar mudando constantemente e em geral não vai ser possível predizer o que acontecerá no futuro. São desejáveis ajustes rápidos para estas mudanças.

Um efeito não desejável é que os mercados são geralmente sistemas caóticos no sentido que variações muito pequenas nos parâmetros podem resultar em mudanças muito grandes no curso futuro do mercado. Os ajustes dinâmicos em geral usam esta propriedade, mas não podem tornar muito difícil a análise da conduta do sistema, ou predição de como este atuará no futuro.

2.3.5 Sensibilidade à demanda

Os mercados têm habilidade para produzir as coisas que são demandadas, mesmo que eles não tenham sido criados fazendo certas suposições. Isto permite que sistemas de computadores baseados em leis de mercado possam tomar decisões surpreendentes e criar soluções simples porque a demanda dos participantes evoluiu no tempo.

Geralmente, é possível alcançar o comportamento de sensibilidade à demanda dos mercados em um sistema de computadores devidamente programado.

2.4 Trabalhos afins

Na literatura encontramos vários trabalhos correlatos, mas que não satisfazem as principais características necessárias para a alocação de recursos em sistemas paralelos baseados na Internet.

A alocação de recursos é complexa por diversos fatores, dentre os quais destacamos os seguintes.

1. *Alocação simultânea*: as aplições paralelas precisam de uma grande quantidade de recursos, o que deve ser fornecido pelo sistema, já que o desempenho destas está determinado pela alocação simultânea de recursos às suas tarefas.

2. *Segurança e propriedade*: a propriedade dos recursos na Internet é de diferentes partes, e cada uma tem diferentes exigências de segurança para que seus computadores participem do sistema.

3. *Dinâmica da Internet*: em uma rede como a Internet, o conjunto de máquinas disponíveis está mudando continuamente; assim, a disponibilidade dos computadores na rede não é previsível.

Nesta situação os tradicionais esquemas de alocação de recursos não são efetivos. Os esquemas tradicionais procuram otimizar algumas medidas de desempenho (tempo de resposta, throughput) o que é feito na maioria das vezes com um algoritmo centralizado e com informação completa sobre o sistema, ou com um algoritmo descentralizado baseado em consenso. Mas os fatores que dificultam a alocação de recursos mencionados anteriormente tornam as aproximações tradicionais inadequadas e impraticáveis, já que informação completa sobre o sistema para algoritmos centralizados não é fácil de se obter e de se manter atualizada em sistemas grandes. Além disso o consenso só pode ser obtido em sistemas pequenos ou médios, mas não em sistemas baseados na Internet onde o número de participantes é potencialmente grande.

2.4.1 Projeto PopCorn

PopCorn [PO97] (The Hebrew University of Jerusalem) é um sistema para computação global sobre a Internet. Este sistema fornece funções básicas para que qualquer programador possa usar a Internet como um computador virtual, o qual é implementado usando todos os processadores sobre a Internet que desejam participar do sistema.

O sistema é implementado usando Java [JSM, DD] como linguagem de programação e applets Java para alcançar uma participação de processadores remotos em grande escala,

pela facilidade da execução dos applets em qualquer browser que implemente uma máquina virtual Java (JVM) [JVM].

Alocação de recursos e incentivo à participação em PopCorn

PopCorn oferece uma interface de programação baseada em *computelets*, que são componentes independentes de um programa paralelo e que têm um alto nível de abstração. Os *computelets* têm sua própria função principal de execução, além dos dados que precisam para executar. Um *computelet* também tem o preço oferecido para sua execução, código para manipulação e verificação das respostas obtidas, e manipuladores de eventos para os resultados [SL].

Cada *computelet* é dotado de um orçamento. POPCORN envia automaticamente estes orçamentos para um mercado (escolhido pelo programador com antecedência) que, por sua vez, reenvia estes para um vendedor de tempo de processador (um computador que é parte do POPCORN,) que executa o *computelet* e retorna os resultados. A negociação entre compradores e vendedores é dinâmica e é feita de acordo com mecanismos econômicos, que resultam no pagamento de recursos do comprador para o vendedor.

O mecanismo de incentivo à participação está baseado na boa vontade das pessoas, doações de tempo de CPU, pagamentos diretos, intercâmbio de informação ou de serviços (execução futura de aplicações do dono do computador no ambiente PopCorn ou uso de jogos de computador enquanto o processador está sendo usado no mercado). Alguns destes mecanismos não parecem ser adequados, já que colocar um computador disponível para outras pessoas pode não recompensar, e a permissão para o uso de jogos do site só alcança um público restrito.

Um programa paralelo escrito para PopCorn deve ter um orçamento para a execução de cada *computelet*. Os pagamentos pela execução são feitos de acordo com o mercado. Existem dois mecanismos de mercado implementados. Um vendedor de tempo de CPU visita uma página web com um browser que tenha habilitada a máquina virtual Java. Esta página contém um applet que começa a trabalhar com os computadores vendedores, que recebem computelets para executar. Para isto o vendedor de recursos tem que visitar o web site do mercado, onde ingressa a informação de sua conta previamente registrada (login e

senha), para começar a venda do tempo de sua CPU e ir acumulando *PopCoins*[2]. Por default o vendedor só vende seu tempo de CPU ao comprador com a maior oferta, mas existe um mecanismo alternativo para a venda, no qual o vendedor não precisa ter uma conta, apenas visita a página de onde descarrega um applet e começa a receber computelets para executar, enquanto ele brinca com um jogo, olha uma pintura, ou obtém algum tipo de informação em troca pelo tempo de CPU.

A alocação de recursos em PopCorn está baseada em leilões. Existem 3 tipos de leilões, e cada leilão manipula um mercado de recursos separado. O primeiro leilão é o chamado de "Repeated Vickrey Auction" onde o tempo de CPU dos vendedores é leiloado entre os atuais compradores cada vez que um novo computelet requer execução. Em cada rodada deste leilão, os compradores submetem ofertas para um computelet especial que tem as funções de leiloeiro. O ganhador é aquele que faz a oferta mais alta. Para este caso o vendedor pode especificar um preço inicial, e os compradores precisam especificar as ofertas que podem ser submetidas entre as rodadas. O segundo tipo é um leilão chamado "double auction" onde os compradores e vendedores submetem um preço mínimo, um preço máximo e possivelmente uma taxa de variação a ofertar. O vendedor começa com uma oferta no preço máximo que vai sendo automaticamente diminuída na taxa especificada até que encontrar um comprador ou alcança o preço mínimo. Da mesma maneira, os compradores começam oferecendo o preço mínimo e vão incrementando suas ofertas até que se encontra um vendedor com o preço oferecido, ou se alcança o preço máximo. Quando um comprador é encontrado, o computelet é enviado para execução sobre o computador vendedor, e o pagamento é feito ao preço combinado. O terceiro mecanismo é chamado de "clearinghouse double auction", que é similar ao anterior, exceto que em cada rodada é feita mais de uma alocação (ou seja, vários recursos são alocados para vários computelets ao mesmo tempo).

Grupos em PopCorn

O sistema PopCorn agrupa os computadores por mercados e a informação de cada mercado é distribuída em vários módulos (os módulos centralizam as funções de alocação, pagamento e negociação) que se encarregam de manipular todo o processo da alocação, assim como de fazer o pagamento de PopCoins para os computadores. Neste mercado existem os agentes de compra e de venda, e uma entidade encarregada de buscar as propostas destes. A

[2]Assim é chamada a unidade monetária desta economia.

informação é armazenada numa pilha com prioridades por oferta. As melhores ofertas ficam no topo da pilha.

Uma propriedade negativa desta abordagem é que o mercado não é distribuído, o que o converte em um gargalo do sistema POPCORN. Testes têm demonstrado que quando uma grande quantidade de computelets de tamanho muito pequeno chegam para o mercado, o computador onde fica o mercado gasta a maior parte de seu tempo na comunicação. Uma solução é distribuir o mercado para eliminar este gargalo. Outra solução é implementar uma rede de mercados que possam se comunicar e balancear a carga entre eles, mas isto ainda não foi implementado e precisa ser melhor avaliado.

O método de alocação do PopCorn é útil para aplicações que têm poucas tarefas. O mecanismo do leilão não é interessante para aplicações massivamente paralelas porque estas geram muitas mensagens. Outro fato é que em aplicações paralelas o esquema de leilões pode gerar concorrência entre tarefas da mesma aplicação, que provavelmente têm o mesmo orçamento, e não entre tarefas de aplicações distintas que dariam maior dinamismo ao mercado de preços. O esquema de recompensa também não é proveitoso já que o pagamento com o uso de jogos não é recompensa suficiente para incentivar participações futuras. O acúmulo de créditos para uso posterior em execuções de aplicações próprias não é muito útil, porque são poucas as pessoas que conhecem e programam este tipo de aplicações, e finalmente a boa vontade para colaboração das pessoas não é motivo suficiente para alcançar uma grande captação de computadores para o sistema, pelo que é necessário um esquema de recompensa mais atrativo.

2.4.2 Algoritmos Microeconômicos para Balanceamento de Carga em Sistemas de Computação Distribuídos

Ferguson *et. al.* [FE96, FE88] propõem algoritmos para alocação de tempo de CPU e de enlace de comunicação (link) usando conceitos microeconômicos. Um computador tem *recursos* (processador, memória, largura de faixa para comunicação) que geralmente são escassos e devem ser alocados aos *consumidores* dos recursos. Os consumidores são as unidades de trabalho chamadas de transações, processos ou tarefas. Os fornecedores de recursos são os processadores e seus sistemas operacionais. É proposto um modelo onde os consumidores e fornecedores (chamados de agentes) procuram satisfazer suas próprias funções de utilidade. Nenhum esforço global é feito para melhorar ou otimizar o desempenho

do sistema como um todo. O objetivo é definir adequadamente as utilidades dos agentes e as regras para maximizar estas utilidades, de forma que permita a melhoria do desempenho global através de uma *mão invisível*, como o que acontece na economia do mundo real com a interação entre a oferta e a demanda [FE96].

A principal vantagem desta abordagem é que a complexidade diminui significativamente quando o sistema distribuído cresce e com ele a dificuldade na tomada de decisões para a alocação. As abordagens microeconômicas limitam esta complexidade dividindo o problema global complexo, da alocação de muitos recursos fornecidos por muitas fontes e usado por muitos tipos de consumidores, em um conjunto de subproblemas simples e independentes.

O sistema é dividido em duas classes de agentes. Os agentes são os processadores e os trabalhos que entram no sistema. Para cada tipo de agente é definido um conjunto de objetivos e regras simples para alcançar seus objetivos no sistema competitivo, sem cooperar com outros agentes. Uma efetiva alocação global de recursos é alcançada indiretamente através da concorrência, procurando a autosatisfação.

As aplicações têm um orçamento que é a quantidade de dinheiro que devem pagar pelos recursos que irão comprar. A compra de recursos é feita submetendo ofertas para um leilão entre todos os consumidores interessados nestes recursos. Os recursos são o tempo de CPU e o enlace de comunicação. O tempo de CPU e medido em ms (milisegundos) e o enlace de comunicação é medido em bytes. A compra do recurso tempo de CPU é por faixas de tempo (uma unidade de tempo como mínimo, a unidade é medida em milisegundos). Para submeter ofertas, a quantidade de tempo requerida pela aplicação deve ser conhecida com antecedência. O enlace de comunicação é oferecido por byte a transmitir. Para submeter ofertas pelo enlace, é necessário conhecer a quantidade de bytes a transmitir. As aplicações calculam seu orçamento antes de submeter as ofertas, ou seja criam uma lista com os computadores nos quais podem submeter as ofertas (que o orçamento pode pagar).

Esta lista de computadores é criada a partir de um dos três tipos de preferência para orçamento, que são mostradas a seguir:

Preferência pelo preço: nesta preferência a aplicação deseja ser servida com o menor custo possível (gastando a menor quantidade possível de seu orçamento). Nesta preferência, na maioria dos casos observados, a aplicação é executada em processadores remotos.

Preferência pelo tempo de serviço: preferência segundo a qual a aplicação prefere um computador que possa servi-la rapidamente. Na maioria dos casos, as aplicações são executadas no mesmo processador onde foram criadas, devido ao tempo para a migração, mas devem migrar se não tiverem um orçamento compatível com este computador.

Preferência pelo tempo de serviço versus preço: nesta preferência são consideradas as duas preferências anteriores no cálculo do orçamento. Existem pesos para cada critério, e o a preferência resultante será dada pelo valor alocado ao peso do critério predominante.

A alocação é decidida por meio de leilões. Existem dois modelos de leilão implementados: o *Sealed Bid Auction* e o *Dutch Auction.* No *Sealed Bid Auction* todas as ofertas por um recurso são abertas ao mesmo tempo quando o recurso estiver disponível. Ganha quem oferecer mais por unidade de recurso (unidade de tempo para CPU e byte a transmitir para enlace de comunicação). No modelo de leilão *Dutch Auction* o processador coloca um preço para o recurso disponível e solicita ofertas. Se nenhuma oferta for submetida, o preço será diminuído. Se houver mais de uma oferta, o processador aumenta o preço e solicita mais ofertas, tendo como novo preço base para o leilão, o maior oferecido. O ganhador será escolhido, depois de uma ou mais rodadas em que somente uma oferta é submetida. O *Dutch Auction* é similar a múltiplas rodadas do *Sealed Bid Auction.*

Esta abordagem parece ser interessante para sistemas distribuídos mas não para sistemas massivamente paralelos, já que o esquema de leilões sobrecarrega o sistema com muitas mensagens. São necessárias mensagens para publicar o preço do recurso, para submeter as ofertas, para receber as respostas, para submeter novas ofertas e para indicar o ganhador do leilão. Por outro lado a concorrência entre aplicações não é comum, mas sim a concorrência entre tarefas da mesma aplicação, que têm os mesmos orçamentos e parâmetros. Outra desvantagem na implementação deste modelo é a suposição de que o tempo de processamento é conhecido, dado que não é simples de se obter.

2.4.3 PINCO: A Resource Management Tool for Heterogeneous Networks

PINCO (Programming envIronment for Network of COmputers) é um conjunto de programas utilitários para administrar os recursos de uma rede de estações de trabalho.

Estes utilitários estão implementados sobre PVM (Parallel Virtual Machine) [GBD+94] e sua arquitetura é similar a esta. O PINCO está baseado em um *PINCO Master Daemon* executando sobre o Host Master (geralmente o de melhor desempenho da rede). O Master Daemon é um processo único na rede. Este daemon está composto por três módulos de software : o *Load Monitor (LM)* que recolhe a informação da situação de cada host, o *Resource Manager (RM)* que é o responsável pelo balanceamento da carga de trabalho entre os host, e o *Communication Library INterface (CLIN)* que contém todas as funções de comunicações, que são baseadas em PVM. Em cada host é executado o *PINCO Local Daemon* que é composto também de três módulos: o *Local Load Monitor (LM)* que avalia periodicamente a carga local e envia esta informação para o Daemon Master, o *Process Handler (PH)*, que é o responsável por ativar tarefas locais de acordo com os requerimentos do Master Daemon e por enviar informação sobre o estado da execução de processos quando se precisa, e o *Local Communication Library INterface* que contém as funções de comunicação para o daemon local.

Também há uma interface de usuário simples para iniciar a execução de aplicações paralelas sobre um cluster de estações de trabalho. PINCO é composto de 3 partes: o *Job Scheduler*, o *Resource and Task Manager* que fornecem as utilidades e o *Application Interface* que atua no nível intermediário, entre a aplicação e os outros dois componentes. Também fornece um ambiente de programação para ajudar os programadores no desenvolvimento de aplicações paralelas.

Cada aplicação paralela é executada sobre uma partição, ou seja, sobre um subconjunto de nós. Devido à heterogeneidade da arquitetura de hardware, precisa-se de alguma forma de normalização para avaliar e comparar dinamicamente o poder computacional de cada nó. É usado o poder de um computador Pentium 133 para as normalizações. A décima parte do poder de um Pentium 133 descarregado representa uma unidade de medida chamada ETP (Tenth of a Pentium). Em cada momento PINCO assume que tem um conjunto de Pentiums 133 descarregados e atende cada requerimento escolhendo um conjunto de máquinas físicas que oferece o número de ETPs requeridos pela aplicação para sua execução.

A alocação em PINCO é feita calculando um valor em ETP para cada computador da rede por meio de um conjunto de testes (benchmarks) e, quando uma aplicação precisa ser alocada, é calculada a quantidade de ETPs que ela precisa para ser executada e que será fornecida pelo Job Scheduler.

A maior desvantagem deste sistema está no fato de que é necessário conhecer a quantidade de ETPs que a aplicação precisa para sua execução, já que calcular os requerimentos de processamento de uma aplicação não é simples, mesmo para aplicações pequenas. Outra desvantagem é que a rede pode potencialmente ser inundada com mensagens periódicas para o Load Monitor (LM) desde os daemons locais, para atualizar o estado dos computadores. O fato do Load Monitor ser centralizado é outra desvantagem porque em redes grandes torna-se um gargalo. Por estes motivos um sistema deste tipo não é adequado para sistemas massivamente paralelo virtuais baseados na Internet, já que estes estão formados por um número grande de computadores, e suas aplicações por um número elevado de tarefas.

2.4.4 Economia WALRAS

O ambiente de desenvolvimento desenvolvido por Michael Wellmans chamado WALRAS [WM93] fornece uma interface de programação para a criação de agentes em uma economia computacional. WALRAS é um ambiente de programação geral orientado ao mercado, e serve para a construção e análise de sistemas de alocação de recursos distribuídos.

O propósito deste sistema é transformar problemas distribuídos complexos em problemas gerais de equilíbrio econômico[3], encontrando a solução do problema original com o cálculo do equilíbrio geral do modelo que o representa.

WALRAS fornece os métodos para definir as partes da economia a implementar. Um *bem* representa qualquer coisa que pode ser comercializada, incluindo os serviços feitos pelos processos. Um *leilão* representa um mecanismo para aceitar as ofertas e determinar o preço para um bem específico. Um *Agente* pode ser qualquer entidade que participará em um leilão. Os *consumidores* são uma classe de agentes que participam no mercado para maximizar certas funções de utilidade.

[3]Mas sem mudar de natureza; os problemas gerais são representados em um modelo econômico onde os elementos interagem e procuram a satisfação geral.

Nesta economia existem *consumidores* e *produtores*. Todos os consumidores são dotados com uma quantidade de *bens*, que devem ser negociados com o objetivo de maximizar as utilidades. Os *produtores* são uma classe de agentes que participam no mercado transformando as *entradas* que obtêm dos *consumidores* em *saídas* para estes. As *Ofertas* podem ser submetidas por qualquer dos agentes participantes de um leilão, e geralmente estas ofertas são divididas em *ofertas de demanda* e *ofertas de abastecimento* (dependendo do tipo de agente que a submete).

WALRAS calcula os valores nos quais o mercado alcança o equilíbrio. Estes valores são calculados para todas as ofertas ao mesmo tempo. Especificamente, somente é calculado o preço no qual o sistema obtém o equilíbrio, a partir das ofertas atuais.

WALRAS faz uma suposição que em economia é chamada de "concorrência perfeita". Cada agente considera o preço como está, e não considera o efeito de suas ações sobre o equilíbrio de preços. Esta suposição traz como resultado a existência de uma conduta individual particular para cada agente. Esta suposição não permite que monopólios existam, ou que pequenos grupos de fornecedores explorem o poder do mercado (oligopólio). Estes agentes usam a habilidade para incrementar seu proveito pelo aumento do preço. Porém agentes nesta situação não estão atuando de maneira racional (para maximizar seu benefício individual) e a vantagem de não precisar de regulações legais como as economias do mundo real, tornam esta abordagem interessante e prática.

2.4.5 Sistema para Processamento Massivamente Paralelo na World Wide Web: JoiN

O JoiN é uma plataforma para processamento massivamente paralelo baseado na Internet (MPVC) [EH98]. Esta baseada em applets Java que podem ser executados em browsers que tenham uma máquina virtual Java, e oferece uma interface de programação baseada em tocas de mensagens.

Este MPVC está distribuído em grupos, e os grupos estão formados por computadores chamados de *trabalhadores* e outros, que cumprem uma função especial, chamados de *coordenadores*. Esta distribuição é feita para evitar que os coordenadores fiquem sobrecarregados quando o número de computadores cresce em grande escala.

Os grupos de JoiN estão conectados logicamente usando uma topologia de hipercubo binário para permitir que as tarefas em grupos diferentes possam se comunicar. Esta estrutura lógica de conexão recebe o nome de *Hipercubo Dinâmico Virtual (DVH[4])*. O DVH está formado por D dimensões, e cada dimensão recebe um identificador de grupo formado por D bits que representam a posição do grupo no hipercubo (Fig. 2.1). Esta topologia foi escolhida por vários motivos: é usada amplamente em computadores paralelos reais, tem sido muito estudada e apresenta as vantagens de simetria, diâmetro logarítmico e boa tolerância às falhas [RF87]. Um outro motivo para escolher esta topologia é que tem-se observado que os padrões de comunicação normalmente encontrados nos algoritmos paralelos (anel, grade n-dimensional, árvore, entre outros) estão contidos no hipercubo binário e podem ser obtidos pela desconsideração de alguns dos seus enlaces de comunicação. Estes hipercubos são chamados de dinâmicos por que têm a capacidade de aumentar ou diminuir de tamanho de acordo com número de computadores. Estes processos são chamados de expansão e retração, respectivamente.

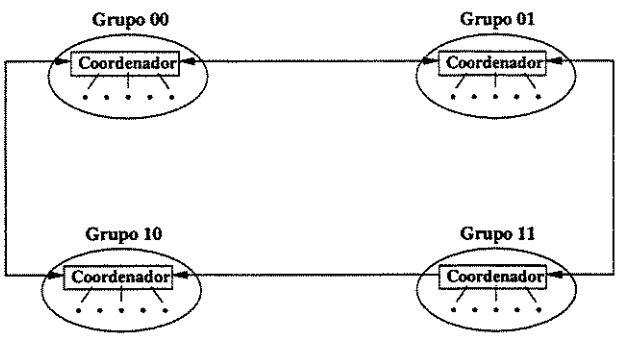

Figura 2.1: Hipercubo de duas dimensões formado por grupos. As setas indicam as conexões lógicas que existem entre coordenadores.

Os grupos colaboram entre si para a alocação. Cada coordenador de grupo tem uma lista de outros grupos que podem ser vizinhos lógicos da topologia usada para interconectá-los, ou um grupo especial de vizinhos chamado de *Buddy Set* [EH98]. Este grupo é um conjunto de vizinhos formado de tal maneira que minimiza as probabilidades de ocorrência de colisões e congestionamento. Uma colisão ocorre quando dois ou mais grupos selecionam o mesmo grupo para enviar um pedido de criação de tarefas. O congestionamento ocorre quando um conjunto dos grupos vizinhos ficam sobrecarregados e não são capazes de criar novas tarefas para outros grupos.

[4]Dynamic Virtual Hipercube.

Como este MPVC está baseado na Internet, os computadores que formam parte desta rede são de donos particulares que precisam de um estímulo para colocar seus computadores à disposição do sistema. Embora o JoiN ainda não tenha implementado nenhum esquema de recompensa, sua especificação prevê a existência destes mecanismos.

Por suas características de escalabilidade, heterogeneidade e complexidade, o sistema JoiN é adotado como referência nas discussões das propostas deste trabalho.

2.5 Conclusões

Neste capítulo foram mostrados os motivos pelos quais a economia pode ser usada em ciência da computação, deixando claro que os mercados são uma boa opção para os sistemas de alocação de recursos, como mostrado em alguns dos sistemas descritos em 2.4. Fica claro que pode ser aproveitado o grande conhecimento existente na área da economia nas pesquisas em ciência da computação.

Nos trabalhos descritos percebemos claramente (pelo fato do crescente número destes sistemas) que existe uma tendência em usar sistemas de processamento massivamente paralelos virtuais baseados na Internet, mesmo com as limitações hoje existentes como: largura de faixa limitada, falta de segurança, alta latência na rede de comunicação, entre outros. O fato de que as velocidades dos links estão aumentando e de que novos mecanismos para solucionar os problemas de segurança estão surgindo, fazem que os sistemas baseados na Internet tenham um futuro promissor.

Embora as propostas desta dissertação não sejam específicas para um MPVC em especial, o MPVC JoiN será usado como base nas discussões apresentadas nos próximos capítulos.

Capítulo 3

Classificação dos computadores

3.1 Introdução

Os MPVCs são formados por computadores que estão conectados à Internet [EM99]. A participação de um computador em um MPVC se dá por meio de uma página hospedada em um Web site onde o computador que deseja participar declara a sua disponibilidade como é mostrado na Fig. 3.1.

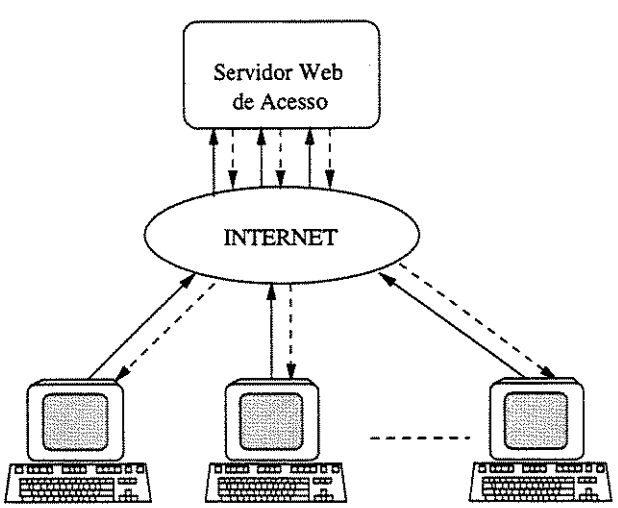

Figura 3.1: Ingresso no MPVC

Neste processo os computadores devem ser registrados e, de alguma maneira, deve ser feita uma diferenciação entre os mesmos, devido à heterogeneidade presente na Internet.

A primeira idéia que surge para diferenciar os computadores é usando a potência do computador. Para isso teríamos que definir quais os critérios para avaliar a potência de um computador, já que potência é um termo muito amplo e não é fácil definí-lo dentro de um contexto. Temos que pensar em um critério simples, concreto, exato e que represente bem um computador diferenciando-o de outros.

A velocidade do computador é um bom critério, mas teríamos que definir que tipo de velocidade e de quais elementos do computador. Por exemplo, pode-se medir a velocidade do processador, barramento, acesso à memória, placas de rede, portas paralelas, portas seriais, controlador de acesso direto à memória (DMA), controlador de vídeo, do co-processador matemático, do processador gráfico e de mais outros elementos que formam um computador. Uma das questões que surge é: quais destas medidas são úteis? E, quais são factíveis? A utilidade dos testes feitos depende da aplicação que precisa conhecer estes dados. Tecnicamente todos os testes mencionados acima podem ser feitos, mas a principal limitação para fazê-los é a capacidade de armazenamento necessária para guardar os seus valores. Para nossas necessidades, a velocidade do processador e do enlace de comunicação são as mais importantes. A velocidade do processador é um indicador do poder de processamento, enquanto a velocidade de link indica o poder de comunicação.

Na literatura atual, para a medição de parâmetros de computadores são usados testes (benchmarks) padronizados [BMPD], como os testes para medir FLOPS (Floating Point Operations Per Second), 3D-Bench (para determinar a velocidade da placa de vídeo), Busperf (para medir a taxa de transferência do bus), dentre outros. Há também software especializado para calcular o poder computacional de um computador [CSS], de uma rede, e dos dispositivos conectados nela [HPB]. Estes testes são muito específicos e por isso muito exatos. A principal desvantagem de usar os softwares existentes para realizar os testes é que em geral são muito demorados e precisam de certos privilégios no sistema para executar (que alguns sistemas não podem oferecer). Para o caso de sistemas de processamento massivamente paralelo é preciso que os testes sejam rápidos e seguros. Neste aspecto, os testes existentes não são úteis e torna-se necessário desenvolver um novo método de teste. O mesmo deve ser simples, de execução rápida, seguro e ter em conta as limitações do MPVC onde será usado.

A classificação proposta neste trabalho para os computadores é baseada em um parâmetro que representa o poder potencial de um computador quando este ingressa no sistema, mas ele não é um indicador constante no tempo. A carga de um computador pode aumentar

ou diminuir por ações do dono, ou por eventos na rede da qual o computador faz parte. A solução para este problema é criar uma maneira de medir o poder do computador a cada certo tempo, ou quando a carga muda. Mas isto também não é possível em sistemas MPVC para os quais esta classificação está sendo proposta, pela quantidade de mensagens necessárias para manter atualizada a classificação em tabelas. Também não existe uma maneira simples de se saber quando um computador mudou de nível de carga sem ter acesso a elementos do computador que às vezes não estão acessíveis (por exemplo, applets não têm acesso a recursos do sistema operacional[1]).

A maneira de calcular o parâmetro para o computador (que chamaremos de velocidade) é feita por meio de um programa que implementa um conjunto de testes (benchmarks). Este programa é descarregado na forma de um *Thread* (uma linha de execução), junto com a plataforma básica do MPVC quando o computador ingressa no sistema.

Podem existir várias maneiras para calcular a carga de um computador e as mais gerais são apresentadas na seção a seguir.

3.2 Métodos de Classificação de Computadores

A medição da carga de um computador não é assunto trivial, mas dela depende em boa parte o sucesso de algoritmos de balanceamento de carga.

Ao falarmos de classificar os computadores, podemos dividir os testes em 3 tipos, sendo que um deles combina as vantagens dos outros dois, tentando minimizar as desvantagens para obter um método simples e eficiente de classificação que será usado na alocação de recursos apresentado no Capítulo 4.

Os tipos de testes são mostrados a seguir, assim como as vantagens e desvantagens de cada um deles.

[1] Os applets Java normalmente não têm acesso ao disco [N98A, SUN97a], mas versões mais atuais de Java permitem isto se o dono do computador der permissão explicitamente, baseado em applets que são assinados digitalmente [N98B], oferecendo segurança para o computador e para o dono.

3.2.1 Método Dinâmico

Este método de teste consiste em executar benchmarks a cada certo tempo ou quando algum parâmetro do computador muda. Os testes são feitos para calcular valores que representam a velocidade de um computador. Os resultados destes testes devem ser registrados em algum meio físico. Para nosso caso estes resultados são enviados para um agente (chamado de coordenador de grupo) que registra os valores de todos os computadores em uma tabela. Estes testes podem medir a velocidade do processador, a quantidade de memória, velocidade do link, etc., mas fazer isto consome muito tempo e corre-se o risco de gastar mais tempo na execução de testes que na execução de aplicações. Além disso, há o problema da quantidade de mensagens que são geradas para manter a informação atualizada, o que pode saturar o coordenador e torná-lo um gargalo.

Outra desvantagem é que este tipo de teste gera muita informação, e precisa-se de um meio para armazená-la. Isto pode ser uma limitação, pelo fato da maioria dos sistemas de processamento massivamente paralelo baseados na Internet (MPVC) serem implementados sobre browsers com Máquinas Virtuais Java que têm uma série de restrições de segurança [N98A, SUN97a]. Estas restrições são, por exemplo, a não permissão de acesso ao disco rígido local, ou de abertura de conexões com outros computadores. Mas este ambiente está mudando, conforme novas maneiras de prover segurança vão sendo criadas e conforme os browsers nas versões mais atuais vão implementando soluções para lidar com este problema [N98B].

Este tipo de teste pode ser exato e atualizado, mas as desvantagens enumeradas fazem com que sua implementação seja ainda inviável para um MPVC baseado na Internet.

3.2.2 Método Estático

Este método de teste se baseia na medição de parâmetros das características físicas do computador (tipo de processador, memória, velocidade do link) e na quantidade de tarefas ou processos atualmente executando no computador para determinar a carga.

Ele parece ser mais eficiente porque não gera tráfego adicional na rede e nem usa tempo de computação para fazer benchmark. Entretanto não é garantida a confiabilidade das medidas, já que que o computador pode ficar mais carregado ou descarregado por

eventos fora do controle do sistema e de maneira imprevisível. Além disso a confiabilidade dos testes depende dos parâmetros a medir e do número de vezes que a medição é feita.

3.2.3 Método Híbrido

O método de testes dinâmicos parece ser conveniente para qualquer sistema, mas as desvantagens mencionadas em 3.2.1 o fazem não muito interessante para sistemas como o considerado neste trabalho. Por outro lado os testes estáticos não sobrecarregam o sistema, mas também não são exatos. Tendo esta situação, na qual nenhum dos métodos apresentados é adequado, é possível pensar em termos médios, ou seja em uma combinação dos dois métodos anteriores. O método proposto é chamado de híbrido porque usa testes dinâmicos na primeira vez que é feita a medição (quando o computador entra no sistema) e testes estáticos depois que o computador é registrado e classificado. Os testes estáticos são baseados no número de tarefas e alguns parâmetros adicionais que ajudam a representar o computador através de um valor produzido a partir dos outros.

O método de classificação requer que o MPVC esteja distribuído em grupos (a formação dos grupos será tratada com maiores detalhes no Capítulo 4), tendo um coordenador por grupo. O coordenador se encarrega, dentre outras coisas, de registrar o valor inicial da velocidade do computador em uma tabela local, a qual é única para o grupo.

Todo o processo de descarga, execução dos testes e registro dos resultados são mostrados na Fig. 3.2. O computador que deseja participar do MPVC ingressa em uma página web e declara a sua disponibilidade (passo 1 da Fig. 3.2). O applet é descarregado para o computador participante (passo 2). O applet inicia a execução dos testes e o resultado final é enviado para o coordenador de grupo (passo 3) que se encarregará de registrar e classificar o computador na tabela local de computadores (passo 4). Quando o applet de benchmarks é descarregado, também é informado ao computador o grupo ao qual pertencerá e no qual deve se registrar. Depois que o computador registra a carga no coordenador local, este atualiza a carga pelo teste estático cada vez que existe uma mudança no número de tarefas executando no computador. Em tempo de execução o coordenador usa a informação obtida no teste dinâmico, e as informações dos testes estáticos para tomar decisões de alocação.

A alocação é feita normalmente para computadores que têm um nível mínimo de poder computacional, e assim se garante que o sistema funcionará com um mínimo de desempe-

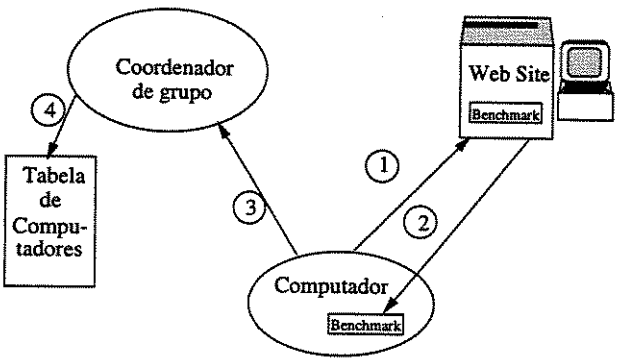

Figura 3.2: Descarga do applet de testes benchmark.

nho. A referência tomada como valor mínimo de desempenho é um computador que só consegue executar uma tarefa, e que estará em um estado chamado de **sobrecarregado** com qualquer outra tarefa adicional. Qualquer computador estará neste estado se o seu poder computacional estiver abaixo do poder computacional do computador de referência. Estar neste estado significa que não é possível executar mais nenhuma tarefa. O resultado disto é que a alocação de tarefas será feita nos computadores que estiverem acima deste poder computacional mínimo. Quando se precisar fazer uma alocação e todos os computadores estiverem no estado **sobrecarregado** (não conseguindo alocar mais tarefas), serão contactados outros grupos para auxiliar na alocação. O estado sobrecarregado para um computador é indicador de que este não pode aceitar nenhuma tarefa além das atuais. Se um computador chega no estado sobrecarregado, este é marcado como inelegível para uma nova alocação até a carga dele diminuir.

Descrição dos testes realizados

Os testes feitos para classificar os computadores são 4 e cada um deles procura medir elementos do computador que consideramos importantes para processamento de aplicações de propósito geral em um MPVC. Destes testes, 3 são relativos a velocidade de CPU e 1 a respeito de enlace de comunicação.

- Teste de Loop Vazio: usado para medir a velocidade bruta do processador. O teste consiste de uma certa quantidade de loops vazios, ou seja, um segmento repetitivo de programa sem nenhum código específico.
- Teste de operações simples em ponto flutuante: usado para determinar o poder de

cálculo matemático básico do processador. Para este teste são consideradas 4 operações básicas em ponto flutuante : soma, subtração, multiplicação e divisão.

- Teste de operações complexas em ponto flutuante: usado para determinar o poder de cálculo avançado do processador. Para este teste são feitas 3 operações em ponto flutuante : exponenciação, raiz quadrada e logaritmo natural.

- Teste de velocidade de link: usado para estimar a velocidade do enlace de comunicação e a latência da rede. O teste é feito usando 8 tamanhos de pacotes de dados entre o web site e o computador em teste: 256 B, 512 B, 1 KB, 2 KB, 4 KB, 8 KB, 16 KB e 32 KB.

Logicamente estes testes podem mudar dependendo do tipo de aplicação que o sistema MPVC deve executar.

Todos os testes são repetidos 5 vezes e então calcula-se a média dos valores resultantes para cada um. Esta média será usada no cálculo da taxa de processamento ou de transferência de dados por unidade de tempo. Por exemplo se o computador demora em média 100 ms para executar 10^5 operações em ponto flutuante, então a taxa de processamento para operações simples em ponto flutuante para este computador será de 10^6 operações por segundo. As taxas calculadas para cada teste são combinadas em um valor que chamamos de *preço* do computador, que será detalhado na próxima seção, e que servirá como base para o método de alocação proposto no Capítulo 4.

3.3 Os preços dos recursos

Ao classificarmos os computadores precisamos de um parâmetro para diferenciar uns dos outros. Mas, no momento de calcular este valor temos que fazer considerações a respeito do custo real do serviço dos computadores dependendo do seu potencial.

3.3.1 Definição do preço para os computadores

A discussão desta subseção concentra-se em qual deve ser o preço a pagar pelo serviço dos computadores, já que existem uns mais velozes que outros.

Para esta discussão partimos do pressuposto de que existe um computador com velocidade V, o qual para executar uma aplicação com duração T_v segundos deve receber P créditos/segundo. Para um computador de velocidade duas vezes maior que V, ou seja $2V$, a quantidade a pagar seria $2P$ se consideramos que o preço está em relação direta com a velocidade. Se isto for certo então teríamos uma tarifa básica para um computador de velocidade V, e o preço em relação direta com a velocidade do computador seria

Computador com velocidade $NV \Rightarrow$ preço a pagar será NP créditos/seg, tempo de execução será

$$T_{nv} = \frac{1}{N} T_v \quad segundos \tag{3.1}$$

onde:
V : Velocidade base,
P : preço pago a um computador com velocidade V, para executar uma aplicação,
T_v : tempo de execução de uma aplicação em um computador com velocidade V,
N : número de vezes que o computador é mais rápido que um computador de velocidade V.
T_{nv} : tempo de execução de um aplicação em um computador de velocidade NV.

Na prática, ao se pagar dois computadores com velocidade V (pagamento total será $2P$) não se obterá o mesmo desempenho que pagando para um computador que tenha velocidade $2V$ (também pagando $2P$), já que as expressões anteriores não estão completas (não consideram tempo de comunicação para a divisão do trabalho entre os computadores). A partir disso observamos que o desempenho de C computadores com velocidade V é sempre menor que o desempenho de um computador com velocidade NV (para quando $C = N$), mesmo quando tem-se o caso de paralelismo de dados, no qual o problema a ser resolvido consegue usar todos os computadores disponíveis.

C Computadores com velocidade $V \Rightarrow$ tempo de execução $\frac{T_v}{C} + T_{com}(C)$

onde:
$T_{com}(i)$: função que denota o tempo de comunicação entre i computadores.

Observamos que o tempo total para executar uma aplicação que possua paralelismo é dado pelo número de computadores nos quais a aplicação é distribuída, mais o tempo de

comunicação entre estes. Com isto é possível concluir que, com uma maior quantidade de computadores, o tempo de execução deve ser menor, mas sendo limitado pelo valor mínimo que pode ser obtido com aplicações paralelas (limite de speedup) [LP+99].

Portanto, o tempo de execução de uma aplicação executando em C computadores será dado pela Eq. 3.2.

$$T_n = \frac{T_v}{C} + T_{com}(C), \quad para\ C > 1 \tag{3.2}$$

onde:
T_n : Tempo que C computadores com velocidade V cada demoram para executar a aplicação.

Para o caso em que $N = C$, podemos calcular quantas vezes um computador com velocidade NV é mais rápido que C computadores com velocidade V basta dividir 3.2 por 3.1.

$$\begin{aligned} \frac{T_n}{T_{nv}} &= \frac{\frac{T_v}{N} + T_{com}(C)}{\frac{1}{N}T_v} \\ \frac{T_n}{T_{nv}} &= 1 + \frac{N * T_{com}(N)}{T_v} = 1 + \beta \end{aligned} \tag{3.3}$$

Como $\beta = N * T_{com}(N)/T_v$ é sempre maior que 0, para $N > 1$ obtém-se:

$$T_n > T_{nv} \tag{3.4}$$

Na Eq. 3.4, observa-se que o tempo de execução de uma aplicação em N computadores com velocidade V é maior que o tempo de execução da mesma em um computador com velocidade NV.

Portanto, é recomendável que o preço pago a um computador com velocidade NV seja maior que o valor total pago a N computadores com velocidade V. Este valor a mais no preço pode ser baseado no diferencial de velocidade expresso por β.

3.3.2 Pagamentos dos computadores

Os proprietários dos computadores participantes em um MPVC deveriam ter um incentivo para manter e/ou aumentar a sua participação. Existem muitas maneiras pelas quais o computador ganharia créditos (moeda virtual do modelo econômico), dentre as quais podemos citar:

1. Disponibilidade: o computador ganha uma quantidade $C_{Disponibilidade}$ de créditos por unidade de tempo, apenas por estar disponível para o sistema, mesmo que não processe nada.

2. Trabalho: o computador ganha $C_{Trabalho}$ créditos por unidade de tempo de execução das tarefas, ou seja, os créditos que a tarefa paga ao computador para se executar nele. Estes créditos têm uma relação direta com a qualidade do computador.

3. Incentivo à conclusão: valor adicional de créditos ($C_{Incentivo}$) é para os computadores que terminarem com sucesso a execução de uma tarefa. O objetivo é incentivar os donos dos computadores a os deixarem disponíveis até o processamento concluir.

4. Incentivo ao cumprimento de prazo: valor ($C_{CumpPrazo}$) a ser pago somente se uma tarefa é executada dentro de um prazo pré-determinado.

Então a quantidade de créditos ganha por um computador será:
$Ganho = C_{Disponibilidade}[+C_{Trabalho}][+C_{Incentivo}][+C_{CumpPrazo}]$,
onde
[ ] indica um ganho opcional.

Com este esquema garantimos que no mínimo os computadores ganharão uma quantidade de créditos por disponibilidade, o que ajuda a incentivar a participação de um maior número de computadores. Estes créditos podem vir a ser convertidos mais tarde em bens ou serviços, como por exemplo, números para uma loteria onde prêmios superiores aos créditos podem ser obtidos.

3.3.3 Proposta para cálculo do custo de um computador para o sistema

Embora em 3.3.1 tenha sido demonstrado que o pagamento para um computador com velocidade NV deve ser maior que o total pago a N computadores com velocidade V, nesta proposta para cálculo do preço dos computadores, isto não será levado em conta. A maior dificuldade para tal consideração está na determinação do fator β, que representa os requisitos de comunicação da aplicação, os quais são muito variados e dependem muito da implementação do algoritmo. Sendo assim, os preços serão calculados tendo em conta somente os resultados dos testes (benchmark) descritos na Seção 3.2.3 e representarão os custos para processamento nos computadores.

Os preços são medidos em créditos, que é uma unidade virtual da moeda na economia virtual implementada para a alocação de recursos. O crédito é uma unidade contínua (é possível ter frações de créditos como preço) que deve ser paga aos computadores pelas aplicações para a execução de suas tarefas.

Devido à heterogeneidade do hardware dos computadores na Internet é necessário alguma forma de padronização para avaliar e comparar dinamicamente o poder computacional destes computadores. Os resultados de cada teste são normalizados com respeito aos mesmos testes feitos em um computador usado como referência.

Para encontrar um valor que representa a velocidade da CPU optou-se pela média aritmética dos valores obtidos nos testes de CPU como mostrado na Eq. 3.5 e esta média é combinada com o valor obtido no teste de link para obter o preço que será usado para fazer a classificação. Os motivos para se usar a média aritmética é que os testes de CPU têm uma correlação entre eles, são influenciados pelo relógio do sistema e não são independentes. Não é possível que o valor de um teste possa aumentar ou diminuir sem afetar os outros; por exemplo quando o relógio (clock) do computador aumenta então todos os testes irão se alterar.

$$T_{cpu} = \frac{T_{cpu1} + T_{cpu2} + T_{cpu3}}{3} \tag{3.5}$$

Pode se considerar o teste de link como independente da velocidade do processador, já que dentro de certos limites a velocidade do link está determinada mais pela tecnologia de rede usada que pela velocidade do processador. Por isto, consideramos a média dos testes

de CPU (T_{cpu}) e o teste de link (T_{link}) como vetores independentes (ortogonais), podendo encontrar uma resultante que represente ambos. A composição destes dois testes é dada pela Eq. 3.6.

$$Price_i = \sqrt{(T_{cpu})^2 + (T_{link})^2} \tag{3.6}$$

Os valores obtidos nos testes são normalizados com respeito a um computador de referência. Usamos o computador mais lento de nossos testes como referência, mas os valores usados podem ser determinados segundo o critério que se queira ter mais em conta. Os testes de referência foram executados no computador *ken.fee.unicamp.br*[2] e serviram para todo o processo de classificação aqui mostrado. O valor normalizado de cada teste representa quantas vezes o computador é mais rápido que o computador de referência nesse teste, ou seja, quantos computadores de referência representa o computador medido.

O preço encontrado com a Eq. 3.6 indica o valor que as aplicações deverão pagar pela execução de cada uma de suas tarefas. Para facilitar o processo de alocação fazemos a suposição de que todas as tarefas são iguais. Esta suposição é feita porque não existe uma maneira simples de determinar a quantidade de processamento necessária para a execução de uma tarefa. A maneira mais simples de saber a quantidade de processamento de uma tarefa é executando-a, mas isto não é prático; outra maneira é revisando o código da tarefa, mas isto também não é viável.

O preço é armazenado pelo coordenador do grupo em uma tabela local. Além do preço e o identificador do computador (usamos o endereço IP ou o nome do computador se estiver disponível) são armazenados a data do teste, o preço atualizado, o número de tarefas atualmente executando, a quantidade de créditos acumulados (créditos ganhos por trabalho efetuado, que depois poderá ser usada para fazer o pagamento ao computador). Estes dados são usados para fazer a alocação e são atualizados de acordo com a variação da carga do computador e com as condições do mercado nas quais a alocação é feita.

[2]Estação de trabalho SPARCstation, CLOCK: 110MHz, RAM: 32MB, SO: SunOS 5.5.1

3.4 Estado dos grupos

Como foi mencionado anteriormente, os grupos estão formados por um conjunto de computadores e um coordenador. Definimos para efeitos de alocação o *estado de grupo*. Como para os computadores, para os grupos também existe um estado no qual não é possível alocar mais tarefas entre seus computadores. Este estado do grupo é chamado também de **sobrecarregado**. Os preços (calculados pelo número de tarefas executando) dos computadores do grupo são usados para determinar o estado do grupo, partindo do mesmo princípio de que o nível mínimo de desempenho médio do grupo deve ser o do computador de referência, ou seja que a média dos preços dos computadores (o preço representa o poder computacional do computador) seja no mínimo 1. Quando a média dos preços do grupo for menor que 1, então o grupo estará no estado sobrecarregado, não podendo alocar mais nenhuma tarefa de nenhuma aplicação até seu estado mudar (o preço médio do grupo aumentar pela diminuição de tarefas). A Eq. 3.7 é usada para calcular o estado do grupo.

$$E_i = \frac{1}{N} \sum_{i=1}^{N} Price_i \tag{3.7}$$

onde:
E_i : estado do grupo i;
$Price_i$: preço do computador i, calculado tendo em conta o número de tarefas executando;
N : número de computadores do grupo.

O estado do grupo é calculado depois de cada alocação, e quando houver uma mudança no estado (passar de **não sobrecarregado** para **sobrecarregado** ou vice-versa); este fato deve ser informado aos grupos vizinhos.

Este esquema de atualização do estado do grupo minimiza as mensagens trafegadas pela rede, fato que será explorado no algoritmo de alocação de recursos proposto no próximo capítulo.

3.5 Testes de classificação de computadores

Fizemos uma implementação do método de classificação proposto. Para isso criamos um applet Java que é descarregado do servidor Web para o computador testado, e um programa

servidor que executa no servidor Web. O computador se conecta com o Web site (passo 1 da Fig. 3.2) e o applet é descarregado neste computador (passo 2). Os testes descritos no método híbrido são executados no computador e os resultados são enviados para o coordenador do grupo ao qual ele pertence (passo 3). Os resultados são armazenados na tabela de computadores mantida pelo coordenador (passo 4).

Foram testados 54 computadores. A execução destes testes foi feita principalmente com a participação de pesquisadores do LCA/FEEC (Laboratório de Computação e Automação da Faculdade de Engenharia Elétrica e de Computação), mas também houve participação de pesquisadores de outros laboratórios da Unicamp, que ingressaram na página de testes (`http://www.dca.fee.unicamp.br/projects/join/alocacao/pedro/BenchMark/`), assim como algumas pessoas do exterior que foram convidadas para colaborar nos testes[3]. Os testes foram executados em sua grande maioria sobre os browsers Netscape e Internet Explorer.

A execução do applet se inicia com os testes de velocidade de CPU, e continua com teste de loop vazio, de operações básicas em ponto flutuante, de operações complexas em ponto flutuante e acaba com o teste de enlace de comunicação. A tela de execução do applet é mostrada na Fig. 3.3.

Os testes de comunicação são feitos criando-se uma conexão de rede entre o computador que está sendo testado e o computador servidor http (Web site) do LCA/FEEC (`http://www.dca.fee.unicamp.br`). O teste consiste em enviar pacotes de distintos tamanhos: 256 B, 512 B, 1024 B, 2 KB, 4 KB, 8 KB, 16 KB e 32 KB, desde o computador para o servidor, e receber outro pacote do mesmo tamanho. O tempo é medido desde que se envia o pacote até o recebimento da resposta. Em testes preliminares observamos que não existe uma variação significativa (desvio padrão pequeno) quando estes são feitos num intervalo curto de tempo em que não são iniciadas novas aplicações no computador; por isto os testes somente são repetidos 5 vezes cada, e os valores usados para o cálculo das taxas são a média das 5 repetições. Com este valor temos o tempo médio que demora um pacote para trafegar pela rede nos dois sentidos. Usando a metade deste tempo é calculada a taxa de transferência de dados pela rede em bytes por segundo para cada tamanho de pacote (ver Fig. 3.5).

[3]Foram enviados convites para muitos amigos do exterior, mas só alguns aceitaram participar; a negativa foi por razões de segurança o que demonstra que isto é ainda um grande problema para sistemas baseados na Internet.

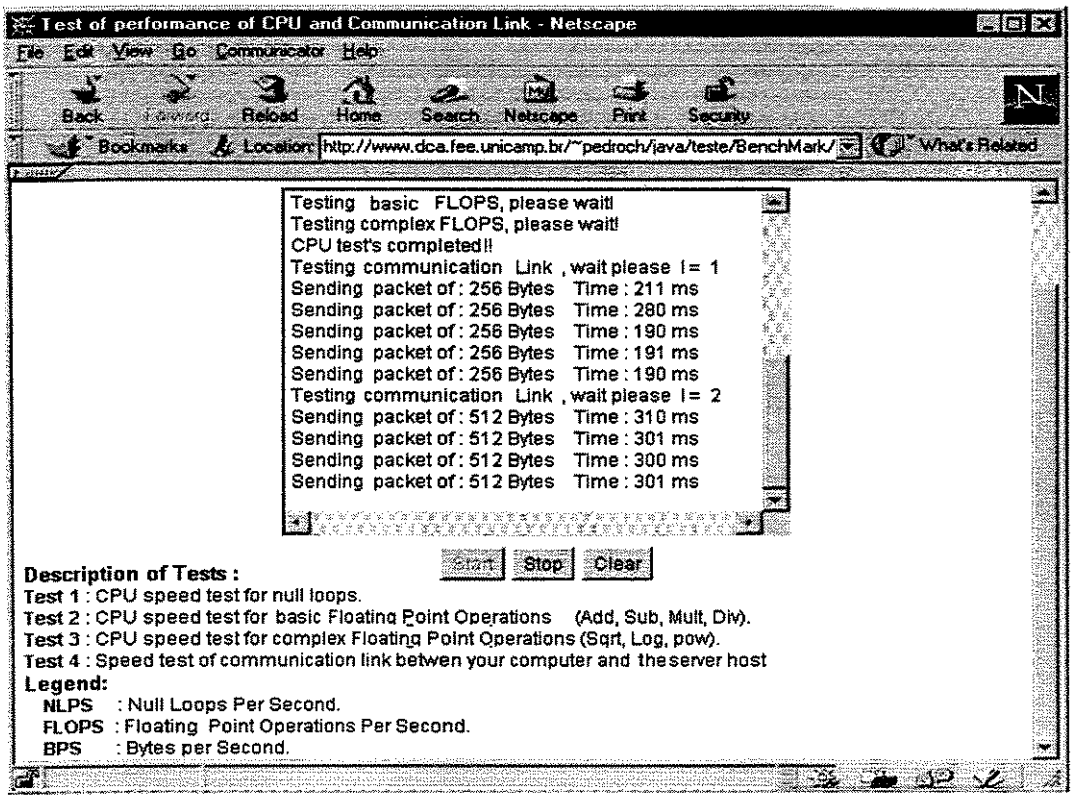

Figura 3.3: Tela de execução do applet de testes benchmark.

Observamos que a execução do teste de enlace de comunicação com todos os tamanhos de pacote é muito demorada. Como os testes devem ser rápidos, necessitamos implementar este teste de uma maneira mais eficiente e que ainda seja confiável. A alternativa é usar pacotes menores ou menos pacotes. A decisão de que tamanho de pacote usar depende de vários critérios, os quais podem ser:

- usar o tamanho de pacote cujo resultado tenha o menor desvio padrão (que foi calculado com os dados obtidos nos testes preliminares). Para este caso o tamanho seria de 16 KB. Este critério nos daria maior confiabilidade para encontrar uma taxa com a menor variação, mas o teste continuaria sendo demorado devido ao tamanho do pacote trafegando na rede;

- usar o menor pacote (256 B) para obter rapidez na transmissão. Neste caso não se teria as melhores taxas de transmissão nem se teria uma média muito confiável, mas o tempo de teste poderia ser reduzido;

- usar o pacote de tamanho que resulte na maior taxa de transmissão, onde se tenha um ponto de saturação da curva de taxas de transferência (Fig. 3.4). Com este critério se teria uma média da taxa de transmissão mais representativa. Porém, pelo observado nos testes, a curva se satura a partir dos 8 KB, o que nos traz as desvantagens de pacotes grandes já explicadas.

Pelas razões acima expostas decidimos usar pacotes pequenos (256 Bytes) para a implementação do teste de enlace. O fato de usar este tamanho de pacotes nos leva a obter taxas de transmissão não muito exatas, o que se vê refletido também no preço do computador. Como o mesmo tipo de erro ocorre com todos os computadores, ele não representa uma limitação, e ainda obtemos a vantagem da rapidez dos testes. Um outro motivo para a escolha do tamanho de 256 Bytes é que a maioria das mensagens normalmente encontradas em redes de computadores é pequena [PH96].

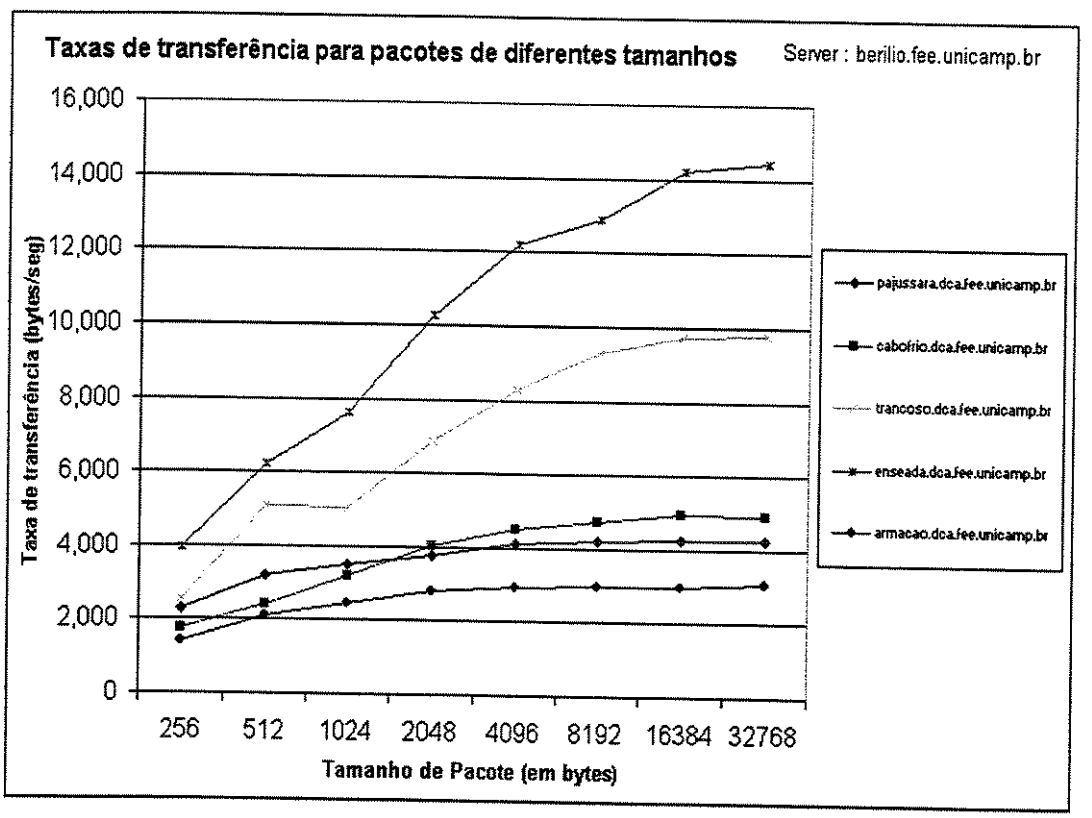

Figura 3.4: Taxas de transferência de dados pela rede com tamanhos diferentes de pacotes (subconjunto das máquinas testadas).

Como observamos na Fig. 3.5, os computadores com enlace de comunicação mais

Computador\T.Transferência	256 B	512 B	1024 B	2048 B	4096 B	8192 B	16384 B	32768 B
pajussara.dca.fee.unicamp.br	2276	3190	3525	3761	4121	4213	4255	4261
cabofrio.dca.fee.unicamp.br	1753	2415	3210	4012	4501	4723	4934	4918
trancoso.dca.fee.unicamp.br	2560	5069	5032	6896	8266	9304	9752	9754
enseada.dca.fee.unicamp.br	3938	6244	7613	10266	12190	12891	14204	14419
armacao.dca.fee.unicamp.br	1388	2098	2465	2823	2943	2991	2997	3051
saopaulo.dmo.fee.unicamp.br	2286	3346	4719	5761	5894	6285	7571	7568
labest01.ime.unicamp.br	502	2498	2629	3208	2141	3330	3013	2429
kinsky.lrprc.fee.unicamp.br	2116	3737	5333	6113	8095	7420	8547	8585
berilio.fee.unicamp.br	1796	1513	1424	1225	1355	1362	1137	576
200.4.193.230	115	118	197	517	949	1027	625	942

Figura 3.5: Taxas de transferência de pacotes com diferentes tamanhos (valores expressos em bytes/segundo.)

rápido são os que estão no mesmo domínio e a diferença com outros subdomínios do mesmo domínio testado (fee.unicamp.br) não é muito significativa, como se observa entre os computadores na Fig. 3.5 (veja os computadores kinsky.lrprc.fee.unicamp.br, berilio.fee.unicamp.br). Quando o teste é feito com computadores externos, observamos que as taxas caem de maneira significativa (veja o computador 200.4.193.230[4]). Estas diferenças são refletidas no preço, mas sua influência é atenuada pelo fato de usarmos a Eq. 3.6 para encontrar o preço como explicado na seção 3.3.3.

Os resultados de alguns dos computadores testados e os valores normalizados dos testes, assim como o preço são mostrados na Fig. 3.6. Esta tabela está ordenada por preço. Os valores normalizados são expressos em número de vezes que o computador é mais rápido em relação ao computador de referência. Na figura 3.7 são mostrados os mesmos valores obtidos nos testes, de maneira gráfica.

Nos testes realizados pudemos observar que os valores de preço maiores são obtidos para computadores pessoais (PCs). A razão é que estes computadores estão executando Windows ou Windows NT como sistema operacional, o qual possui implementações mais atualizadas e eficientes da máquina virtual Java. Para o caso de estações de trabalho com sistema operacional Unix, observa-se que os valores obtidos nos testes de CPU são baixos. A estação Unix mais rápida é guaratuba.dca.fee.unicamp.br que é uma das estações mais rápidas do LCA, mas mesmo assim tem um preço que é 6 vezes menor que o PC mais bem avaliado (campbell.lrprc.fee.unicamp.br).

Para o caso do teste de enlace de comunicação não existem muitas diferenças entre as duas arquiteturas, como pode se observar na coluna Teste 4 da Fig. 3.6 (Teste 4)

[4]Este é um computador instalado no Peru.

Computador / Teste Normalizado	Teste 1	Teste 2	Teste 3	Média T.CPU	Teste 4	Preço
campbell.lrprc.fee.unicamp.br	980.39	555.56	104.17	546.70	62.17	550.23
cotovelo.dca.fee.unicamp.br	666.67	555.56	93.75	438.66	50.30	441.53
meaipe.dca.fee.unicamp.br	641.03	370.37	33.33	348.24	54.64	352.50
pfeiffer.lrprc.fee.unicamp.br	568.18	400.00	68.81	345.66	60.90	350.99
net90134.ime.unicamp.br	617.28	344.83	61.98	341.37	46.61	344.53
net90135.ime.unicamp.br	543.48	357.14	68.18	322.93	31.10	324.43
gonzaga.dca.fee.unicamp.br	390.63	384.62	65.22	280.15	55.83	285.66
mocooca.dca.fee.unicamp.br	526.32	232.56	16.80	258.56	54.64	264.27
net90177.ime.unicamp.br	312.50	113.64	38.86	155.00	50.99	163.17
net90178.ime.unicamp.br	303.03	120.48	39.06	154.19	31.37	157.35
lacom06.ime.unicamp.br	260.42	90.91	26.98	126.10	47.53	134.76
ibm14.fem.unicamp.br	210.08	80.00	26.79	105.62	34.48	111.11
ipanema.dca.fee.unicamp.br	162.60	60.61	20.03	81.08	40.24	90.52
cabofrio.dca.fee.unicamp.br	162.07	60.24	19.76	80.69	40.06	90.09
ilheus.dca.fee.unicamp.br	161.03	54.64	19.87	78.51	38.10	87.27
grumari.dca.fee.unicamp.br	4.62	4.34	9.84	6.27	66.33	66.62
dunas.dca.fee.unicamp.br	9.88	9.31	22.26	13.81	64.39	65.86
calipso.dt.fee.unicamp.br	3.80	3.12	8.36	5.10	60.84	61.06
itanhaem.dca.fee.unicamp.br	4.63	4.33	10.27	6.41	59.18	59.53
berilio.fee.unicamp.br	4.73	4.52	6.68	5.31	54.80	55.06
jureia.dca.fee.unicamp.br	9.97	6.14	21.10	12.40	43.05	44.80
itapua.dca.fee.unicamp.br	3.12	3.07	5.06	3.75	40.70	40.88
atalaia.dca.fee.unicamp.br	3.30	2.98	4.59	3.62	36.76	36.94
pontadareia.dca.fee.unicamp.br	3.31	2.94	3.98	3.41	36.62	36.78
aracati.dca.fee.unicamp.br	3.27	2.97	3.69	3.31	36.43	36.58
mel.dca.fee.unicamp.br	2.75	2.98	4.05	3.26	35.62	35.76
angra.dca.fee.unicamp.br	3.07	2.77	2.58	2.81	34.34	34.46
pipa.dca.fee.unicamp.br	3.29	2.99	3.61	3.30	30.12	30.30
peruibe.dca.fee.unicamp.br	3.20	2.66	3.95	3.27	29.70	29.88
lagoinha.dca.fee.unicamp.br	2.19	2.08	2.68	2.32	29.64	29.73
labest02.ime.unicamp.br	3.35	3.02	3.56	3.31	26.96	27.17
tenorio.dca.fee.unicamp.br	3.20	2.70	3.35	3.08	24.85	25.04
juquei.dca.fee.unicamp.br	2.29	2.03	2.98	2.43	16.99	17.16
ken.fee.unicamp.br	1.00	1.00	1.00	1.00	1.00	1.41

Figura 3.6: Computadores classificados em ordem decrescente de preço.

considerando que algumas estações de trabalho têm enlaces de comunicação bem mais velozes que os dos PCs.

Observa-se tambem que não sempre a velocidade de CPU é quem influencia maiormente no cálculo do preço. Para certo valores mínimos o teste de CPU é quem determina o preço como se mostra na Fig. 3.8.

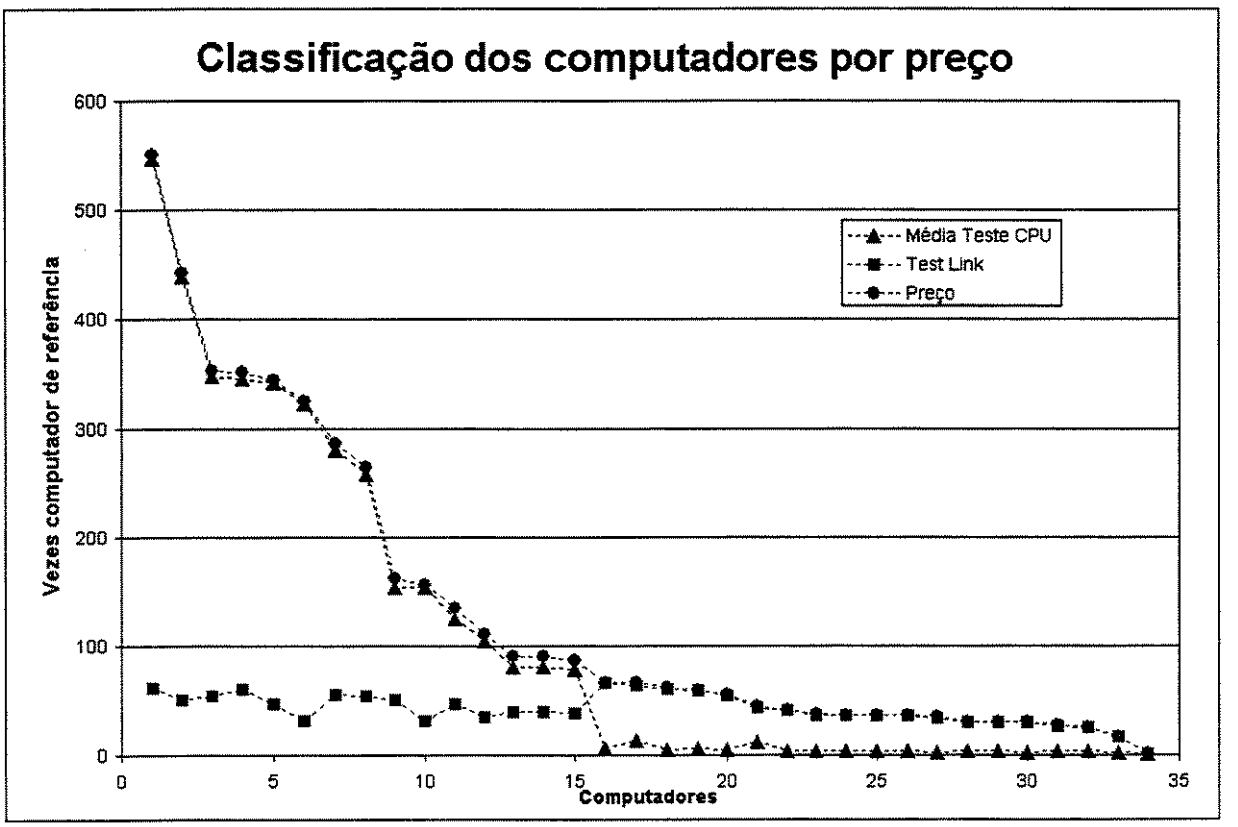

Figura 3.7: Distribuição dos preços dos computadores, com os resultados de cada um dos testes normalizados, assim como o preço calculado a partir deles.

3.6 Conclusões

Neste Capítulo apresentamos a classificação dos computadores de acordo com os parâmetros medidos. Descrevemos o método híbrido que combina as vantagens dos métodos estáticos e dinâmicos e procura diminuir as desvantagens destes. Foi feita uma análise para definir os preços cobrados pelos computadores, afim de demonstrar por que um computador mais veloz deve ser mais caro que um conjunto equivalente de computadores mais lentos.

Também foram apresentados os mecanismos de medição do potencial dos computadores que fazem parte do MPVC e uma maneira de classificá-los por um parâmetro que representa o poder computacional do computador com respeito a um computador de referência. Esta classificação se faz usando um parâmetro chamado de preço do computador e que é calculado como o módulo da soma vetorial da média dos testes de CPU e do teste de enlace de comunicação, ambos normalizados em relação a uma máquina de referência. Finalmente

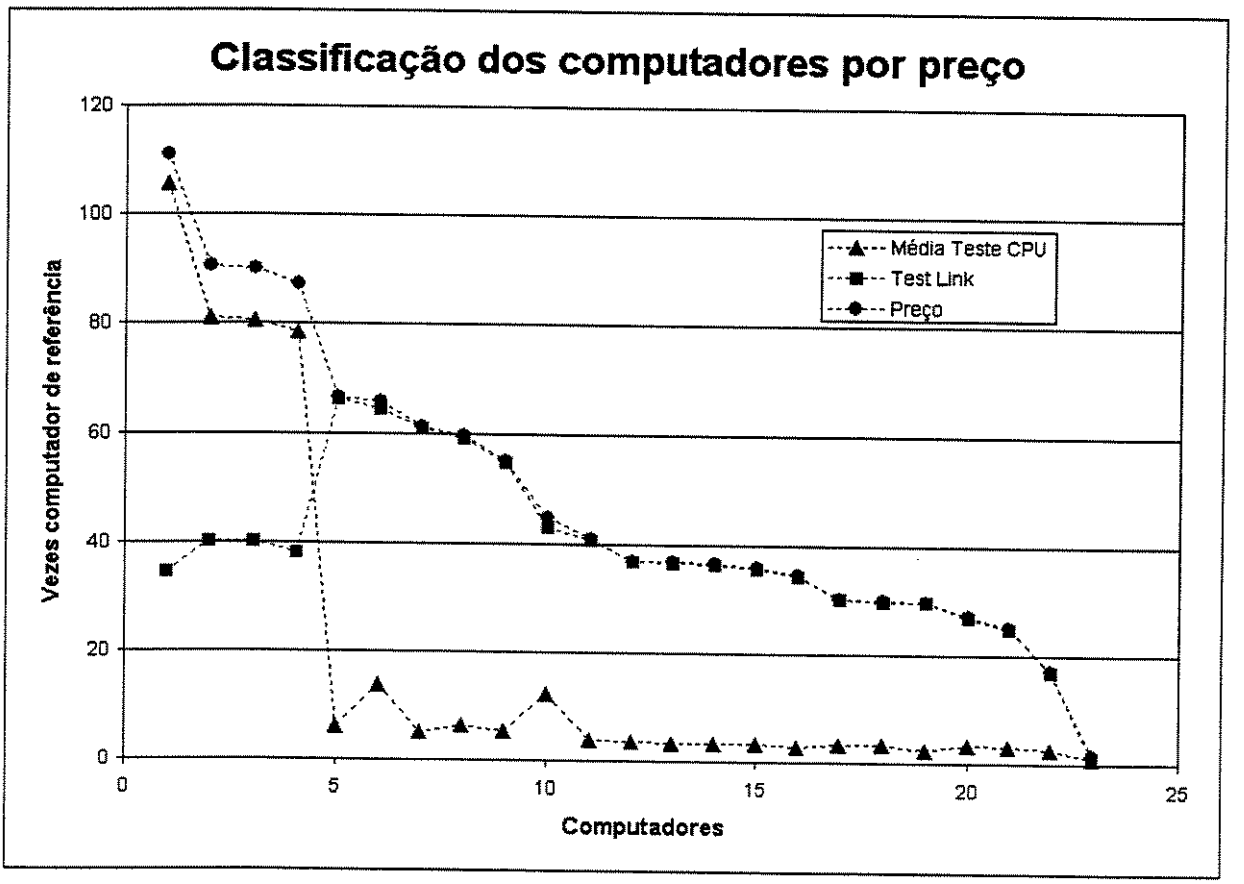

Figura 3.8: Distribuição dos preços dos computadores (preços abaixo de 120).

neste Capítulo foi apresentada a classificação final de alguns computadores testados com o método proposto, a qual será usada para um algoritmo de alocação proposto no capítulo seguinte.

Capítulo 4

Alocação de recursos usando preços

4.1 Introdução

A alocação de recursos em sistemas de processamento massivamente paralelo não é simples. A configuração e as características especiais destes sistemas fazem que as abordagens tradicionais não sejam apropriadas nem factíveis para alcançar uma alocação adequada, já que trabalha-se com quantidades muito grandes de recursos e estes algoritmos são centralizados e baseados em consenso e cooperação. Nestas condições parece que apenas os sistemas que implementam algum tipo de competição podem ser efetivos [FE96]. Existem dois motivos para afirmar isto. Primeiro, a cooperação e a procura do consenso são políticas instáveis e podem não funcionar se algum dos participantes falha na cooperação. Existem algumas técnicas que corrigem estes problemas como *Byzantine Agreement* [LS82, PSL80] mas ao custo de um aumento de complexidade. Os benefícios da concorrência é que esta é inerentemente estável. Se algum dos agentes falha na concorrência, apenas esse resulta prejudicado. O segundo motivo é que algoritmos cooperativos podem não ser práticos em sistemas dinâmicos grandes, já que é difícil cooperar para alcançar um consenso quando o tamanho do sistema aumenta, e as informações necessárias para a alocação podem estar desatualizadas.

Propomos neste capítulo um método de alocação de recursos utilizando preços como parâmetros quantitativos do poder dos computadores que fazem parte do computador massivamente paralelo virtual (MPVC). Neste esquema de alocação de recursos serão usadas

como base as idéias expostas no capítulo anterior, ou seja, a classificação proposta para os computadores e a maneira de cálculo do poder computacional (preço).

4.2 O mercado de recursos ideal

O princípio básico do mercado econômico de recursos computacionais deve ser alocar estes recursos da melhor maneira entre os processos ou aplicações que precisam deles. As características ideais destes mercados devem ser as seguintes.

- Existência de um mecanismo de priorização para a alocação que racionalize o uso dos recursos.
- Possibilidade de tomar decisões certas em cada alocação. No caso de mercados econômicos, esta decisão é baseada no preço.
- Ofertas e pedidos de recursos facilmente interpretados, ou seja, os mecanismos para fazê-los públicos devem ser simples.
- Ajuste dinâmico do mercado para se adaptar às mudanças do ambiente.
- Existência de um mecanismo computacional simples para alocar o recurso a uma oferta.
- Garantia de uma alocação completa para diferentes tipos de requerimento, especialmente em aplicações paralelas que precisam de uma grande quantidade de recursos.
- Tempos de alocação pequenos em comparação com o tempo de processamento das aplicações.

Pode não ser possível alcançar todos estes objetivos simultaneamente, mas eles servem como guia para a definição de métodos de alocação de recursos.

4.3 Requisitos para a alocação

Pelas características dos MPVCs, é necessário atender alguns requisitos para que a alocação possa ser feita. Dentro destes requisitos temos o que trata da escalabilidade, ou seja, a

maneira de fazer com que o sistema continue eficiente enquanto cresce, e o outro que diz respeito à obtenção dos recursos para que as aplicações possam ser executadas.

4.3.1 Divisão dos computadores em grupos

O fato do MPVC considerado estar baseado na Internet, levanta a questão de como organizar os muitos computadores que o compõem, já que a Internet está formada por uma diversidade de topologias de interconexão. Os sistemas mencionados no Capítulo 2 usam diferentes topologias e alguns estão organizados em grupos distribuídos através da rede.

É desejável para a alocação que os computadores participantes do MPVC estejam distribuídos em grupos e que estes grupos estejam conectados logicamente uns com outros usando alguma topologia lógica de conexão como se mostra na Fig 4.1. Estar conectado um com outro logicamente, significa que um computador funcionando como coordenador do grupo[1] conhece o endereço do outro.

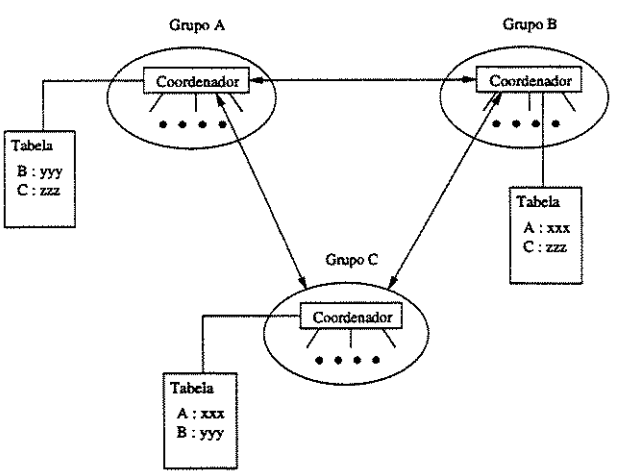

Figura 4.1: Conexão lógica entre grupos.

Não é especificada nenhuma topologia de conexão em particular, mas o ideal seria que cada grupo estivesse conectado com a menor quantidade possível de outros grupos, e que ainda assim a topologia usada permitisse trocar informações entre todos os grupos. Isto se explica pela necessidade de armazenar informações dos computadores e das tarefas atualmente executando neles. À medida que o número de computadores cresce no sistema,

[1]Em cada grupo existe um computador com a função especial de coordenar os outros computadores do grupo. Este computador é chamado de "coordenador"

o número de tarefas também cresce, e como conseqüência disto o número de informações sobre estas. Sendo assim é necessário usar uma maneira escalável de representar esta informação, para que sempre esteja disponível e possa ser rapidamente recuperada.

Devido ao tamanho dos MPVCs, a tendência é agrupar os computadores participantes a fim de diminuir a complexidade do sistema. Alguns sistemas são agrupados de maneira tão trivial como o modelo proposto por [FE96] mostrado na Fig. 4.2. Cada computador conhece os endereços dos outros computadores do mesmo grupo, os identificadores das tarefas que estes estão executando e os preços dos outros computadores. Esta informação deve ser mantida atualizada por todo o sistema. Este tipo de representação não é escalável a uma quantidade grande de computadores, já que à medida que o número de computadores aumenta, também aumentam as tarefas e a memória que se precisa para armazenar estes dados em cada computador. O esforço para manter esta informação sincronizada cresce dramaticamente à medida que novos computadores entram ou saem.

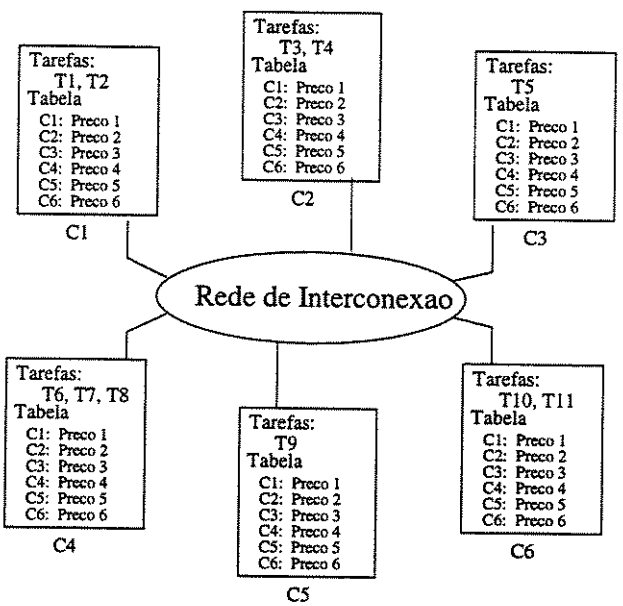

Figura 4.2: Agrupamento dos computadores e representação das informações sobre tarefas para alocação de recursos e balanceamento de carga baseados em modelos econômicos (trabalho afim explicado em 2.4.2).

PopCorn agrupa os computadores por mercados que estão divididos em vários módulos que se encarregam de manipular todo o processo da alocação como foi explicado na Seção 2.4.1. Devido à centralização dos módulos (cada módulo se encarrega de uma função específica como alocação, pagamento, negociação), este agrupamento não é conveniente

para representar a informação pela possibilidade de gerar gargalos no sistema.

A topologia de hipercubo dinâmico virtual (DVH) foi usada por JoiN (Seção 2.4.5), e será adotada na implementação e testes desenvolvidos no Capítulo 5 pelas vantagens que tem demonstrado e por ser uma alternativa interessante para prover escalabilidade.

4.3.2 Orçamento das aplicações e prioridades

Sendo a alocação proposta baseada em preços (cada recurso tem seu preço), é lógico imaginar que estes têm que ser pagos pelas aplicações para que estas possam executar suas tarefas. Isto implica que as aplicações devem ter um orçamento que deve ser usado para este fim. Esta é uma condição indispensável para a alocação, já que parte do dinheiro das aplicações será transferido para as contas dos computadores como forma de pagamento pela execução das tarefas. O orçamento das aplicações é expresso em créditos e deve ser distribuído entre todas as suas tarefas.

Ao pensarmos em dinheiro e compra de recursos, podemos pensar também nos critérios usados na decisão de compra. Os critérios para a compra são baseados em prioridades. Existem duas prioridades para a compra de recursos:

- prioridade baseada em economia de orçamento, na qual a aplicação deseja gastar o mínimo possível do seu orçamento, sem importar o tempo a usar para terminar a execução. Nesta prioridade corre-se o risco de que a aplicação demore muito para ser concluida;
- prioridade baseada em tempo de execução, na qual a aplicação precisa terminar o mais rápido possível, sem importar o orçamento a gastar [2]. O algoritmo proposto na Seção 4.6 garante que as aplicações obtenham os computadores de melhor desempenho de acordo com seu orçamento.

Não existe um critério definido para decidir a quantidade de dinheiro a alocar às aplicações, e isto está fora do escopo desta dissertação. Nos testes usamos quantidades arbitrárias, mas mantendo uma lógica e coerência nos orçamentos. Para os testes foram criadas aplicações que chamamos de "pobres"; que têm um orçamento baixo, e aplicações

[2] logicamente, dentro dos limites do seu orçamento.

ricas, com alto orçamento. O orçamento diz a respeito à aplicação e tem que ser dividido entre todas as tarefas da mesma.

O critério de alocação de orçamento para as aplicações é uma maneira de controlar prioridade de execução. Aplicações com orçamentos altos terão acesso aos computadores mais caros (os mais rápidos) acabando o processamento no menor tempo. Conseqüentemente, a disponibilidade de recursos para as aplicações de baixo orçamento são menores.

4.4 Atualização de preços dos computadores

Na dinâmica de mercado, os preços dos recursos estão variando constantemente. As variações ocorrem pela oferta e demanda como em qualquer economia real, onde ofertas de recursos podem ou não ser absorvidas pela demanda. Nesta interação, os preços se adequam às condições gerais do mercado, aumentando quando a demanda é maior que a oferta, ou diminuindo quando a oferta é maior. Além deste motivo para atualizar os preços, existe um outro que é a carga dos computadores, ou seja, quando muda o número de tarefas executando em um computador, o preço é modificado em função do número atual já que, conforme os computadores adquirem mais trabalho, o poder deles diminui.

A seguir são apresentadas duas maneiras de atualizar os preços, assim como as vantagens e desvantagens de cada uma delas.

4.4.1 Atualização de preços pelo número de tarefas

Esta atualização é baseada apenas no número de tarefas que o computador está executando, ou seja, impõe uma relação direta entre o preço atual do computador e o número de tarefas que este computador executa. A referência para se calcular o preço atual de um computador é o preço inicial obtido no teste, quando o computador entrou no sistema. A fórmula que calcula o preço atual é mostrada na Eq. 4.1

$$LoadPrice_i(N_E) = \frac{InitialPrice_i}{N_E + 1} \tag{4.1}$$

onde:
$LoadPrice_i(N_E)$: preço do i-ésimo computador quando tem N_E tarefas executando.

$InitialPrice_i$: preço do i-ésimo computador obtido no teste benchmark. Representa o preço quando o computador tem 0 tarefas executando nele.
N_E : número de tarefas atualmente executando no computador.

O termo +1 é usado para calcular o preço "futuro", com mais uma tarefa executando (como se a tarefa que vai entrar no computador já estivesse executando) e que seria o que uma nova tarefa iria pagar. Por exemplo, se o preço inicial de um computador for 10 créditos por tarefa e existir uma tarefa executando nele, então o preço mostrado para a tarefa que pretende executar nele seria $LoadPrice_2 = \frac{10}{1+1}$ ou $LoadPrice_2 = 5$, ou seja a metade do preço inicial (calculado como se houvesse duas tarefas executando, sendo que só existe uma no computador). Assim evitamos o problema da "propaganda enganosa", que seria o caso em que o preço mostrado apenas tem em conta as tarefas em execução no momento. Isto seria enganoso porque quando a nova tarefa fosse executar, ela só obteria o poder do computador com duas tarefas executando, sendo que ela pagou por ter apenas uma (isso é o que foi mostrado para ela).

A vantagem de usar este tipo de atualização é o cálculo simples e rápido do preço (pouco esforço computacional), além de que não se precisa armazenar informação adicional. Mas isto traz como conseqüências uma variação muito rígida dos preços e uma disponibilidade dos computadores apenas para as aplicações que tenham um orçamento que possa pagá-los. Neste tipo de atualização não é garantida a execução de aplicações com baixo orçamento; estas só serão executadas quando o sistema ficar carregado e os preços caírem como conseqüência da carga.

4.4.2 Atualização de preços pelas forças de mercado

Além da atualização do preço em função do número de tarefas (Eq. 4.1), neste tipo de atualização se tem em conta outros fatores.

Para o caso do mercado de recursos, esta atualização ocorre também pela dinâmica da oferta e demanda, ou seja, se há mais oferta os preços devem diminuir, enquanto que se há maior demanda os preços sobem. O mecanismo para implementar esta situação de mercado é fazer que os computadores mudem de preço após várias tentativas de alocação sem sucesso, ou quando a alocação é freqüente. Por exemplo, se temos um computador que não foi escolhido na alocação mesmo depois de V_P pedidos de alocação de tarefas, ele deve

diminuir seu preço por que isto pode ser uma indicação de que este é muito alto ou de que a oferta supera a demanda. Se um computador é escolhido V_A vezes seguidas para receber aplicações, ele deve aumentar seu preço, já que isto pode ser um indicador que tal preço é muito barato ou que a demanda supera a oferta. A Eq. 4.2 mostra a variação do preço do computador pela oferta e demanda. Os valores destes parâmetros (V_P e V_A) dependem do dinamismo que se queira dar ao mercado, porque valores altos fazem que os preços do mercado mudem mais lentamente, acontecendo o contrario para valores baixos. Para o fator de atualização de preços ($F_{Atual.}$) foi usado 10 % com bons resultados na alocação.

$$MarketPrice_i = LoadPrice_i(N_E) * ((1 - F_{Atual.})^{(nLost-nWon)}) \tag{4.2}$$

onde:
$MarketPrice_i$: preço para o i-ésimo computador, definido pelas forças do mercado.
$LoadPrice_i(N_E)$: preço para o i-ésimo computador, definido pela carga do computador (calculado pela Eq. 4.1).
$F_{Atual.}$: fator de atualização do preço do computador pelas forças do mercado (entre 0 e 1).
$nLost$: contador incrementado cada vez que acontecem V_P tentativas seguidas de alocação sem sucesso.
$nWon$: contador incrementado cada vez que acontecem V_A tentativas seguidas de alocação com sucesso.

Ná equação 4.2 observa-se que os valores de $nLost$ e $nWon$ determinam o nível dos preços do mercado (para $nLost > nWon$, $MarketPrice$ é mais baixo que $LoadPrice$; para $nWon > nLost$, $MarketPrice$ é mais alto que $LoadPrice$). Os preços definidos pelo mercado seguem a tendência histórica dos valores de $nWon$ e $nLost$, e voltam aos níveis iniciais se estes termos alcançam os mesmos valores e se anulam.

Assim, o mercado sempre estará em um nível no qual os preços refletem a oferta e demanda existente, e se mantêm nesse estado até que as condições mudem (ou seja, aumentem as tentativas seguidas com sucesso ou sem sucesso).

A vantagem deste tipo de atualização é que ela oferece um mercado de preços muito dinâmico, onde preços estão aumentando e diminuindo devido à oferta e demanda, dando maiores possibilidades às aplicações para obterem os recursos que precisam, e aos computadores de obter os melhores preços pelo serviço. Além destas vantagens, neste método de atualização do preço existe uma possibilidade maior de que aplicações com pequenos

orçamentos sejam executadas. Se não há aplicações com grande orçamento demandando recursos, os preços podem cair por força do mercado, criando assim a possibilidade de que as aplicações que têm um orçamento pequeno sejam executadas.

Uma desvantagem deste método é que se precisa de um maior esforço computacional para controlar as variações do mercado, ou seja, contabilizar as vezes que um computador é contatado para alocação mas não há sucesso, e o número de vezes seguidas que em um computador são alocadas tarefas.

Embora o método de atualização de preços pelo mercado pareça ideal para qualquer tipo de sistema, um algoritmo de alocação de recursos para sistemas de processamento massivamente paralelo baseado na Internet deve ter em conta as duas maneiras de atualização: uma para estimar a carga do computador de maneira mais precisa, e outra para oferecer a possibilidade de qualquer aplicação poder ser executada, sem importar muito o orçamento que tenha.

No algoritmo apresentado na seção 4.6 são usadas as duas maneiras de atualização, oferecendo um esquema híbrido de alocação de recursos. Para o cálculo da carga do computador é usado a atualização pelo número de tarefas, enquanto que para a alocação é usada a atualização pelas forças do mercado (que é o preço que as tarefas deverão pagar para executar).

Manipulação do mercado

Existe a possibilidade de que o mercado econômico de recursos seja manipulado com a submissão de aplicações fantasmas[3], para fazer que os preços aumentem ou diminuam.

Pode haver uma pessoa interessada em executar uma aplicação nos computadores mais rápidos, mas pagando um preço baixo. Para isso, ela submeteria aplicações com baixo orçamento até que os preços fossem desvalorizados ao nível que ela deseja. Assim, mesmo tendo gasto créditos com as aplicações fantasmas que submeteu, ela obtém um ganho pelo uso dos computadores mais rápidos pagando preço menor.

O outro extremo seria um grupo de donos de computadores aplicar uma tática para

[3]Chama-se aplicação fantasma neste contexto àquelas cuja única finalidade é influenciar os preços dos recursos no sistema.

fazer que seus computadores fossem mais valorizados e assim obter um ganho maior. Esta tática consistiria em submeter muitas aplicações fantasmas com alto orçamento a fim de fazer que os preços aumentassem. Os donos dos computadores não teriam prejuízo, já que boa parte dos créditos usados pelas aplicações fantasmas poderiam ser ganhos pelos seus próprios computadores e os preços no mercado aumentariam, gerando maiores ganhos vindos de outras aplicações reais que precisam ser executadas.

Como se observa, existe a possibilidade de que o mercado seja manipulado a fim de se obter benefícios pessoais. Para solucionar parcialmente o problema pode-se implementar limites inferior e superior na variação dos preços (atualmente existe um limite inferior que está determinado pelo estado sobrecarregado do computador, ou seja, o estado equivalente ao do computador de referência executando mais de uma tarefa). Além dos limites, poderia se pensar também em uma espécie de coordenador de coordenadores, encarregado de avaliar esporadicamente os preços a fim de adotar uma política para estabilizá-los se for necessário (funcionando como um banco central que regula a economia).

4.5 O problema da prioridade das aplicações

Se um computador está vazio (sem nenhuma tarefa executando) e uma tarefa precisa ser executada nele, ela paga o preço combinado para usufruir de forma exclusiva do poder de processamento do computador. Quando chega outra tarefa com intenção de executar no mesmo computador, ela paga para usufruir do computador ao preço combinado mas compartindo-o com outra tarefa. Neste esquema de alocação, existe uma injustiça com a primeira tarefa, já que provavelmente ela pagou uma quantidade maior para executar sozinha no computador, enquanto a tarefa que chegou depois pagou menos (por que o preço está em função do número de tarefas) e agora compartilha o mesmo processador.

A solução para este problema deve ser baseada em alocação de prioridades de execução para as tarefas. A prioridade de uma tarefa diz respeito à frequência com que ela toma posse do processador e deve ser proporcional ao preço pago pela execução. Assim, tarefas que pagassem mais teriam sempre maior prioridade de execução e as que pagassem menos teriam um menor tempo de execução alocado para elas.

Esta discussão está ligada ao conceito de qualidade de serviço (QoS) usado em redes de computadores. Neste esquema as tarefas estão classificadas de acordo ao orçamento.

Assim, para cada classe é alocada uma percentagem de tempo da CPU, que seria usada só pelas tarefas dessa classe. Cada classe deveria ter um número máximo de tarefas para fornecer o mínimo de "QoS" para a classe. Por exemplo se as tarefas forem classificadas em 3 classes, para a classe de maior prioridade seria alocada uma percentagem maior de CPU (70 % por exemplo), para a segunda classe uma percentagem menor (20 %), enquanto que a última classe ficaria com o que sobrasse (10 %).

Em termos de programação, o esquema de prioridades pode ser implementado usando as prioridades para os "Threads" fornecidas pela maioria de linguagens multithread existentes, como o Java. Mas há várias outras soluções propostas para QoS em redes que podem ser adotadas para implementar um mecanismo de priorização de recursos dentro do contexto deste trabalho. Porém, esta discussão será deixada como sugestão para um trabalho futuro.

4.6 Alocação de recursos computacionais usando RA-AP (Resource Allocation Algorithm using Prices)

Nesta seção é apresentado o algoritmo de alocação de recursos computacionais em sistemas de processamento massivamente paralelo virtuais baseados na Internet[4]. Este algoritmo usa o método de classificação de computadores proposto no Capítulo 3, as maneiras de atualização de preço propostas nas seções 4.4.1 e 4.4.2, e está baseado na plataforma JoiN apresentada na Seção 2.4.5.

O algoritmo se inicia quando um computador qualquer na Internet conecta-se ao Web site do MPVC e declara sua disponibilidade como mostrado na Fig. 3.2. No processo de ingresso no MPVC, o computador (que chamaremos de trabalhador) faz os testes e se registra no coordenador do grupo ao qual pertencerá. O coordenador tem uma lista de computadores, que estão ordenados em ordem decrescente de preços. Este processo é repetido por todos os computadores que desejam formar parte do MPVC, e o poder computacional é balanceado entre os grupos conforme descrito em 4.7.1. O processo de alocação é mostrado na Fig. 4.3 e explicado a seguir, passo a passo.

[4]As idéias iniciais deste algoritmo foram os algoritmos de alocação de recurso centralizado e distribuído propostos em [PM99]

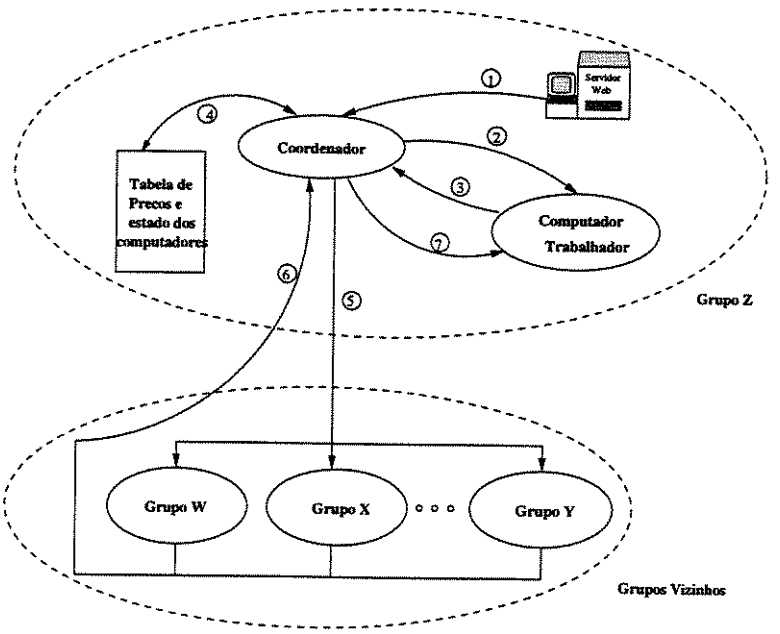

Figura 4.3: Algoritmo de Alocação de Recursos usando Preços (Resource Allocation Algorithm using Prices - RAAP).

1. O algoritmo de alocação começa quando uma aplicação (que também é chamada de tarefa "mãe") é descarregada do servidor para um coordenador de um grupo que não esteja sobrecarregado (como definido em 3.4). Caso todos os grupos estejam sobrecarregados é enviada uma mensagem para a aplicação "mãe", indicando a impossibilidade da alocação.

2. O coordenador procura um computador não sobrecarregado de seu grupo e envia a aplicação para ser executada neste computador. O coordenador procura na sua tabela, entre os computadores mais rápidos que a aplicação pode pagar, aquele com menor carga para enviar a aplicação "mãe". Se todos os computadores estiverem sobrecarregados, o coordenador envia o pedido de criação desta aplicação para outro coordenador do seu *buddy set* (explicado em 2.4.5).

3. Quando a tarefa "mãe" precisar criar N tarefas, esta fará um pedido, indicando o número de tarefas a criar, a prioridade da aplicação e o orçamento por tarefa[5] que pode ser pago por execução, e envia este pedido para o coordenador.

4. O coordenador recebe o pedido, analisa-o e, dependendo da prioridade da aplicação,

[5]O orçamento da aplicação é dividido entre o número de tarefas a criar.

procura por computadores na tabela. Se a prioridade da aplicação é economia, o coordenador procura pelos computadores mais baratos da lista; se é tempo de execução procura pelos computadores mais caros que podem ser pagos pelo orçamento. As tarefas são alocadas entre os computadores de maneira eqüitativa para balancear a carga entre estes computadores (a distribuição será explicada detalhadamente em 4.7.2).

5. As tarefas são alocadas até que os computadores do grupo fiquem sobrecarregados (não conseguem alocar mais nenhuma tarefa). Com as que faltarem para completar a alocação é criado um novo pedido e enviado para um coordenador do *buddy set*. Quando um pedido de criação de tarefas chega de outro coordenador, ele é tratado quase da mesma maneira que quando chega um pedido de uma aplicação. A diferença é que somente são alocadas as tarefas que forem possíveis, sem pedir para outros coordenadores a criação de tarefas que não puderem ser criadas.

6. Se o coordenador faz o pedido de criação de tarefas para outro coordenador, ele espera pelos identificadores das tarefas criadas. Se não forem criadas todas as tarefas, tenta-se com outros coordenadores. Se mesmo assim não se conseguir alocar todas as tarefas, tenta-se de novo no mesmo grupo do coordenador, voltando a pedir aos coordenadores do *buddy set* (neste intervalo de tempo o estado dos computadores do grupo pode ter mudado). Esta repetição ocorre no máximo T vezes.

7. O coordenador envia a resposta da alocação para a tarefa mãe com os identificadores das tarefas criadas. Se após o passo 6 ter ocorrido T vezes não se conseguir criar todas as tarefas, é enviada uma mensagem para a tarefa mãe indicando que a alocação não foi completada, além dos identificadores das tarefas criadas. Dependendo da aplicação a tarefa mãe pode decidir eliminar as tarefas criadas e tentar novamente mais tarde, ou aproveitar as que foram criadas em número abaixo do solicitado.

Os cálculos neste algoritmo são simples, não requerendo um grande esforço computacional. As mensagens necessárias são poucas: há mensagens somente para fazer o pedido da criação e para voltar o resultado à tarefa solicitante. O número de mensagens para manter atualizada a informação sobre o estado dos grupos também é pequeno, já que, como foi explicado em 3.4, a variação do estado destes não é freqüente. A quantidade pequena de mensagens para o algoritmo funcionar constitui-se na principal vantagem do mesmo, já que a Internet é lenta e algoritmos com uma grande quantidade de mensagens não são adequados para sistemas MPVCs baseados na mesma.

Outro ponto favorável do algoritmo é que, exceto no caso em que todos os grupos fiquem no estado sobrecarregado, é maior a probabilidade de execução de aplicações com orçamentos menores. Obviamente o volume de orçamento define a prioridade para que algumas aplicações possam usar computadores mais rápidos que outros.

A principal desvantagem do algoritmo é que ele depende de um ponto centralizado para armazenar os preços dos recursos e fazer a alocação, apesar do sistema geral de alocação estar distribuído pelos grupos do MPVC. Além disso, o coordenador e o processo alocador (o que se encarrega de fazer a alocação) não precisam estar no mesmo computador, o que pode diminuir os efeitos da centralização.

4.7 Balanceamento e carga

Como o MPVC é formado por grupos e os grupos formados por computadores, o balanceamento da carga deve ser feito nestes dois níveis. A seguir é mostrado um esquema proposto para equilibrar a carga nos grupos e entre os grupos.

4.7.1 Balanceamento de carga no sistema (entre os grupos)

O MPVC está organizado em grupos, e deve existir um critério para alocar os computadores aos grupos a fim de equilibrar a carga entre estes. Quando um computador entra no sistema (entra no Web site e declara a sua disponibilidade), é descarregado o programa de teste para medir seu poder computacional. Além disso é indicado ao computador o grupo ao qual pertencerá, que é onde terá que se registrar depois dos testes serem feitos. A decisão sobre qual grupo este computador deve pertencer está baseada na distribuição eqüitativa de computadores entre grupos. Por exemplo, se existirem 4 grupos, os computadores são alocados um a um em ordem seqüencial para os grupos 1, 2, 3 e 4, repetindo à medida que novos computadores chegam.

Como o sistema é dinâmico, ou seja, os computadores podem estar entrando e saindo continuamente, é possível implementar um mecanismo mais eficiente de alocação dos computadores aos grupos baseado no poder computacional destes (preço inicial). Os computadores seriam distribuídos para os grupos baseados no preço, de maneira a ter-se em todos os grupos aproximadamente o mesmo poder computacional. O mecanismo de imple-

mentação é simples: somente se precisa de uma tabela com uma entrada por grupo, onde se coloca o poder computacional alocado a cada grupo (ou seja, a soma dos preços). Cada novo computador é alocado ao grupo que tem a menor soma. Quando um computador sai de um grupo, seu preço é substraído do preço total do grupo. Em termos de execução o esforço é mínimo, já que só é necessário percorrer a lista dos grupos.

A desvantagem de usar este mecanismo é que os computadores que ingressam no MPVC deverão informar o seu preço para o Web site, para que este lhes indique o grupo ao qual pertencerão. Da mesma forma, quando um computador deixa o sistema, e este fato é percebido pelo coordenador do grupo, que deve informar ao Web site para que este atualize o poder computacional do grupo.

4.7.2 Balanceamento de carga dentro do grupo

Os coordenadores são os responsáveis pela alocação de tarefas dentro de um grupo. Para isto eles mantêm um registro de todos os computadores com os preços $LoadPrice$ e $MarketPrice$ de cada um deles, além da carga atual em número de tarefas. O balanceamento de carga no grupo acontece pela dinâmica do mercado, onde as aplicações estão competindo pelos recursos dos computadores.

O preço $MarketPrice$ é usado para fazer a escolha dos computadores onde as tarefas serão alocadas, enquanto o preço $LoadPrice$ é usado para calcular o número de tarefas a alocar em cada um dos computadores.

O Algoritmo de alocação RAAP descrito na Seção 4.6 recebe um pedido de criação de tarefas (e alocação nos computadores) e escolhe uma lista de computadores que a aplicação pode pagar (comparando os créditos a serem pagos por tarefa com o $MarketPrice$ dos computadores).

A escolha nesta lista dos computadores que efetivamente receberão tarefas é feita calculando-se o número de créditos a serem pagos pela aplicação pela execução de todas as tarefas ($ApplicationCredits$). Este valor é dado pelo número de tarefas a criar na aplicação (N_T) multiplicado pelo preço a ser pago por cada tarefa (B). A procura na lista é feita somando-se os preços atuais calculados pela Eq. 4.1 ($LoadPrice(N_E)$) dos computadores,

até o número de créditos requeridos (*ApplicationCredits*) [6], ou a lista acabar.

Se a soma de créditos obtidos da lista de computadores escolhidos (*AvailableCredits*) for maior que uma percentagem (*PerformanceFactor*) do número de créditos requeridos pela aplicação (*ApplicationCredits*), então as tarefas são alocadas aos computadores desta lista; senão a alocação é cancelada. Isto é feito para garantir um mínimo de desempenho para a aplicação, assim como para forçar a distribuição das tarefas entre vários computadores.

A alocação de tarefas para cada computador é feita de maneira proporcional ao poder computacional de cada um (usando o $LoadPrice(N_E)$). Se existem dois computadores com $LoadPrice(N_E)$ igual a 10 e 5 créditos por tarefa, então para o computador de preço 10 será alocado o dobro de tarefas que para o computador de preço 5. Como se assume que as tarefas são equivalentes, os computadores terão alocada a mesma carga em relação a seu poder de cômputo.

Para determinar o número de tarefas a alocar por computador é usada a Eq. 4.3. Este número depende do *LoadPrice*, do total de créditos disponíveis na lista escolhida e do número de tarefas totais a criar.

$$N_i = \frac{N_T * LoadPrice_i(N_E)}{AvailableCredits} \tag{4.3}$$

onde:
N_i : número de tarefas a alocar no computador i,
N_T : número de tarefas requeridas pela aplicação,
$LoadPrice_i(N_E)$: preço atual do computador i dado pelo número de tarefas N_E (Eq. 4.1).
AvailableCredits : soma de créditos da lista de computadores escolhidos.

Um exemplo de alocação para uma aplicação que precisa criar 10 tarefas, com 30 créditos por tarefa é mostrado na Fig. 4.4.

No exemplo da Fig. 4.4, deseja-se a alocação de uma aplicação que requer 10 tarefas e pode pagar 30 créditos por tarefa (B) e existe uma tabela de computadores no grupo que armazena os identificadores dos computadores assim como os preços *LoadPrice* e *MarketPrice*. Nesta tabela existem 12 computadores (Computer1 até Computer12).

[6] Este critério é adotado para alocar inicialmente uma tarefa por computador.

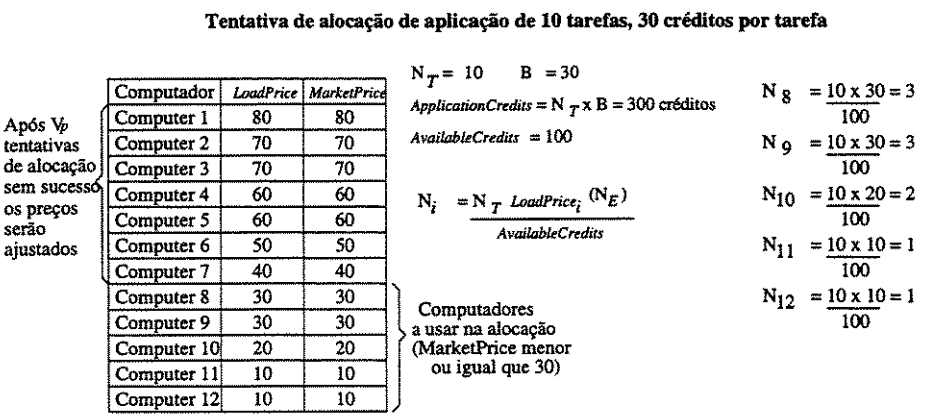

Figura 4.4: Exemplo de alocação de uma aplicação com 10 tarefas, com 30 créditos por tarefa (Prioridade por tempo de execução).

Para a alocação, o processo alocador calcula *ApplicationCredits* que será a quantidade de créditos necessários para as tarefas e que devem ser fornecidos pelos computadores ($10 * 30 = 300$). Após isto ele escolhe os computadores cujo *MarketPrice* seja menor ou igual ao orçamento por tarefa, até completar a quantidade de *ApplicationCredits* requeridos ou a lista acabar. Em seguida, o processo alocador obtém o valor *AvailableCredits* somando os valores LoadPrice desde o primeiro até o ultimo computador escolhido. Neste caso são escolhidos os computadores Computer8 até Computer12 e *AvailableCredits* se torna 100. Para cada um dos computadores é calculado o número de tarefas a ser criado com a Eq. 4.3. Desta maneira em cada computador é alocado um número de tarefas proporcional à sua carga atual (dado por *LoadPrice*). Para o caso dos computadores que não foram escolhidos (Computer1 até Computer7) pode ser feito um ajuste no preço do mercado (*MarketPrice*) dependendo do parâmetro V_P.

No exemplo da Fig. 4.5 é mostrado o caso de prioridade por economia, ou seja, a escolha dos computadores onde será feita a alocação é a partir dos mais baratos.

Neste exemplo se observa que a lista escolhida é de Computer22 até o Computer9, já que a soma de *LoadPrice* destes computadores é igual ao valor de *ApplicationCredits* (a lista não acaba). Como no processo de escolha ainda sobraram computadores que a aplicação poderia pagar (Computer8) o valor de *AvailableCredits* é igual a *ApplicationCredits*. Em seguida é calculado o número de tarefas que serão criadas em cada computador. Como o número de tarefas pode ser um valor fracionário, este será arredondado para o número inteiro superior a fim de permitir que os computadores mais baratos sejam usados. Por

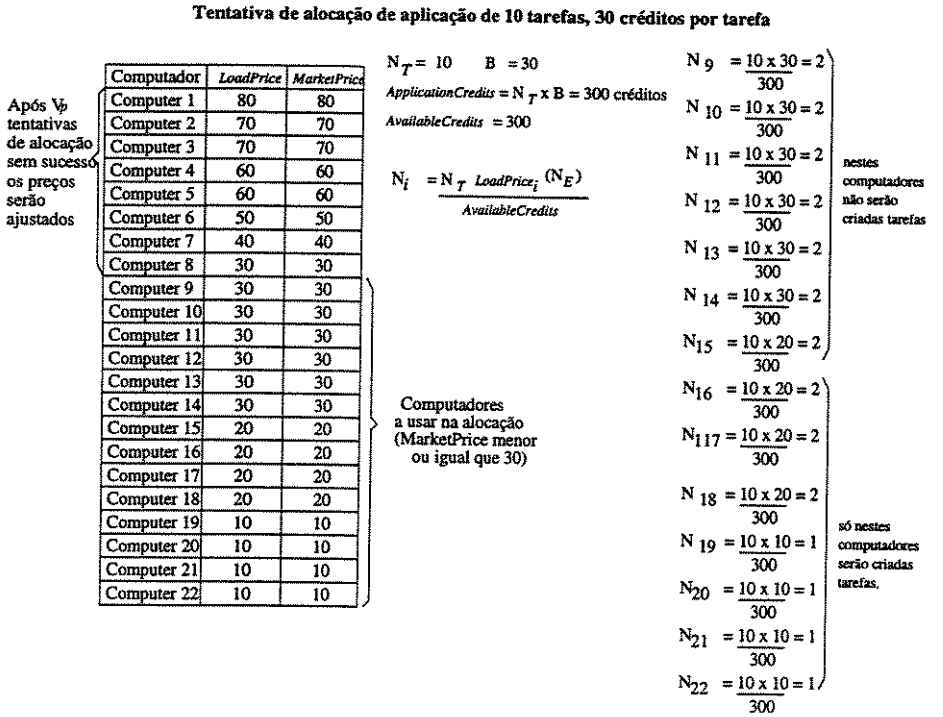

Figura 4.5: Exemplo de alocação de uma aplicação com 10 tarefas, com 30 créditos por tarefa (prioridade por economia).

este motivo em alguns computadores da lista escolhida não será criada nenhuma tarefa (só serão criadas tarefas nos computadores Computer22 até Computer16).

Além do esquema de balanceamento de carga mostrado, é possível ainda usar um critério extra para a alocação no caso de existirem muitos computadores com $MarketPrice$ igual [7]. O critério usado é de escolher os computadores que estejam menos carregados, ou seja, aqueles cujo $LoadPrice$ é menor. Poderia ser considerado também um mecanismo que permitisse a otimização da alocação, mas isto será deixado para um trabalho futuro.

A procura na lista até a alocação das tarefas nos computadores é feita em diversas iterações. Cada iteração ocorre independentemente da alocação ser bem sucedida ou não, e será usada como unidade de medida indireta do tempo para apresentar os resultados do RAAP no Capítulo 5.

[7] A possibilidade de que isto aconteça é muito pequena, mas existe.

4.8 Tolerância às falhas.

Como o algoritmo está projetado para ambientes heterogêneos, a maioria dos problemas que podem fazer o algoritmo não funcionar adequadamente são derivados das vulnerabilidades que estes ambientes apresentam, como problemas nas comunicações e falhas em elementos importantes do MPVC.

Alguns dos problemas que podem acontecer e que precisam ser tratados são mostrados a seguir.

- O coordenador some. Por algum motivo o coordenador fica fora do ar, e perde-se a tabela de preços dos computadores.
- O preço de um computador não é gravado na tabela de precos do coordenador.
- Um pedido enviado por uma aplicação para um coordenador some.
- Os pedidos de um coordenador para outros coordenadores somem.
- As respostas da alocação somem.
- O preço de um computador é armazenado incorretamente.
- São submetidas aplicações oferecendo uma certa quantidade de créditos, mas elas não têm os créditos para pagar pela sua execução.

Grande parte destas questões é resolvida pela implementação dos mecanismos básicos de tolerância a falhas do MPVC e deverá ser tratada em conjunto com tais mecanismos.

4.9 Conclusões

Neste capítulo foram explicadas as condições básicas para a alocação de recursos baseada em preços, e as maneiras de atualização de preços numa economia virtual. Também foram apresentados os mecanismos de balanceamento de carga, que é feito em dois níveis: entre grupos e dentro do grupo. Também foi apresentado o algoritmo RAAP (Resource Allocation Algorithm using Prices) que, pelas vantagens expostas (maior probabilidade de

alocação de recursos às aplicações, poucas mensagens trafegadas pela rede), apresenta-se como uma alternativa viável para a alocação de recursos em sistemas de processamento massivamente paralelo baseados na Internet, tendo em conta suas restrições. Os resultados da implementação e testes serão analisados no capítulo seguinte.

Capítulo 5

Implementação do Algoritmo de Alocação RAAP

Para testar a viabilidade das idéias expostas nos capítulos anteriores, foi implementado um simulador usando a linguagem de programação Java e programação concorrente. O simulador foi construído sobre o MPVC JoiN (Seção 2.4.5), que é um sistema para processamento massivamente paralelo na Internet. O programa simulador usa todos os recursos oferecidos pelo JoiN, desde comunicações até criação de tarefas, passando pela criação de grupos e definição de coordenadores.

Embora o JoiN na versão atual (versão 0.1) só tenha um grupo, o simulador do algoritmo RAAP está projetado para tirar proveito de todos os grupos que JoiN venha a implementar.

5.1 Suposições sobre as tarefas

O algoritmo de alocação de recursos usando preços (RAAP) se baseia na suposição de que todas as tarefas são iguais, ou seja, que precisam do mesmo poder computacional para executar. Isto é usado na atualização de preços do RAAP que usa o número de tarefas, sem ter em conta os requerimentos de processamento de cada uma.

O motivo para esta suposição é que não existe uma maneira simples de determinar a

quantidade de processamento necessário para a execução de uma tarefa. A maneira mais simples de saber tal quantidade em uma tarefa é executando-a, mas isto não é prático; outra maneira é revisando o código da tarefa, mas isto também não é viável.

5.2 Simulação do RAAP

No simulador, as aplicações são alocadas por um mesmo processo alocador e compartilham os computadores em um mercado de recursos.

Define-se como "Aplicação alocada" aquela que já teve todas as suas tarefas alocadas nos computadores.

O programa desenvolvido foi testado simulando a alocação de 10 aplicações com orçamento variado, usando 50 computadores e com preços de computadores entre 10 e 50 créditos por execução de tarefa. Estas situações simuladas são ideais, ou seja, são situações não reais mas que servem para analisar o funcionamento do algoritmo.

O simulador foi testado em três situações de início da alocação (Fig. 5.1).

Início Simultâneo: as aplicações começam a alocação de tarefas simultaneamente, ou seja, são alocadas concorrentemente.

Início Aleatório: as aplicações começam a alocação de tarefas de maneira aleatória. As aplicações são alocadas conforme vão sendo submetidas ao MPVC (os intervalos de tempo entre submissão de aplicações não é previsível).

Início Sequencial: as aplicações são alocadas sequencialmente, ou seja, a alocação de tarefas de uma aplicação começa após o término da alocação de outra. Uma nova alocação de tarefas não começa se a anterior ainda não terminou.

Cada situação descrita acima, foi testada tendo aplicações com orçamento alto, médio e baixo. O orçamento alto foi de 50 créditos, o orçamento médio foi de 10 créditos e o baixo de 1 crédito por execução de tarefa. O número de tarefas por aplicação foi classificado em 4 níveis: muito alto, alto, médio e baixo. O valor considerado foi de 150 tarefas por aplicação para muito alto, 100 para aplicações com número de tarefas alto, 50 para aplicações com número médio de tarefas e 10 para as aplicações com número baixo de tarefas.

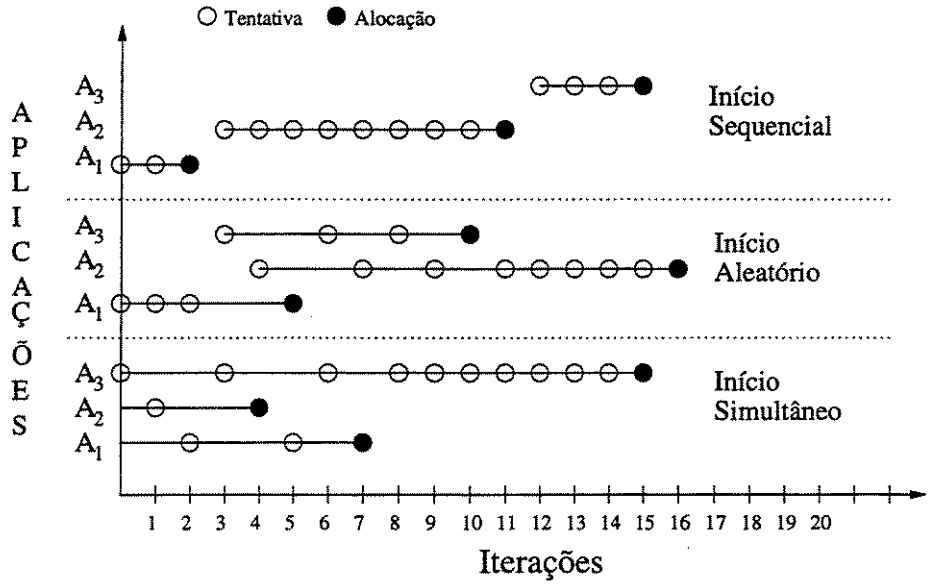

Figura 5.1: Situações de início da alocação.

Para a simulação também foi considerado o número de tentativas de alocação por aplicação (parâmetro T explicado na Seção 4.6), ou seja variou-se o valor usado como número máximo de tentativas que o coordenador faz procurando por computadores para a aplicação. Isto é feito para determinar o impacto deste fator na alocação, já que quando uma tentativa de alocação não tem sucesso ocorre uma depreciação no mercado. Para a simulação foram usado três valores de T: alto(100), médio (50) e baixo (10).

Os outros parâmetros usados na simulação são :

- Fator de desempenho mínimo para a aplicação: $PerformanceFactor = 20\%$.
- Fator de atualização dos preços: $F_{Atual.} = 10\%$.
- Número de tentativas consecutivas de alocação para mudar o preço: $V_P = V_A = 3$.

Os valores considerados para os testes são relativos e podem mudar. Os mesmos foram escolhidos de forma a abranger a maioria das situações que o processo de alocação encontrará na prática e de forma a possuir uma relação entre os valores absolutos que seja fácil de visualizar e analisar.

Uma árvore parcial das situações simuladas é mostrada na Fig. 5.2.

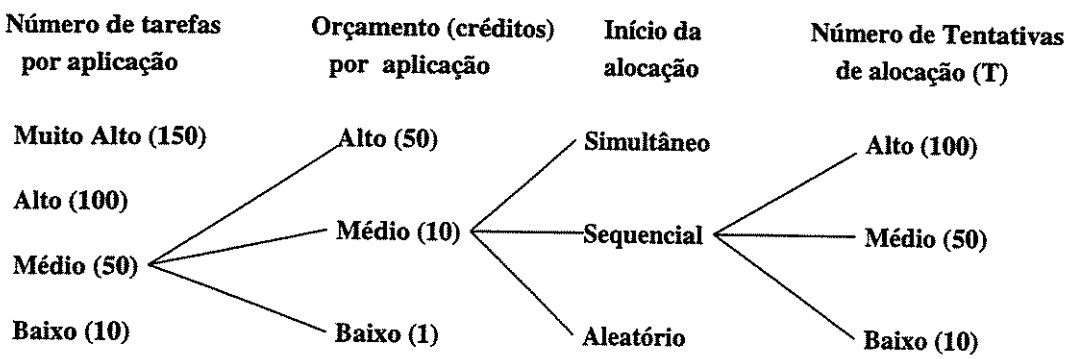

Figura 5.2: Árvore parcial das situações simuladas no algoritmo RAAP.

Foram simuladas todas as situações resultantes das combinações dos valores na árvore. Por exemplo, começando de esquerda para direita, uma das simulações foi: número de tarefas por aplicação *alto* com um orçamento por aplicação *médio* com início de alocação *aleatório* e com um número baixo de tentativas de alocação por aplicação.

As classes usadas no programa simulador são descritas no anexo A, e o esquema de iteração dos elementos do programa são mostrados na Fig. 5.3. A configuração do ambiente JoiN e do simulador é explicado no anexo B.

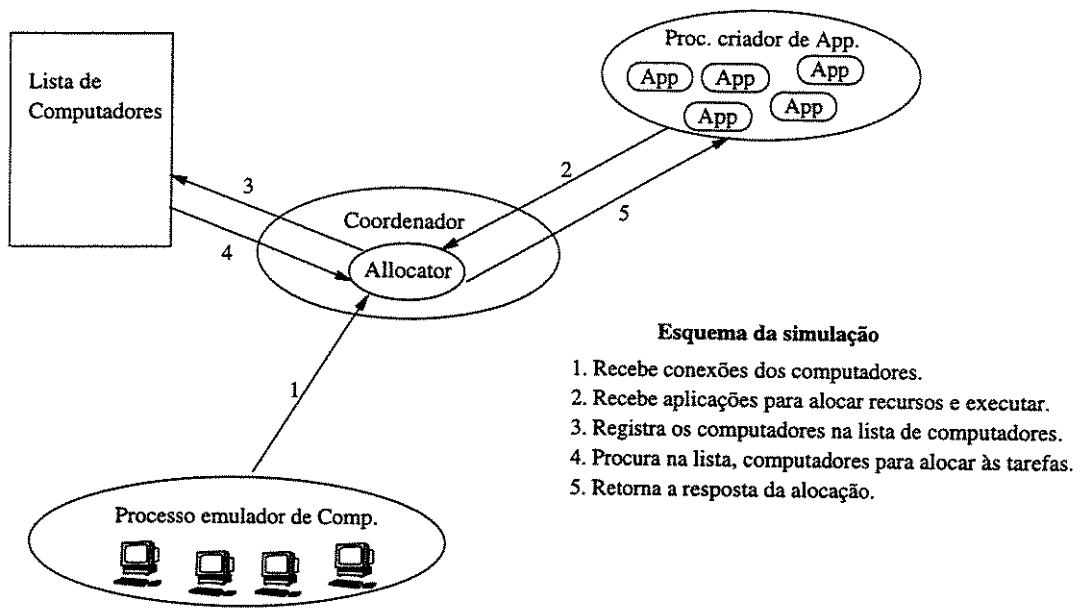

Figura 5.3: Iteração dos elementos do simulador do algoritmo RAAP.

Na figura 5.3 existe um processo encarregado de emular os computadores que fariam

parte do MPVC, um processo que cria as aplicações a serem submetidos para execução e um processo encarregado da alocação (Allocator). Emular um computador em nosso contexto significa gerar a informação de um computador virtual que será armazenada pelo processo alocador, e que será usado na alocação. Criar as aplicações significa gerar os requerimentos de aplicações paralelas que serão submetidas ao processo alocador para serem servidas. Os processos criadores de computadores e de aplicações são módulos independentes e podem ser executados em qualquer computador do MPVC, enquanto o processo alocador (allocator) é executado no coordenador.

5.3 Resultados

Os resultados foram analisados nas várias situações em que o simulador foi testado. Os valores usados como muito alto, alto, médio e baixo são relativos, mas servem para fazer uma análise destes valores e suas variações sobre a alocação.

Nesta seção é mostrado o comportamento da variação dos preços, com uma análise de cada um dos motivos pelos quais o preço do computador é atualizado (pelo número de tarefas executando e pelas forças do mercado).

As Figs. 5.4, 5.5 e 5.6 mostram os resultados obtidos nas simulações para cada uma das situações testadas. O conteúdo de cada tabela se refere a um número de tentativas diferente. A tabela da Fig. 5.4 mostra os resultados quando o número de tentativas de alocação por aplicação é 10, a tabela da Fig. 5.5 mostra os resultados quando esse valor é 50 e a da Fig. 5.6 quando ele é 100.

Cada linha da tabela mostra os resultados da alocação de tarefa para as 10 aplicações submetidas em uma execução do simulador.

As tabelas estão divididas em várias colunas. A primeira coluna indica o número do caso avaliado. As três seguintes colunas se referem às variações dos parâmetros de testes. Da coluna 5 até a 9 tem-se as estatísticas tiradas dos resultados de cada alocação (total de iterações, máximo, mínimo, média e desvio padrão - DP). As 10 colunas seguintes mostram os valores em iterações para cada uma das aplicações(A1 - A10). Nas duas colunas finais mostra-se o número de aplicações que foram e que não foram alocadas.

Alocação de Recursos (resultados em iterações)
(10 Tentativas de alocação por aplicação)

				Estatísticas (Iterações)					Iterações para alocar											
Caso	Início alocação	Orç/Tarefa	No Tar/App	Total	Max	Min	Média	DP	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Aloc	Não
1	Simultânea	Alto	Muito Alto	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
2			Alto	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
3			Médio	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
4			Baixo	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
5		Meio	Muito Alto	51	8	0	5.10	1.7	3	3	4	4	5	5	7	6	8	6	10	0
6			Alto	46	8	0	4.60	1.8	3	2	3	5	5	4	4	5	7	8	10	0
7			Médio	37	8	0	3.70	2.3	1	1	3	3	3	3	3	7	8	5	10	0
8			Baixo	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
9		Baixo	Muito Alto	56	10	4	5.60	1	8	NA	10	10	8	NA	NA	10	NA	10	6	4
10			Alto	55	10	4	5.50	1.6	8	NA	NA	10	10	NA	10	10	9	NA	6	4
11			Médio	71	10	2	7.10	1.6	6	10	10	NA	10	7	9	9	NA	10	8	2
12			Baixo	64	9	0	6.40	1.8	8	6	7	8	9	5	4	8	NA	9	10	0
13	Seqüencial	Alto	Muito Alto	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
14			Alto	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
15			Médio	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
16			Baixo	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
17		Meio	Muito Alto	36	10	1	4.00	4.1	NA	10	1	10	1	8	1	1	1	3	9	1
18			Alto	32	10	1	3.56	3.9	NA	10	1	1	10	1	1	5	2	1	9	1
19			Médio	24	9	1	2.67	2.8	1	1	NA	9	1	1	1	4	5	1	9	1
20			Baixo	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
21		Baixo	Muito Alto	12	10	1	4.00	5.2	NA	NA	NA	NA	NA	NA	1	NA	10	1	3	7
22			Alto	12	10	1	4.00	5.2	NA	NA	NA	NA	NA	NA	1	NA	10	1	3	7
23			Médio	11	9	1	3.67	4.6	NA	NA	NA	NA	NA	NA	1	1	NA	9	3	7
24			Baixo	4	1	1	1.00	0	NA	NA	NA	NA	NA	NA	1	1	1	1	4	6
25	Aleatório	Alto	Muito Alto	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
26			Alto	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
27			Médio	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
28			Baixo	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
29		Meio	Muito Alto	51	8	0	5.10	1.9	2	2	5	5	6	5	5	6	7	8	10	0
30			Alto	46	8	0	4.60	1.8	2	3	3	5	6	4	4	8	5	6	10	0
31			Médio	37	5	0	3.70	1.6	1	1	4	3	3	5	5	5	5	5	10	0
32			Baixo	10	1	0	1.00	0	1	1	1	1	1	1	1	1	1	1	10	0
33		Baixo	Muito Alto	33	10	4	8.25	1.7	6	9	8	10	NA	NA	NA	NA	NA	NA	4	6
34			Alto	66	10	3	9.43	1.1	7	NA	10	10	NA	9	10	10	NA	10	7	3
35			Médio	81	10	1	9.00	1.3	6	8	NA	10	10	9	9	10	10	9	9	1
36			Baixo	76	8	0	7.60	0.5	8	8	8	8	7	7	8	7	7	8	10	0

Figura 5.4: Resultados da simulação do RAAP, quando o número de tentativas de alocação por aplicação é 10 (NA: não alocada).

5.3.1 Número máximo de tentativas de alocação

O número máximo de tentativas de alocação para as aplicações (T) é um parâmetro importante no resultado final (ver duas últimas colunas das tabelas). Observa-se nas tabelas que quando o número é alto (100, Fig. 5.6) é garantida a alocação de todas as aplicações, mesmo estas tendo orçamentos pequenos; enquanto para valores pequenos (10, Fig. 5.4) muitas aplicações ficam sem ser alocadas. Já para um valor médio (50, Fig. 5.5), observa-se que existem aplicações que ficam sem ser alocadas, mas isto só acontece quando o início da alocação é sequencial e o orçamento das tarefas é pequeno. O motivo é que a alocação das aplicações é feita uma depois da outra, o que não dá a possibilidade que outras aplicações dêem um maior dinamismo ao mercado (só uma aplicação por vez). Para o caso das aplicações com orçamento médio e alto, em quase todos os casos (exceto para 50 tentativas,

Alocação de Recursos (resultados em iterações)
(50 Tentativas de alocação por aplicação)

				Estatísticas (iterações)					Iterações para alocar											
Caso	Início alocação	Orç/Tarefa	No Tar/App	Total	Max	Min	Média	DP	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Aloca	Não A
1	Simultânea	Alto	Muito Alto	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
2			Alto	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
3			Médio	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
4			Baixo	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
5		Meio	Muito Alto	51	8	2	5.10	1.79	4	4	4	2	6	6	4	7	6	8	10	0
6			Alto	46	10	1	4.60	2.41	5	4	1	5	2	4	10	4	5	6	10	0
7			Médio	37	7	1	3.70	2.00	1	1	4	4	2	5	3	4	6	7	10	0
8			Baixo	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
9		Baixo	Muito Alto	112	17	5	11.20	3.05	9	13	10	5	12	12	17	12	11	11	10	0
10			Alto	110	15	9	11.00	2.58	6	10	9	13	12	9	12	11	15	13	10	0
11			Médio	100	15	5	10.00	2.58	9	7	8	9	11	7	10	13	11	15	10	0
12			Baixo	76	11	1	7.60	2.01	6	6	5	7	7	7	11	10	10	7	10	0
13	Seqüencial	Alto	Muito Alto	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
14			Alto	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
15			Médio	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
16			Baixo	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
17		Meio	Muito Alto	51	22	1	5.10	6.92	22	1	11	1	8	1	1	4	1	1	10	0
18			Alto	45	22	1	4.50	6.87	22	1	1	10	1	1	6	1	1	1	10	0
19			Médio	36	20	1	3.60	6.06	1	1	20	1	1	1	2	7	1	1	10	0
20			Baixo	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
21		Baixo	Muito Alto	60	21	1	6.67	7.63	NA	16	21	1	1	10	1	8	1	1	9	1
22			Alto	58	21	1	6.44	7.70	NA	16	21	1	11	1	1	1	1	5	9	1
23			Médio	43	20	1	4.78	7.56	NA	16	1	20	1	1	1	1	1	1	9	1
24			Baixo	24	16	1	2.67	5.00	NA	16	1	1	1	1	1	1	1	1	9	1
25	Aleatório	Alto	Muito Alto	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
26			Alto	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
27			Médio	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
28			Baixo	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
29		Meio	Muito Alto	51	7	3	5.10	1.60	2	3	4	7	6	5	6	6	6	6	10	0
30			Alto	46	7	3	4.60	1.58	2	4	4	4	3	7	5	5	5	7	10	0
31			Médio	37	8	1	3.70	2.36	1	1	2	3	3	3	4	7	5	8	10	0
32			Baixo	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
33		Baixo	Muito Alto	112	16	8	11.20	2.94	7	8	11	10	8	14	13	12	13	16	10	0
34			Alto	110	14	7	11.00	2.21	8	7	11	13	10	11	12	14	11	13	10	0
35			Médio	100	16	5	10.00	2.67	10	6	12	9	9	9	11	10	8	16	10	0
36			Baixo	76	9	1	7.60	0.97	8	6	9	9	8	7	7	8	7	7	10	0

Figura 5.5: Resultados da simulação do RAAP, quando o número de tentativas de alocação por aplicação é 50 (NA: não alocada).

início sequencial e orçamento baixo) é garantida a alocação, devido à dinâmica oferecida pela iteração das aplicações na procura de recursos.

Nos resultados mostrados, observa-se na tabela da Fig. 5.6 que para as condições nas quais o mercado foi simulado, o número de tentativas de alocação para o qual o mercado garante a alocação de todas as aplicações é 67, ou seja, se o número de tentativas for fixado para este valor e as outras condições se mantiverem, a alocação será feita, desde que os computadores não fiquem sobrecarregados.

O número máximo de tentativas de alocação por aplicação (T) é função do número de tentativas sem sucesso para mudar os preços num computador (parâmetro V_P e V_A explicados em 4.4.2). Assim, quando V_P e V_A são altos, a variação do mercado ocorre mais

Alocação de Recursos (resultados em iterações)
(100 Tentativas de alocação por aplicação)

				Estatísticas (iterações)					Iteraçoes para alocar											
Caso	Início alocação	Oríç/Tarefa	No Tar/App	Total	Max	Min	Média	DP	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Aloc	Não
1	Simultânea	Alto	Muito Alto	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
2			Alto	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
3			Médio	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
4			Baixo	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
5		Meio	Muito Alto	51	8	0	5.10	1.79	4	4	4	2	6	6	4	7	6	8	10	0
6			Alto	46	10	0	4.60	2.41	5	4	1	5	2	4	10	4	5	6	10	0
7			Médio	37	7	0	3.70	2.00	1	1	4	4	2	5	3	4	6	7	10	0
8			Baixo	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
9		Baixo	Muito Alto	112	17	0	11.20	3.05	9	13	10	5	12	12	17	12	11	11	10	0
10			Alto	110	15	0	11.00	2.58	6	10	9	13	12	9	12	11	15	13	10	0
11			Médio	100	15	0	10.00	2.58	9	7	8	9	11	7	10	13	11	15	10	0
12			Baixo	76	11	0	7.60	2.01	6	6	5	7	7	7	11	10	10	7	10	0
13	Seqüencial	Alto	Muito Alto	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
14			Alto	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
15			Médio	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
16			Baixo	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
17		Meio	Muito Alto	51	22	0	5.10	6.92	22	1	11	1	8	1	1	4	1	1	10	0
18			Alto	45	22	0	4.50	6.87	22	1	1	10	1	1	6	1	1	1	10	0
19			Médio	36	20	0	3.60	6.06	1	1	20	1	1	1	2	7	1	1	10	0
20			Baixo	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
21		Baixo	Muito Alto	112	67	0	12.44	20.67	67	21	1	1	10	1	8	1	1	1	10	0
22			Alto	110	67	0	12.22	20.74	67	21	1	11	1	1	1	1	5	1	10	0
23			Médio	100	67	0	11.11	20.90	67	1	20	1	1	1	1	1	1	6	10	0
24			Baixo	76	67	0	8.44	20.67	67	1	1	1	1	1	1	1	1	1	10	0
25	Aleatório	Alto	Muito Alto	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
26			Alto	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
27			Médio	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
28			Baixo	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
29		Meio	Muito Alto	51	7	0	5.10	1.60	2	3	4	7	6	5	6	6	6	6	10	0
30			Alto	46	7	0	4.60	1.58	2	4	4	4	3	7	5	5	5	7	10	0
31			Médio	37	8	0	3.70	2.36	1	1	2	3	3	3	4	7	5	8	10	0
32			Baixo	10	1	0	1.00	0.00	1	1	1	1	1	1	1	1	1	1	10	0
33		Baixo	Muito Alto	112	16	0	11.20	2.94	7	8	11	10	8	14	13	12	13	16	10	0
34			Alto	110	14	0	11.00	2.21	8	7	11	13	10	11	12	14	11	13	10	0
35			Médio	100	16	0	10.00	2.67	10	6	12	9	9	9	11	10	8	16	10	0
36			Baixo	76	9	0	7.60	0.97	8	6	9	9	8	7	7	8	7	7	10	0

Figura 5.6: Resultados da simulação do RAAP, quando o número de tentativas de alocação por aplicação é 100 (NA: não alocada).

lentamente e como consequência disto as aplicações pobres precisam de um maior número de iterações para serem alocadas. Valores de T acima de 50 garantem um maior sucesso na alocação de aplicações em mercados com características similares ao testado.

Para a análise nas seções seguintes, só serão usados os resultados quando o número de tentativas é 50 e 100, que são os mais representativos, pois permitem que quase todas as alocações sejam finalizadas.

Uma desvantagem de se usar um valor muito alto para o número de tentativas (T) é que pode haver uma redução significativa nos preços dos computadores, sem a garantia de que todas as aplicações serão alocadas. Sendo este o caso, as primeiras alocações não teriam sucesso, e as próximas provavelmente aconteceriam mais facilmente por que os preços

estariam muito baixos, sendo prejudicados os computadores por estarem trabalhando num mercado desvalorizado.

5.3.2 Início da Alocação

O inicio da alocação diz respeito à maneira como as aplicações serão iniciadas no MPVC. O caso mais comum em sistemas paralelos é o início aleatório, já que as aplicações estarão sendo submetidas em intervalos não regulares.

Como observado nos resultados (Fig. 5.4, Fig. 5.5 e Fig. 5.6), os números de iterações totais necessárias para alocar todas as aplicações com início aleatório e simultâneo são iguais, apesar de haver diferenças no número de iterações por aplicações individuais. O fato dos valores totais serem iguais sugere que, se o objetivo é apenas alocar todas as tarefas sem importar o número de iterações individuais, pode ser usado qualquer um dos dois inícios, mas se o objetivo é que as aplicações sejam alocadas em intervalos mais regulares (as aplicações são alocadas aproximadamente na mesma quantidade de iterações), então a escolha deve ser o início aleatório pelo mostrado no desvio padrão do número de iterações para a alocação (ver última coluna de estatísticas das tabelas para os inícios simultâneo e aleatório).

Para o início sequencial de aplicações com orçamento médio e baixo e um número muito alto de tarefas observa-se que o número de iterações para a alocação das primeiras aplicações (A1, A2, A3) é sempre alto. Isto ocorre porque a depreciação de preços ocorre somente pela tentativa de alocação de uma única aplicação, ao contrário dos outros tipos de início. A partir das alocações seguintes o número de iterações diminui já que os preços encontram-se em um nível baixo. Se o MPVC adotar o início sequencial, o mais recomendável será fazer a alocação com um orçamento médio ou alto para garantir maior possibilidade de sucesso de alocação das aplicações; mas se o orçamento for baixo, então o recomendável será incrementar o número máximo de tentativas de alocação para ter maior probabilidade de sucesso.

Como mostrado nas figuras 5.7, 5.9 e 5.8, o início simultâneo oferece melhores resultados na alocação para qualquer número máximo de tentativas de alocação. Isto é porque para este início todas as aplicações estão já disponíveis quando o processo inicia a alocação, o que faz que e o mercado tenha um comportamento mais dinâmico, oferecendo melhores

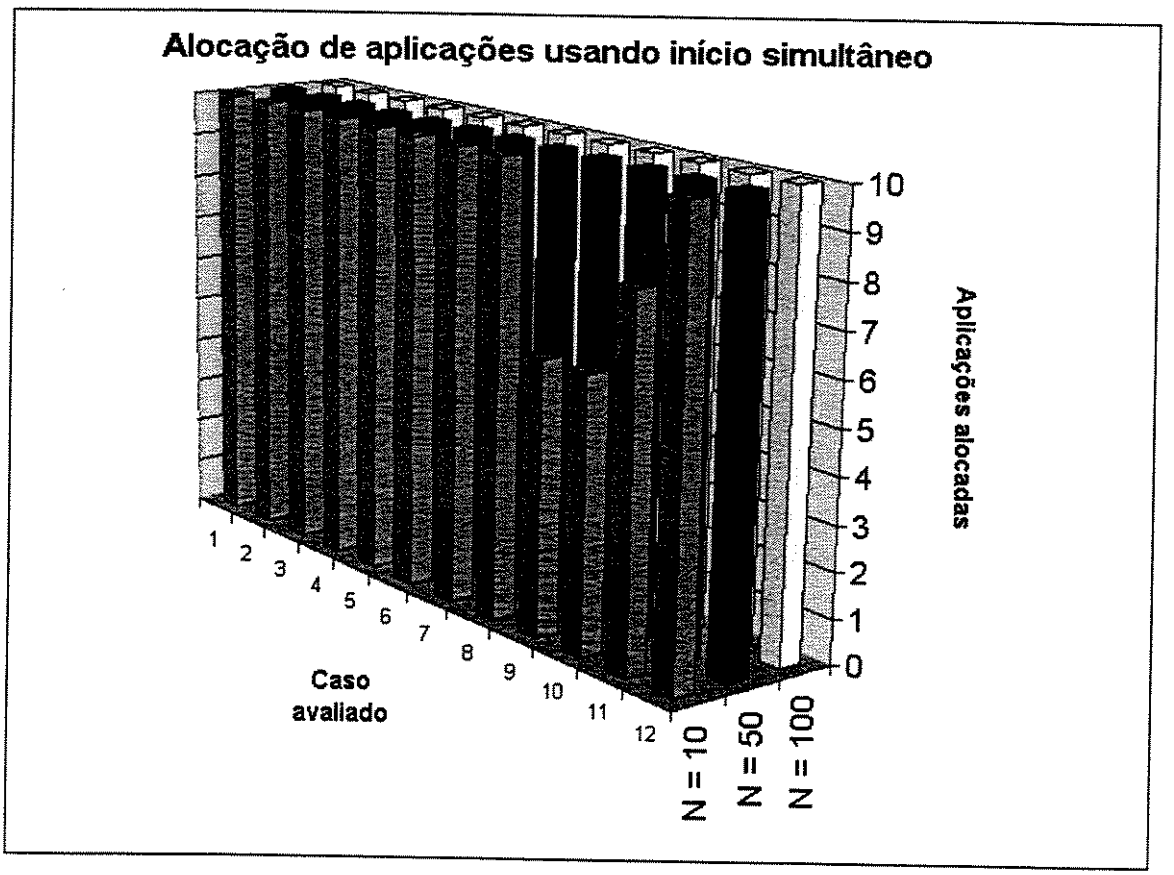

Figura 5.7: Alocação de aplicações usando início simultâneo.

oportunidades para a alocação. No caso do início sequencial, os resultados de alocação são os menos satisfatórios dos três casos analisados. O motivo para isto é que o mercado depende de uma só aplicação de cada vez. O início aleatório oferece resultados intermediários entre o início simultâneo e sequencial. Neste tipo de início pode acontecer que se tenha algumas aplicações sendo alocadas por vez. Também é possível que em um dado momento só exista uma aplicação sendo alocada e em outros momentos podem existir muitas aplicações tentando ser alocadas. Este início é o mais provável de acontecer em MPVCs na prática.

5.3.3 Orçamento por Aplicação

Nas simulações observa-se que para aplicações com orçamento alto a alocação acontece nas melhores condições, ou seja, estas aplicações conseguem os melhores e mais rápidos computadores e são alocadas em apenas uma iteração.

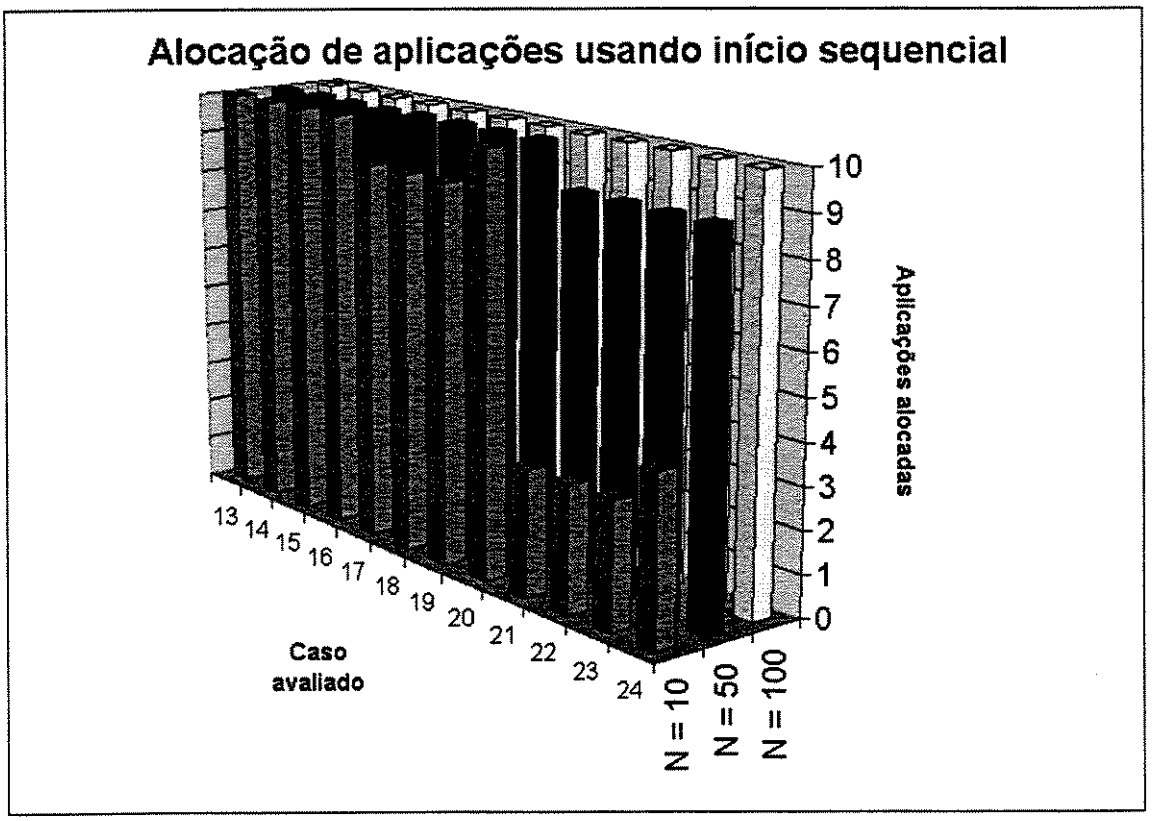

Figura 5.8: Alocação de aplicações usando início sequencial.

Para aplicações com orçamento médio e baixo as condições são diferentes. As tarefas das aplicações concorrem entre si pelos computadores e a alocação acontece em um tempo maior, porque a desvalorização dos preços até chegar aos níveis do orçamento das tarefas pode ser demorada. Para os casos testados, vê-se que um incremento no orçamento de 900% (de 1 cred./tarefa. para 10 cred./tarefa) diminui o número total de iterações em aproximadamente 50 %. Isto nos leva a pensar que a variação do orçamento não é muito influente na alocação, mas pela definição econômica do algoritmo RAAP o orçamento é sempre o que determina a prioridade da alocação, com as aplicações ricas sendo alocadas e executando antes que as aplicações pobres.

A respeito do orçamento, poderia se pensar em ter um esquema de orçamento dinâmico, ou seja a mesma aplicação poderia ir aumentando a quantidade oferecida pela execução das tarefas até conseguir alocar todas. Isto possibilitaria que as tarefas fossem alocadas em menor tempo. Como o pedido é feito uma única vez, quando a tarefa mãe precisa de novas tarefas, então um esquema deste tipo só seria possível de se implementar com a criação de

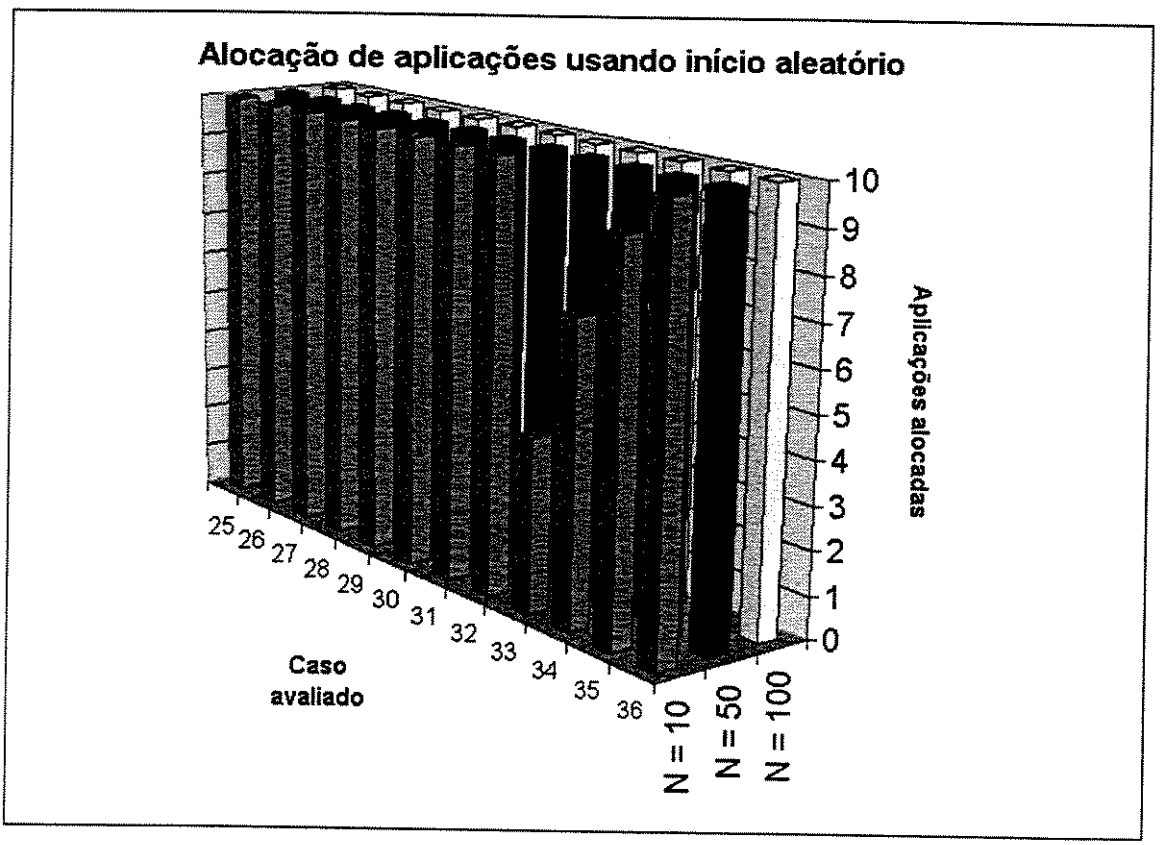

Figura 5.9: Alocação de aplicações usando início aleatório.

novos pedidos que substituiriam os anteriores da mesma tarefa solicitante. Fazer isto tem o custo adicional de maior tráfego na rede, além do esforço maior que o processo alocador precisaria fazer para controlar os pedidos que devem ser substituídos. Além disso, um aumento do orçamento não traz diminuições significativas no tempo de alocação. Por estes motivos, na prática o orçamento é fixo e só pode mudar quando a aplicação é submetida novamente ao sistema.

5.3.4 Número de tarefas por Aplicação

Observa-se que para aplicações com alto número de tarefas e orçamento médio ou baixo a alocação precisa de várias iterações. Já para número alto ou muito alto de tarefas por aplicação, a alocação pode não ter sucesso, o que significa que este parâmetro pode influir de maneira significativa sobre o processo de alocação.

Finalmente pode ser concluído que o número de tarefas por aplicação é importante para a alocação e pode ser determinante no processo, já que o algoritmo procura por computadores até obter uma quantidade mínima deles onde a aplicação possa executar. Assim, se em uma iteração não se consegue, tenta-se outra vez até conseguir, ou até que finalmente chega-se ao número máximo de tentativas de alocação, o que será finalmente o determinante do sucesso da alocação, como foi explicado em 5.3.1.

5.3.5 Simulação e resultados de situações reais

Sabendo que as situações anteriormente descritas são ideais e podem não ser comuns na prática (principalmente no que diz respeito à suposição de que todas as aplicações têm o mesmo orçamento), fizemos uma simulação variando o orçamento das aplicações. Como o caso mais comum na prática é que as aplicações tenham inicio aleatório, escolhemos esta maneira de início para simular a alocação de 10 aplicações com orçamentos diferentes. O parâmetro "Número de tarefas por aplicação" também foi variado. O motivo para esta simulação foi para analisar como se comporta o mercado quando existem aplicações com orçamento variado competindo pelos recursos.

As aplicações foram dotadas de orçamento variado e as situações simuladas foram as seguintes.

- Alocação de 10 aplicações com orçamento alto (Grupo 10x0).
- Alocação de 9 aplicações com orçamento alto e 1 aplicação com orçamento baixo (Grupo (9x1).
- Alocação de 5 aplicações com orçamento alto e 5 com orçamento baixo (Grupo 5x5).
- Alocação de uma aplicação com orçamento alto e 9 aplicações com orçamento baixo (Grupo 1x9).
- Alocação de 10 aplicações com orçamento baixo (Grupo 0x10).

Para cada uma destas situações foi variado o número de tarefas por aplicação, ou seja, usaram-se os mesmos 4 níveis da simulação anterior.

			Estatísticas (iterações)					Iteraçoes para alocar									
Inicio da aloc	Orç/Tarefa	No Tar/App	Total	Max	Min	Média	DP	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
Aleatório	Grupo 10x0	Muito Alto	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1
		Alto	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1
		Médio	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1
		Baixo	10	1	1	1.00	0.00	1	1	1	1	1	1	1	1	1	1
	Grupo 9x1	Muito Alto	92	83	1	9.20	25.93	1	1	1	1	1	1	1	1	1	83
		Alto	81	72	1	8.10	22.45	1	1	1	1	1	1	1	1	1	72
		Médio	82	73	1	8.20	22.77	1	1	1	1	1	1	1	1	1	73
		Baixo	76	67	1	7.60	20.87	1	1	1	1	1	1	1	1	1	67
	Grupo 5x5	Muito Alto	108	27	1	10.80	11.22	1	1	1	1	1	10	19	23	24	27
		Alto	88	26	1	8.80	9.14	1	1	1	1	1	10	18	14	15	26
		Médio	96	27	1	9.60	10.17	1	1	1	1	1	15	14	11	24	27
		Baixo	76	20	1	7.60	7.32	1	1	1	1	1	14	20	13	11	13
	Grupo 1x9	Muito Alto	112	17	1	11.20	5.07	1	9	10	14	7	15	8	17	16	15
		Alto	111	18	1	11.10	5.43	1	8	10	12	9	17	6	13	18	17
		Médio	95	15	1	9.50	4.40	1	7	6	8	14	12	15	13	12	7
		Baixo	72	13	1	7.20	3.94	1	9	9	13	10	6	1	10	5	8
	Grupo 0x10	Muito Alto	112	15	6	11.20	2.82	6	11	14	10	9	9	13	15	14	11
		Alto	110	16	6	11.00	3.16	9	8	10	14	13	14	9	6	11	16
		Médio	100	18	4	10.00	4.47	5	5	4	18	10	14	9	10	13	12
		Baixo	76	11	5	7.60	1.78	9	11	8	9	6	7	5	7	6	8

Figura 5.10: Simulação de aplicações com orçamentos diferentes.

A tabela de resultados desta simulação é mostrada na Fig. 5.10. Nesta tabela observa-se que quando as aplicações têm orçamento alto acontece o mesmo que na situação mostrada na simulação anterior (casos ideais), ou seja, a alocação acontece nas melhores condições (Grupo 10x0 da Fig. 5.10). Já quando existe uma aplicação com orçamento baixo e muitas com orçamento alto, observa-se que a alocação para a aplicação com orçamento baixo é mais difícil que quando todas as aplicações têm baixo orçamento (comparar Grupo 9x1 com Grupo 0x10). Isto ocorre porque as aplicações com alto orçamento geram uma demanda e os preços não são desvalorizados. Pelo contrário, são mais valorizados, fazendo que a aplicação com baixo orçamento só possa ser alocada depois que todas as aplicações com alto orçamento já foram alocadas. Mas, antes dela ser alocada, o mercado precisa ser desvalorizado (o que gera a demora).

Quando existe a mesma quantidade de aplicações com alto e baixo orçamento, observa-se que as aplicações com alto orçamento são alocadas mais rapidamente, sendo que as de baixo orçamento esperam pela desvalorização dos preços para poderem ser alocadas. Neste caso se percebe que as aplicações com baixo orçamento precisam de uma alta quantidade de iterações para serem alocadas, o que é um indicativo de que as aplicações de baixo orçamento são muito influentes no número de iterações necessárias para a alocação total

(Grupo 5x5).

Quando existem muitas aplicações com baixo orçamento e uma com alto orçamento, observa-se que a aplicação com alto orçamento é alocada facilmente, mas esta demanda atendida com sucesso não influi de maneira significativa na alocação das outras aplicações com baixo orçamento. Estas, por sua vez, geram uma maior demanda não atendida que faz os preços se desvalorizarem (Grupo 1x9).

Para a situação na qual todas as tarefas têm baixo orçamento, os preços são desvalorizados por influência das aplicações pobres que estão tentando ser alocadas (Grupo 0x10). O mercado se adequa às condições destas aplicações, sendo a alocação feita de maneira mais regular e fácil (não existem aplicações com alto orçamento para gerar oferta com preços altos, mas sim aplicações com baixo orçamento e que fazem pressão para que os preços sejam desvalorizados).

Portanto, a existência de muitas aplicações com orçamento alto e poucas com baixo orçamento influem de maneira mais significativa na alocação que quando há muitas aplicações com baixo orçamento e poucas com alto orçamento. Em geral as aplicações com baixo orçamento precisam de maior esforço para a alocação, o que se vê demonstrado no número de iterações necessárias para que elas sejam alocadas em todas as situações testadas.

5.3.6 Variação dos preços do computador

Uma preocupação no mercado de preços de recursos, assim como nos mercados reais, é a estabilidade dos preços, ou seja, como evitar o aumento ou diminuição desproporcionado de preços. Para seu bom funcionamento o mercado deve oferecer uma dinâmica que possa ser acompanhada pelos agentes que nele atuam.

No mercado de preços implementado a variação dos preços é influenciada por dois fatores: o número de tarefas em execução e as forças do mercado. O número de tarefas impõe uma relação direta entre a carga atual do computador e o seu preço. A força do mercado é resultado da dinâmica de oferta e demanda existente.

Para a variação dos preços dos computadores pelo número de tarefas (*LoadPrice* calculado com a Eq. 4.1) é esperado que computadores que têm uma grande quantidade de tarefas diminuam seu preço em proporção direta a sua carga, o que é comprovado na

prática, e que o aumentem quando o número de tarefas diminui, chegando ao mesmo nível do preço inicial quando a carga é zero.

Os preços controlados pelas forças de mercado (*MarketPrice* calculado com a Eq. 4.2) saem de um determinado patamar e variam de forma imprevisível, regidos pelas leis da oferta e de procura. Portanto, ao contrário dos preços regidos pelo número de tarefas, não existe uma garantia de que eles retornarão ao ponto de partida. É possível que eles diminuam em um primeiro instante, mas depois voltem a subir até ultrapassarem os seus valores iniciais (mercado inflacionário).

Entretanto, há mecanismos para que os preços voltem para o nível no qual o mercado tem estabilidade. Estes mecanismos estão presentes na maneira como os preços são atualizados. Assim, quando o preço de um computador aumenta muito, ele vai ter pouca demanda porque seu preço é alto. Portanto, a política de atualização de preços pelas forças de mercado fará que o preço diminua até alcançar novamente um nível que as aplicações possam pagar. Quando o computador ganha muitas tarefas para executar, seu preço diminui devido às tarefas, mas pode aumentar pelas forças do mercado.

Numa situação ideal, na qual todas as aplicações têm o mesmo orçamento (Fig. 5.11) observa-se que o preço *LoadPrice* permanece constante no nível inicial, enquanto o preço *MarketPrice* vai sendo desvalorizado até alcançar um nível que as aplicações podem pagar. Neste mesmo gráfico é mostrado que este último vai aumentando quando as tarefas começam a terminar, mas não chega ao valor inicial, diferentemente do que acontece com o preço *LoadPrice*.

Observa-se também que a desvalorização do *MarketPrice* acontece depois de permanecer por 3 iterações com o preço constante. Isto pode ser configurado pelo parâmetro de número de tentativas sem sucesso para mudar os preços num computador (parâmetro V explicado em 4.4.2), o qual determinará o dinamismo do mercado.

Na situação da Fig. 5.12 em que há uma alta demanda por aplicações com alto orçamento, os preços diminuem devido à carga, mas quando esta volta aos níveis iniciais, o preço *MarketPrice* torna-se maior que o inicial por causa da alta demanda que existiu, fazendo o mercado ficar mais valorizado.

Em situação na qual existe muita procura, com a maioria das aplicações com orçamento alto e uma com orçamento baixo, observa-se que os preços diminuem por efeito das tarefas

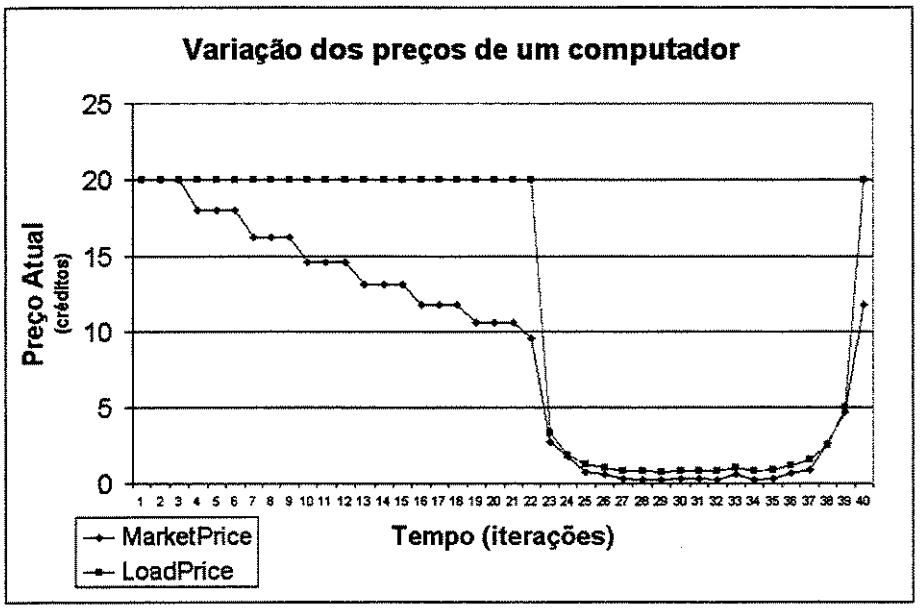

Figura 5.11: Variação do preço de um computador (baixa demanda).

e da demanda das aplicações com alto orçamento (Fig. 5.13). Neste caso a demanda das aplicações com orçamento alto é atenuada pelas tentativas sem sucesso de alocação da aplicação com baixo orçamento. Os preços diminuem pela execução das tarefas e quando os computadores começam a ter a carga diminuída, seus preços começam a aumentar como no caso anterior até que o preço *MarketPrice* seja maior que o preço inicial. Mas observa-se que conforme novas tentativas de alocação sem sucesso acontecem (pela aplicação com baixo orçamento) os preços vão diminuindo novamente.

Para uma situação na qual existe o mesmo de número de aplicações com alto e baixo orçamento (Fig. 5.14), observa-se que a alocação das aplicações com alto orçamento faz com que *LoadPrice* e *MarketPrice* caiam. Conforme as aplicações com alto orçamento que foram alocadas vão terminando, *LoadPrice* vai aumentando. Mas não acontece o mesmo com MarketPrice, já que existem aplicações com baixo orçamento que ainda estão tentando ser alocadas. Quando os preços pelas forças do mercado caem até níveis que as aplicações com baixo orçamento podem pagar, acontece a alocação destas aplicações, e quando as aplicações vão terminando a execução das tarefas, o *LoadPrice* aumenta mas *MarketPrice* permanece em um nível baixo pela alta demanda sem sucesso que se teve.

Já para uma situação na qual a maioria de aplicações têm orçamento baixo e só uma tem orçamento alto (5.15), observa-se que os preços vão sendo desvalorizados até que as

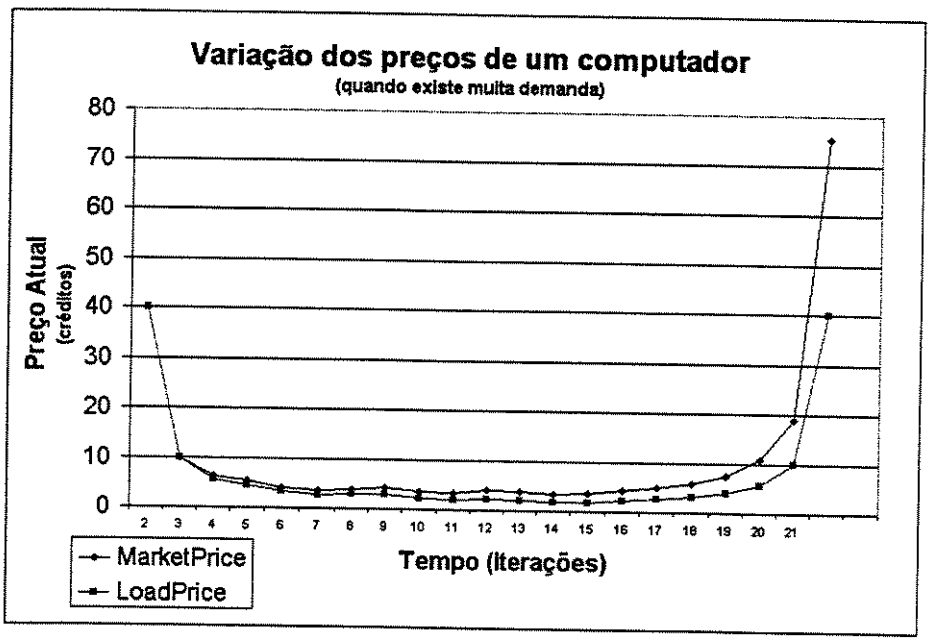

Figura 5.12: Variação do preço de um computador (aplicações com alto orçamento).

tarefas da aplicação com alto orçamento são alocadas e o preço cai pela carga que estas tarefas impõem ao computador. Como a maioria de aplicações tem baixo orçamento, os preços continuam sendo desvalorizados até chegar aos níveis que as aplicações podem pagar. Quando a carga começa a diminuir, $MarketPrice$ aumenta mas não chega aos níveis iniciais porque foi bem desvalorizado para se adequar aos orçamentos das aplicações.

Quando existe muita demanda mas todas as aplicações têm baixo orçamento, observa-se que os preços dos computadores são desvalorizados até alcançar os níveis que as aplicações com baixo orçamento podem pagar (Fig. 5.16). Neste caso, quando as tarefas vão acabando, $MarketPrice$ vai aumentando mas se mantém em um nível baixo, onde o mercado está estabilizado.

Com as situações descritas nesta seção foi mostrado como o mercado é dinâmico e como os preços dos computadores atualizado pelas forças do mercado podem variar de maneira a se adequar às condições do mercado, aumentando a possibilidade de que aplicações com orçamento baixo possam ser alocadas. Se a variação do preço for muito grande, os mecanismos de oferta e demanda atuam para que estes se estabilizem em níveis que as aplicações podem pagar.

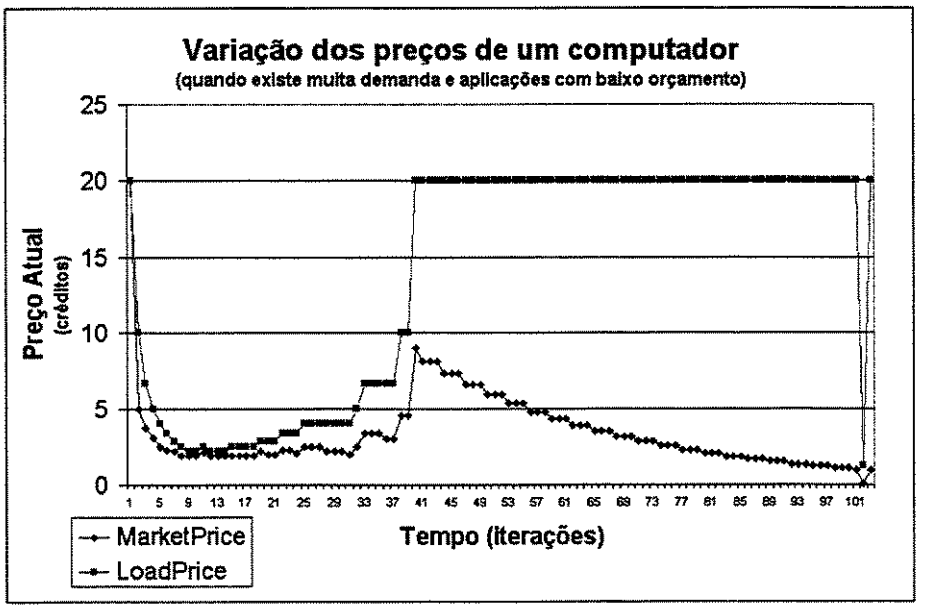

Figura 5.13: Variação do preço de um computador (9 aplicações com alto orçamento e uma aplicação com baixo orçamento).

5.4 Comparações com outros métodos de alocação.

Embora este trabalho apresente resultados qualitativos e quantitativos, os trabalhos afins descritos no Capítulo 2 não têm dados suficientes para viabilizar um comparação detalhada.

No caso de PINCO, a similaridade com este trabalho está somente na classificação que se faz dos computadores. PINCO usa uma unidade de medida chamada de ETP que representa a décima parte do poder de um computador Pentium 133 descarregado. Para a alocação é necessário que o programador especifique a quantidade de ETPs que a aplicação necessitará para a sua execução. Como RAAP não exige um conhecimento prévio do poder de computação necessário a uma aplicação, a comparação dos tipos de alocação se torna difícil.

Como o método de alocação de PopCorn se baseia em leilões, o programador precisa definir para cada *Computelet* (mesmo sendo da mesma aplicação paralela) um orçamento inicial, um valor de incremento para este orçamento e um limite superior para ofertas por um recurso. A existência destes parâmetros, além dos cálculos que eles requerem, torna mais complexa a programação e a busca dos valores adequados para os mesmos. Isto, associado às trocas de mensagens para viabilizar os leilões, faz com que o método de alocação de

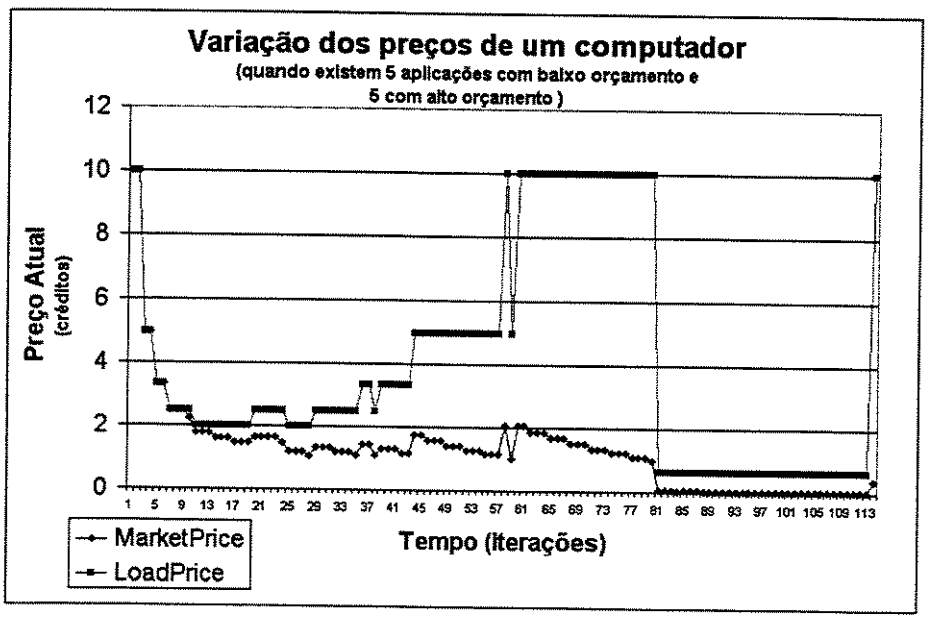

Figura 5.14: Variação do preço de um computador (5 aplicações com baixo orçamento e 5 aplicações com alto orçamento).

PopCorn não seja escalável, e o torna inadequado para sistemas com um número elevado de componentes. No RAAP, um pedido de criação de tarefas de uma aplicação paralela consiste em uma mensagem que contém especificado o orçamento e a quantidade de tarefas a criar. Isto é mais simples de manipular, além da vantagem da diminuição de tráfego na rede, já que leilões não são usados.

A alocação de recursos presente em JoiN atualmente é muito simples. Ela busca o balanceamento de carga entre os computadores, ou seja, as tarefas são alocadas uma por uma entre todos os computadores, tentando manter a mesma quantidade de tarefas executando em cada um. Este trabalho apresenta uma alternativa para a alocação de recursos e para o cálculo de recompensas para os donos dos computadores em futuras versões de JoiN.

Devido às características que distinguem o Algoritmo RAAP, a comparação com outros métodos existentes se torna difícil. Apesar dos métodos de alocação terem em geral quase os mesmos objetivos finais, a maneira de implementá-los faz com que sejam orientados especificamente para a necessidade do ambiente no qual trabalharão. Estas necessidades podem ser rapidez de execução, balanceamento de carga, dentre outros fatores. O RAAP é orientado à alocação efetiva (ou seja, atendimento completo do pedido de recursos),

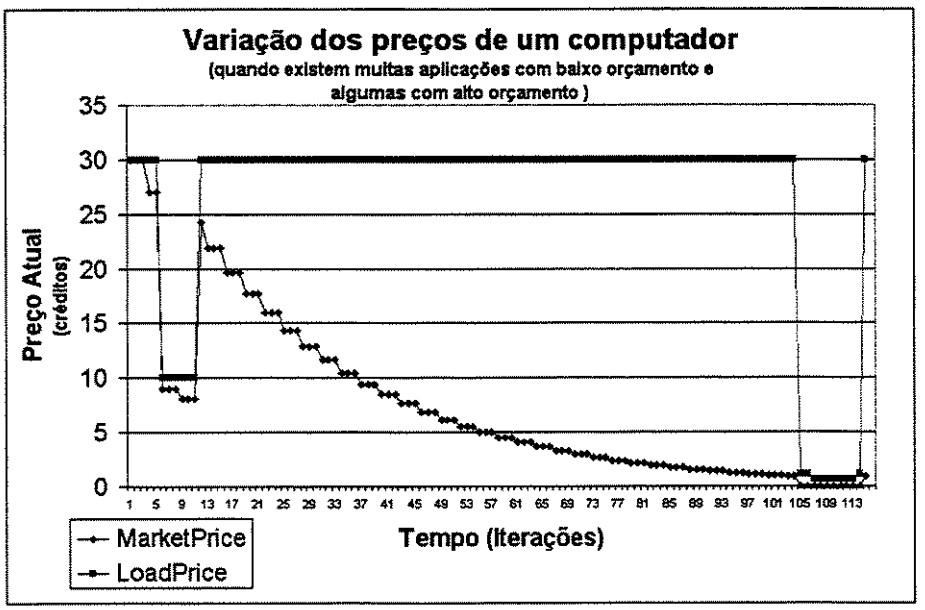

Figura 5.15: Variação do preço de um computador (9 aplicações com baixo orçamento e uma aplicação com alto orçamento).

mantendo um balanceamento de carga, e provendo, além disso, uma base para o cálculo de recompensa para os donos dos computadores participantes.

5.5 Conclusões

Neste capítulo foram mostrados e discutidos os resultados das simulações feitas com o Algoritmo de Alocação usando preços (RAAP). Conclui-se que o parâmetro mais importante é o número máximo de tentativas de alocação já que, dependendo deste, a aplicação será alocada ou não. Já o orçamento das aplicações serve como um meio de priorização, para determinar se a aplicação será servida mais rapidamente. O número de tarefas por aplicação pode ser determinante para a alocação ser feita com sucesso ou não, pois depende deste o número de tentativas de alocação. Se objetivo é apenas alocar as tarefas das aplicações, então o início das alocações pode ser aleatório ou simultâneo, já que em termos totais, o número de iterações é igual. Para início sequencial da alocação, o mercado tem menor dinamismo (as variações das condições do mercado só são influenciadas por uma aplicação por vez) e as aplicações precisam de orçamentos mais altos para serem alocados em melhores condições.

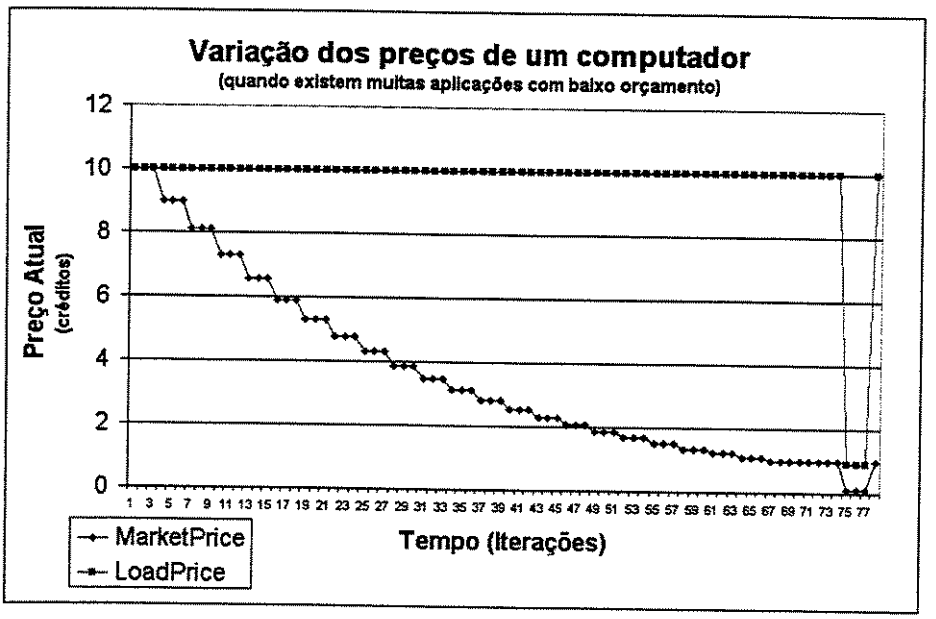

Figura 5.16: Variação do preço de um computador (aplicações com baixo orçamento).

Nas situações analisadas neste capítulo, também foi mostrada a variação dos preços pelas forças do mercado, e foi explicado como os mecanismos de mercado atuam para que os preços voltem para um nível que as aplicações podem pagar.

Capítulo 6

Conclusões finais e trabalhos futuros

6.1 Conclusões

Quando a complexidade de ambientes computacionais é aumentada, evidencia-se a utilidade de usar mecanismos de alocação descentralizados. Esta dissertação combina pesquisas em economia e ciências de computação para fornecer um mercado de recursos para alocação de processadores para tarefas em um ambiente paralelo e heterogêneo.

Os mecanismos mostrados são relativamente simples de implementar e podem ser programados sem maiores dificuldades. O programa desenvolvido para a simulação explorou uma gama ampla de opções e poderá servir como base para uma futura implementação prática em um computador massivamente paralelo como JoiN.

Com os resultados obtidos, mostra-se a viabilidade de se usar abordagens econômicas com sucesso na alocação de recursos computacionais, sendo os conceitos aplicados simples e fáceis de entender. Além disso a implementação e o uso destes conceitos não requer um grande esforço.

Nos mercados de recursos computacionais muitos fatores influenciam no sucesso dos algoritmos de alocação destes recursos. Um amplo estudo destes fatores baseado em simulações foi efetuado, permitindo uma avaliação do impacto de cada um deles no sucesso da alocação. Apesar de terem sido adotados nestas simulações valores arbitrários para os diversos parâmetros envolvidos, o elevado número de casos analisados nos permite concluir

que o comportamento de um sistema real será muito próximo do que foi apresentado.

Várias conclusões podem ser tiradas deste trabalho.

- Na pesquisa bibliográfica observou-se que existe uma tendência em se usar sistemas de processamento massivamente paralelos virtuais baseados na Internet.
- Na classificação dos computadores foi apresentado um método de classificação de computadores num MPVC baseado na Internet. Este método combina vantagens de outros métodos e torna factível a classificação de maneira rápida e exata.
- Foi feita uma primeira aproximação da definição do preço para os computadores,e mostrou-se que, para computadores com alto poder de processamento, deve ser paga uma maior quantidade de créditos que para computadores lentos que em conjunto fornecem o mesmo poder de processamento.
- O Algoritmo RAAP, que implementa duas maneiras de atualização de preços, oferece maiores possibilidades de alocação de recursos para qualquer aplicação, independentemente do orçamento da mesma.
- RAAP em condições normais precisa de poucas mensagens para fazer a alocação, assim como também de poucas mensagens para manter atualizada a informação em todo o sistema.
- RAAP oferece um mecanismo de balanceamento de carga baseado na carga atual do computador.
- RAAP se apresenta como uma alternativa viável para a alocação de recursos em sistemas de processamento massivamente paralelo baseados na Internet.
- Muitos fatores influenciam no sucesso dos algoritmos de alocação de recursos. Em RAAP o parâmetro mais importante é o número máximo de tentativas de alocação que determina o sucesso ou não da alocação. O parâmetro número de tarefas por aplicação pode ser determinante também.
- Os inícios simultâneo ou aleatório da alocação de aplicações podem ser usados indistintamente já que apresentam resultados globais similares. O início sequencial não é recomendável devido ao baixo dinamismo que ele gera no mercado. O início mais comum na prática é o aleatório.

- RAAP implementa mecanismos de mercado que fazem que os preços se estabilizem no nível do orçamento das aplicações. Quando os preços de mercado aumentam ou diminuem muito, os mecanismos de mercado atuam para corrigí-los.

Esta pesquisa representa um passo inicial no desenvolvimento de mercados práticos de alocação de recursos para sistemas computacionais, e espera-se que as idéias aqui expostas sirvam como base para futuras pesquisas. As vantagens dos mercados e a extensa pesquisa sobre mecanismos econômicos servem como base para sistemas de alocação baseadas em mercados em grande escala e espera-se que seu uso seja tão comum em outras áreas como começa a ser na área de ciências da computação.

6.2 Trabalhos futuros

O desenvolvimento de mercados computacionais está ainda na infância. Nesta seção se discutem algumas das questões que precisam de um estudo mais aprofundado, assim como possíveis dificuldades no uso de mercados computacionais em aplicações práticas no futuro.

As tarefas do programador de aplicações para sistemas que implementam mercados computacionais consistem não apenas em programar as rotinas que fazem o processo, mas também em determinar as funções de custo e os orçamentos das aplicações. Uma das questões abertas é sobre os critérios para a alocação de dinheiro (orçamento), já que todos os mercados são sensíveis aos níveis de dinheiro que a economia manipula. As aplicações com grande orçamento têm uma grande influência na determinação do preço de mercado de recursos, e podem aumentar os preços criando uma inflação. Se não houver mecanismos de retorno aos níveis iniciais, há poucas chances de que futuras aplicações com orçamento pequeno possam ser executadas no MPVC.

Em uma economia computacional, é concebível criar dinheiro e injetá-lo no mercado, mas o problema é determinar como esta ação afeta a economia como um todo. Obviamente isto tem conseqüências macroeconômicas tais como inflação e outras que serão melhor entendidas aplicando-se conhecimentos de macroeconomia.

Alguns dos assuntos que devem ser estudados com maior detalhe em trabalhos futuros são mostrados a seguir.

- Cálculo da carga individual das tarefas, ou seja, suas necessidades de processamento.
- Definição de orçamentos para as aplicações, ou seja, uma maneira de calcular a quantidade efetiva de créditos alocados às aplicações.
- Estudo mais detalhado dos problemas de manipulação de mercados, a fim de criar mecanismos para evitá-los ou corrigí-los.
- Consideração de mecanismo otimização na alocação, a ser usado em conjunto com o mercado de recursos implementado.
- Estudo mais aprofundado dos problemas de tolerância às falhas que podem afetar especificamente o algoritmo de alocação de recursos.

Esta pesquisa está baseada principalmente na alocação de processadores da Internet, mas isto não é motivo para limitar estes mecanismos apenas a estes recursos. Muitas das vantagens de se usar abordagens econômicas são desejáveis também para a alocação de outros tipos de recursos como impressora, enlace de comunicação, bancos de dados. Estes mesmos mecanismos podem ser usados também na alocação de recursos locais de computadores como espaço de disco, memória, dentre outros.

Finalmente, esta dissertação pretende contribuir para o entendimento e uso prático de conceitos de economia em sistemas computacionais e esperamos que esta contribuição seja usada para ampliar as pesquisas nesta área.

Bibliografia

[ACP95] T. Anderson, D. Culler, D. Patterson. *A case for Networks of Workstations: NOW*, IEEE Micro, Feb. 1995. `http://now.cs.berkeley.edu`.

[AG99] A. Clematis, G. Dodero, V. Gianuzzi, *A resource management tool for heterogeneous networks* , Proc. PDP'99, 7th Euromicro Workshop on Parallel and Distributed Processing, Madeira, Portugal, Feb. 3-5 1999.

[BMPD] System Optimization Information, *Benchmark Program Descriptions*, Earth Web Inc., 1997.
`http://www.sysopt.com/benchdesc.html`.

[CH92] D. Chaum, *Achieving Electronic Privacy*, Scientific American, pp. 96-101, Aug.1992.

[CO96] Epenema D., Livny M., Van Dantzig R., Evers X., Pruyne J., *A Worlwide Flock of Condors: Load sharing among Workstations Clusters*, Journal of Future Generation Computer Systems, (12): 53-65, 1996.

[CSS] Client/Server Solutions, *Software Benchmark Factory*, 1997
`http://www.csrad.com/`.

[DD] H.M.Deitel, P.J.Deitel, *JAVA, How to Program*, Prentice Hall, First Ed., 1997.

[DL] Dennis K, Liya H, *mpi++: A C++ Language Binding for MPI.*

[DV98] Aleksander Mandic, *Quem ganha dinheiro com a Internet no Brasil?*, Developers' Magazine, pp. 16-17, Set.1998.

[EH98] Eduardo Huerta. *Um Sistema para o Processamento Massivamente Paralelo na World Wide Web*, Dissertação de Mestrado, FEEC/UNICAMP, Aug. 1998.

[EM99] Eduardo Huerta, Marco Henriques. *A Method to Solve the Scalability Problem in Massively Parallel Processing on the Internet*, The 7th Euromicro Workshop on Parallel and Distributed Processing, Funchal, Portugal, pp. 256-262, Feb. 1999. `http://www.dca.fee.unicamp.br/projects/join/docs/ipps_99.pdf`.

[FE88] Ferguson D., Nikolaou C., Sairamesh J., Yemini Y., *Microeconomic Algorithms for Load Balancing in Distributed Computer Systems*, Proc. of International Conference on Distributed Systems (ICDCS' 98), 1998. `http://www.cs.columbia.edu/~jakka/dcs88.ps`.

[FE96] Ferguson D., Nikolaou C., Sairamesh J., Yemini Y., *Economic Models for Allocating Resources in Computer Systems - Market based Control of Distributed Systems*, Ed. Scoot Clearwater, World Scientific Press, 1996. `http://www.cs.columbia.edu/~jakka/bookchapter.ps`.

[FF97] Geoffrey Fox, Wojtek Furmanski. *Computing on the Web. New Approaches to Parallel Processing. Petaop and Exaop Performance in the Year 2007*, IEEE Internet Computing, 1(2), Mar. - Abr. 1997.

[FK98] Ian Foster, Carls Kesselman. *The Globus Project: A Status Report*, Proceedings of the IPPS/SPDP '98 Heterogeneus Computing Workshop, pages 4-18, 1998.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam. *PVM: Parallel Virtual Computer*, a Users Guide and Tutorial for Networked Parallel Computing. The MIT Press, 1994.

[GEOT] Geocities, *G*uia de geoguide sweepstakes, 1996. `http://www.geocities.com/members/tools/geoguide/sweepstakes.html`.

[GSW94] Alok G., Dale S., Andrew W., *Managing The Internet as an Economic System*, Working Paper, Department MSIS, University of Texas at Austin, Jul. 1994.

[HPB] Hewlett-Packard Company, *Netperf 2.1* . `http://www.netperf.org/netperf/NetperfPage.html`.

[IES68] I. Sutherland. *A futures Market in computer time*, Communications of the ACM, 11:449-451, 1968.

[JBA] Jeffrey M., Brian C., Andrew L., *Object Oriented MPI: A Class Library for the Message Passing Interface*, Tecnical Report, Laboratory for Scientific Computing, Department of Computer Science and Engeneering, University of Notre Dame. `http://www.cse.nd.edu/~lsc/research/ompi/`.

[JD94] Jon Doyle, *A reasoning Economy for Planning and Replanning*, Technical Paper, ARPA Plannning Initiative Workshop, Tucson, AZ, Feb. 1994.

[JSM] Sun Microsystems, *JavaTM Development Kit Documentation*, 1998 `http://java.sun.com/products/jdk/1.2/index.html`.

[JVM] Sun Microsystems. *The Java Language Specification*, 1996. `http://java.sun.com/docs/books/jls/html/index.html`.

[LE96] Grimshaw A., Wulf W., *Legion - A View From 50,000 Feet*, Proceeding of the Fifth IEEE International Symposium on High Performance Distributed Computing, Los Alamitos, California, IEEE Computer Society Press, Aug. 1996.

[LP+99] K. Li, Y. Pan, H. Shen, S. Q. Zheng. *A Study of Average-Case Speedup and Scalability of Parallel Computations on Static Networks*, Mathematical and Computer Modelling, 29(9):83-94, May. 1999.

[LS82] L. Lamport, R. Shostak, M. Pease. *The Byzantime Generals Problem*, ACM Transations on Programing Language and Systems, 4(3):382-401, 1982.

[MIDS99] The Matrix and Information Directory Service Web Site, 1999. `http://www.mids.org/growth/internet/`.

[N98A] Netscape Communications Corporation. *Java and Security, 1988*, Developer's edge archived Conference materials. `http://developer.netscape.com/misc/developer/conference/proceedings/j4/`.

[N98B] Netscape Communications Corporation. *Netscape Object Signing. Establishing Trust for Downloaded Software, 1998.* White Paper. Developer's Edge Online Documentation. `http://developer.netscape.com/docs/manuals/signedobj/trust/index.html`.

[NB94] Nataniel Bogan, *Economic Allocation of Computation Time with Computation Markets*, Thesis for degree of Master of Engeneering, Department of Electrical En-

geneering and Computer Science, Massachusetts Institute of Technology - MIT, May. 1994.

[NL] Natawaut Nupairoj, Lionel M. Ni, Performance Evaluation of Some MPI Implementations on Workstation Clusters, Departament of Computer Science, Michigan State University.

[OR98] Ori Regev. *Economic Oriented CPU Sharing System for the Internet*, Master of Science Thesis in Computer Science. Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Jul. 1998.

[PH96] John L. Hennessy, David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publisher, Second Edition, Cap. 7, 1996.

[PM99] Pedro W. Chávez, Marco A. Amaral Henriques. *Alocação de Recursos Computacionais Baseada em Modelo Econômico para Sistemas de Processamento Massivamente Paralelo Virtuais*, XXVI Seminário Integrado de Software e Hardware (SEMISH), XIX Congresso da SBC, Rio de Janeiro, Jul. 1999, pp. 211-200.
`http://www.dca.fee.unicamp.br/~pedroch/SEMISH/PedroSEMISH.SBC99.pdf`.

[PO97] Nisan N., London S., Regev O., Camiel N., *Globally Distributed Computation Over the Internet - The POPCORN Project*, Poster paper in WWW6 - Sixth International World Wide Web Conference, Santa-Clara, Apr. 1997.
`http://www.cs.huji.ac.il/~popcorn/documentation/popcorn-submit.doc`.

[PSL80] M. Pease, R. Shostak, L. Lamport. *Reaching Agreement in the Presence of Faults*, Journal of the ACM, 27(2):228-234, 1980.

[RF87] Daniel A. Reed, Richard M. Fujimoto, *Multicomputer Networks: Message Based Parallel Processing*, Scientific Computation Series. The MIT Press, 1987.

[SI] The SETI Institute, *SETI@home: Search for Extraterrestrial Intelligence.* `http://www.setihome.com`.

[SL] S. London, *The Popcorn Tutorial*, 1997.
`http://www.cs.huji.ac.il/~popcorn/developer/tutorial/index.html`.

[SOH+96] Marc Snir, Steve Otto, Stenven Huss-Lederman, David Walker, Jack Dongarra. *MPI: The Complete Reference*, The MIT Press, 1996.

[SUN97a] Sun Microsystems. *Secure Computing with Java: Now and the Future.* JavaOne Conference, 1997. White Paper.

[SUN97b] Sun Microsystems. *Remote Method Invocation Specification, 1997.* Disponível na documentação do JDK 1.1 e versões posteriores.

[WM93] Michael P. Wellman, *A market-oriented programming environment and its application to distributed multicommodity flow problems*, Journal of Artificial Intelligence Research, 1:1-23, 1993.

Apêndice A

Classes principais do simulador

As classes foram divididas em duas categorias: as que pertencem ao processo alocador e as que pertencem à abordagem econômica. Cada categoria foi encapsulada em pacotes diferentes.

O simulador foi implementado na maneira de uma aplicação paralela, composta por uma tarefa "mãe"que é encarregada de criar todos os componentes do processo alocador. Como a versão do browser usado (Netscape 4.5 para Unix) implementa uma Máquina Virtual Java que não permite escrever em disco desde um applet Java, a classe LogServer, que se encarrega de receber mensagens desde a aplicação e escrever no disco, foi integrada com o Servidor do JoiN. Os detalhes para a execução do programa simulador são detalhadas no Apêndice B.

As classes são descritas a seguir, assim como a declaração em linguagem de programação Java.

A.1 Package join.Allocator

Fazem parte deste pacote as classes que iniciam o processo de alocação, a que faz a alocação, a que se encarrega de guardar os arquivos de resultados da simulação [1], e as que calculam o resultado a partir dos testes (benchmark), classificando os computadores de acordo ao

[1] Esta é a maneira adotada para guardar os resultados para análise posterior.

preço obtido no teste.

Class JoinAllocator: classe inicial do processo que realiza a alocação. Esta classe se encarrega de iniciar os processos alocadores (Allocator) e de criar as aplicações.

```
public class JoinAllocator extends JoinTask{ // Declaração.
    int []TID_Allocators; // Identificadores das Allocator' criados.
    public void CreateAllocators(); // Método que cria os Allocators.
    public void run(); // Método principal de execução.
}
```

Class Allocator: classe principal do Algoritmo; uma instância desta classe é criada no Coordenador. Ela é encarregada de receber as mensagens desde as aplicações e criar objetos ProcessRequest para executá-los. Além disso inicia uma classe ServerBenchMark que é a encarregada de executar os testes com um computador que esteja entrando para formar parte do MPVC.

```
public class Allocator extends JoinTask{ // declaração.
    ListPriceHosts ListPH; // Lista de computadores.
    ListBuddySet ListBS; // Lista dos vizinhos.
    InetAddress AddressThisAllocator; // Endereço IP do Allocator.
    int IDGroup; // Identificador do grupo dentro do Hipercubo.
    Messenger log; // Objeto para log no server.
    // Retorna lista de computadores.
    public ListPriceHosts getListPriceHost();
    // Retorna lista de grupos vizinhos.
    public ListBuddySet getListBuddySet();
    public void run(); // Método principal do Allocator.
}
```

Class ProcessorRequest: classe que se encarrega de interpretar e executar as mensagens que chegam ao Allocator. Esta classe tem um método que se encarrega de fazer a alocação para cada pedido de criação de tarefas que chega.

```
public class ProcessRequest extends JoinTask{ // declaração.
    ListPriceHosts ListPH ;// referência à lista de computadores
```

```
    ListBuddySet ListBS; // referência à lista de vizinhos.
    public synchronized void CreateListOfPrice(ObjectMessage msg);
    // cria a lista de  computadores de acordo à mensagem msg chegada.
    public synchronized void UpdatePricesList(SortedList List);
    // Atualiza os preços dos computadores escolhidos para a alocação.
    public void run(); // método principal da classe.
}
```

Class LogServer: classe que recebe mensagens desde qualquer módulo e escreve no disco do servidor. Esta classe foi integrada no servidor do JoiN e cada vez que chega uma mensagem para ser escrita em disco, cria objetos SaveToLog para escrevê-los.

```
public class LogServer { // declaração
    //cria objetos SaveToLog para cada mensagem om chegada.
    public void save(ObjectMessage om);
}
```

Class SaveToLog: classe que recebe uma mensagem e grava o dado da mensagem no arquivo especificado na mesma mensagem.

```
public class SaveToLog extends Thread { //declaração
    boolean isWriting=false; // para sincronizar a escrita em arquivos iguais
    public SaveToLog(ObjectMessage om) //grava no disco a mensagem om.

}
```

Class Messenger: classe que envia as mensagens do simulador para o servidor.

```
public class Messenger{ // declaração.
    String filename; // arquivo onde gravará a mensagem.
    public void Save(Vector vector); // salvar o Vetor vector.
    public void Save(ListPrice list); // salvar uma lista de preços list.
    public void Save(String string); // salvar um string.
    public void Save(int []ints); // salvar um array de inteiros.
}
```

Class MessageAllocator: classe que contém todos os tag's das mensagens enviadas dos computadores trabalhadores até o Allocator e do Allocator aos computadores trabalhadores.

```
final public class MessageAllocator{  // declaração.
    // Tag de qualquer pedido.
    public static final int ALLOCATORMSG_ANY                = 9999;
    // Tag de pedido de criação de Tarefas
    public static final int ALLOCATORMSG_CREATETASKS        = 1;
    // Tag para registrar o preço de um computador.
    public static final int ALLOCATORMSG_REGISTERPRICE      = 2;
    // Tag para devolver os TID's das tarefas.
    public static final int ALLOCATORMSG_TID_TASKS          = 3;
    //mensagem enviado ao endereço do Allocator onde esta executando o Servei
    public static final int ALLOCATORMSG_IPCOORD            = 4;
    // mensagem enviada para indicar que uma tarefa acabou
    public static final int ALLOCATORMSG_TASKFINISHED       = 10;
    // Prioridade da aplicação por tempo, executar o mais rapido possível.
    public static final int APPPRIORITY_TIME                = 30;
    // prioridade da aplicação por dinheiro, gastar o menos possível.
    public static final int APPPRIORITY_MONEY               = 31;

    // Mensagens usadas para os testes de enlace de comunicacao
    // mensagem para iniciar o teste de enlaçe de comunicação.
    public static String MESSAGE_START_TEST_LINK            = "LINK";
    // mensagem para iniciar o teste foi aceita.
    public static String MESSAGE_START_TEST_LINK_ACK        = "GO";
    // mensagem usada para receber um String no teste de Link com o servidor.
    public static final int ALLOCATORMSG_TESTLINK           = 50;
}
```

Class RequestResource: classe que indica um requerimento de recursos, contendo a quantidade de tarefas a criar, o orçamento da aplicação, e a prioridade. Objetos desta classe são instanciados pelas tarefas "mães" que precisam criar tarefas, e são enviados para o Allocator que responde com uma lista de identificadores das tarefas criadas (TID's) ou uma resposta de fracasso da alocação.

```
public class RequestResource implements Serializable{ //declaração.
    String Task; // nome da tarefa a criar.
    int Request; // número de tarefas a criar.
    double ToSpend;  // orçamento da aplicação.
    int Priority; // prioridade da aplicação.
    public String getTask(); // retorna o número de tarefas.
}
```

Class ServerBenchmark: Classe que é iniciada no Allocator para receber conexões dos computadores para fazer o teste de enlace de comunicação.

```
public class ServerBenchmark extends Thread{ // declaração.
    // número máximo de conexões simultâneas.
    static public int MAX_CONNECTIONS = 100;
    // Socket servidor que recebe as  conexões.
    ServerSocket s=(ServerSocket)null;
    // método principal.
    public void run();
}
```

Class TestServerLink: classe que é iniciada por cada conexão recebida dos computadores clientes para fazer o teste de enlace de comunicação.

```
public class TestServerLink extends Thread{ // declaração.
     Socket socket; // socket de conexão.
     public void run(); // método principal da classe.
}
```

Class AllResults: classe que contém todos os resultados dos testes e o endereço do computador onde foram feitos. Nesta classe são calculados os valores de cada teste em função do tempo.

```
class AllResults implements Serializable{ // declaração.
    double []results; // resultados.
    InetAddress address; // Endereço IP do computador.
```

```
    String computer; // nome do computador.
    double getResult(int i); // retorna o resultado do teste i.
    double []getResults(); // retorna todos os resultados.
    public InetAddress getHost();// retorna o endereço do computador.
    public String getComputer(); // retorna o nome do computador.
}
```

Class PriceCalculator: classe que calcula o preço do computador, a partir dos valores obtidos nos testes (benchmark).

```
public class PriceCalculator{ // declaração.
    double price; // preço calculado do computador.
    AllResults allTest; // objeto que encapsula todos os testes.
    // método que calcula o preço do computador.
    public void CalculatePrice();
    public double getPrice(); // retorna o valor calculado (preço).
}
```

Class TestParameters: classe que contém os valores constantes usados nos testes benchmark.

```
public class TestParameters{ // declaração.
    // Quantidade de operações de loop vazio.
    public final static int LIMIT_VAZIO          = 250000;
    // Quantidade de Loop em operações básicas de ponto flutuante.
    public final static int LIMIT_FLOAT_SIMPLE = 50000;
    // Quantidade de Loop em operações complexas de ponto flutuante
    public final static int LIMIT_FLOAT_COMPLEX = 10000;
    // Tamanho do pacote de dados usado para testar o Link
    public final static int LIMIT_STREAM         = 256;
    // Número de testes a fazer (3 de CPU, 1 de Link)
    public final static int NTEST                = 4;
    // Número de repetições de cada Teste
    public final static int TIMES                = 5;
     // Port usado para fazer o teste de enlace de comunicação.
     public final static int PORT_TEST_LINK      = 9901;
}
```

Class CPUTest classe que implementa os testes para a CPU.

```
public class CPUTest{ // declaração.
     // método para fazer o teste indicado por _value.
     public long Test(int _value);
}
```

Class LinkTest: classe que se encarrega de executar o teste de enlace de comunicação com o Servidor Web.

```
public class LinkTest{ // declaração.
     InetAddress AddressTest; // endereço IP do servidor web.
     public long Test(); // método principal da classe.
}
```

A.2 Package join.Allocator.Prices

Package que contém as classes que implementam a abordagem econômica para a alocação de recursos, assim como os objetos que representam os agentes nesta abordagem.

Class SortedList: classe que implementa uma lista de elementos ordenados para acesso rápido.

```
public class SortedList extends Vector  implements Serializable{ // declaraçã
    // declaração da constante ascendente.
    public static int ASCENDING = 1;
    //  declaração da constante descendente.
    public static int DESCENDING = -1;
    int order; // tipo de ordem desta lista.
    public int getOrder(); // retorna a ordenação desta lista.
    public void Update(); // atualiza ou reordena a lista.
    public boolean Add(Object o); // adiciona um elemento na lista.
    public Object getAt(int pos); // retorna elemento da posição pos.
    public Object getFirst(); // retorna primeiro elemento da lista.
```

```
    public Object getLast(); // retorna último elemento da lista.
}
```

Class ListPrice: classe que implementa a lista base de objetos ordenados, herda de SortedList, implementa os principais métodos para manipular objetos, tais como Add, Delete, Contain, getAt, etc, tendo em conta os objetos ItemHost e ItemBuddy implementados.

```
public class ListPrice implements Serializable, ListInterface{
    SortedList list; // lista ordenada.
    // retorna uma lista com os elementos menores ou iguais que limit,
    //  em ordem ascendente.
    public SortedList getBeforeOfAscending(double limit,int load);
    // retorna uma lista com os elementos menores ou iguais que limit,
    // em ordem descendente.
    public SortedList getBeforeOfDescending(double limit,int load);
    // retorna uma lista com os elementos maiores que limit,
    // em ordem ascendente.
    public SortedList getAfterOfAscending(double limit,int load);
    // retorna uma lista com os elementos maiores que limit,
    // em ordem descendente.
    public SortedList getAfterOfDescending(double limit,int load);
    // retorna a possição do objeto pb.
    public int contain(ItemBase pb);
    // retorna um objeto da posição pos da lista.
    public ItemBase getAt(int pos);
    // apaga o elemento da posição pos da lista.
    public boolean removeAt(int pos);
    // atualiza o elemento O na lista.
    public boolean Update(Object O);
}
```

Class ListPriceHosts: classe que contém os elementos ItemHost. Mantém ordenados os hosts pelo preço.

```
public class ListPriceHosts extends ListPrice{ // declaração.
```

```
    public ItemHost getFirstHost(); // retorna primeiro ItemHost da lista.
    public ItemHost getLastHost(); // retorna último ItemHost da lista.
    // retorna uma lista de ItemHost com preço menor que limit.
    public SortedList getItems(double limit,int type,int what,int load);

}
```

Class ListBuddySet: classe que contém os elementos ItemBuddy (usados para representar os vizinhos dos grupos).

```
public class ListBuddySet extends ListPrice{ // declaração.
    public ItemBuddy getFirstBuddy(); // retorna o primeiro ItemBuddy.
    public ItemBuddy getLastBuddy(); // retorna o último ItemBuddy.
}
```

Class ItemBase: classe base que serve para representar elementos base da SortedList.

```
public class ItemBase extends Object implements Serializable,Compare{
    String Address; // endereço do ItemBase;
    Date date; // data de criação.
    Date current; // data da última atualização.
    public Date getTimeStamp(); // obtém data da última atualização.
}
```

Class ItemHost: classe que representa um computador (Host), com seu endereço IP, preço, estado de carga, e número de tarefas executando atualmente.

```
public class ItemHost extends ItemBase{ // declaração.
    double FirstPrice; // preço inicial.
    double CurrentPrice; // preço atual.
    int NTasks; // número de tarefas.
    double MoneyEarn; // créditos ganhos.
    // retorna o número de tarefas atualmente executando.
    public int getNTasks();
    // atualiza o preço de acordo ao número de tarefas.
    public void UpdatePrice();
```

```
        // incrementa o número de tarefas em n.
        public int incTasks(int n);
        // decrementa o número de tarefas em n.
        public int decTasks(int n);
        //retorna o preço inicial do computador.
        public double getFirstPrice();
        // retorna o preço atual.
        public double getCurrentPrice();
        // adiciona os créditos more aos ganhos.
        public void Earn(double more);
        // retorna a quantidade de créditos ganhos.
        public double getMoneyEarn();
    }
```

Class ItemBuddy Classe que representa um *buddy* ou vizinho do grupo.

```
    public class ItemBuddy extends ItemBase{ // declaração.
        int IDBuddy;
        public int getIDBuddy(); // retorna o ID do buddy.
    }
```

Apêndice B

Funcionamento do simulador

O simulador foi construído sobre a versão JoiN 0.1. As indicações aqui apresentadas para executá-lo provavelmente sofram algumas mudanças nas próximas versões do JoiN.

B.1 Configuração do ambiente de trabalho

As indicações a seguir devem ser observadas para configurar o ambiente JoiN, assim como para executar o programa simulador.

- Colocar no seu arquivo de recursos .cshrc as seguintes instruções:

```
setenv CLASSPATH .:/proj/join/public_html/alocacao/pedro/run
setenv JAVA_HOME /n/lang/JDK
setenv JDK_HOME /n/lang/JDK
```

- Fazer login no servidor Web do DCA (computador www.dca.fee.unicamp.br).
- Executar :

```
cd /proj/join/public_html/alocacao/pedro/run
java join.JoinServer
```

Isto executará o servidor e o deixará pronto para receber máquinas disponíveis.

- No seu diretório .netscape, procurar o arquivo preferences.js

```
cd
cd .netscape
textedit preferences.js &
```

Adicionar no final desse arquivo a linha
`user_pref("signed.applets.codebase_principal_support",true);` .
Isto permitirá que o Netscape interprete applets não assinados digitalmente como se estivessem assinados (para evitar ter que assinar os applets sempre que se faz alguma mudança. Esta facilidade existe precisamente para a etapa de desenvolvimento dos projetos).
Obs: no momento de modificar o arquivo preferencer.js, não pode haver nenhum Netscape sendo executado.

- Executar no mínimo dois processos Netscape em computadores diferentes (Netscape versão 4.xx ou superior em ambiente Unix).

- Carregar a página http://www.dca.fee.unicamp.br/projects/join/alocacao/pedro/run em cada instância do Netscape.

- Fazer click sobre o único link da página.

- Abrir o "Java Console"do Netscape, para observar melhor o progresso das ações. Além do Java Console, o simulador apresenta um janela por cada aplicação sendo alocada.

- Quando o applet estiver sendo carregado, conceda permissões "extraordinárias", tais como permissão para abrir e aceitar conexões com qualquer outra máquina, apertando o botão "Grant".

Seguindo estes passos, não devem existir dificuldades para depois de alguns segundos começar a alocação das aplicações submetidas ao simulador.

B.2 Configuração do simulador

A configuração das aplicações submetidas ao simulador é feita no arquivo AppToAllocate.dat que está no diretório `/proj/join/public_html/alocacao/pedro/run`. Neste arquivo devem ser escritas as linhas que representam as aplicações a serem alocadas. Cada linha representa uma aplicação com quantidade de tarefas a serem alocadas e orçamento por tarefa. Por exemplo, para alocar duas aplicações, uma que precisa criar 50 tarefas com um orçamento de 10 créditos por tarefa e outra aplicação que precisa criar 100 tarefas com orçamento de 20 créditos por tarefa, deve se criar o seguinte arquivo:

```
50 10
100 20
```

Desta maneira o programa simulador lê o arquivo e realiza a alocação. Todas as simulações são alocadas em 50 computadores, cujos preços variam entre 10 e 50. Modificações sobre quantidade de computadores e preços poderão ser feitos diretamente no arquivos fonte do simulador (nas classes CreatorComputer e ComputerList respectivamente). No diretório `/proj/join/public_html/alocacao/pedro/run/programs/Allocator` podem ser encontrados os arquivos fonte. O comando a usar para compilar é `java` *.java Prices/*.java.

Os resultados da simulação são gravados em arquivos em formato ascii (finalizados com .TXT), dentro do diretorio `/proj/join/public_html/alocacao/pedro/run`. É criado um arquivo ComputerXX.TXT por cada computador para armazenar os preços em cada iteração do processo alocador (XX corresponde ao número do computador, de 01 a 50). Após a simulação acabar, os arquivos .TXT devem ser transferidos para outro diretório, já que a próxima simulação a ser feita escreverá sobre os arquivos antigos, perdendo-se os dados.

O número de iterações por aplicação e o número de computadores nos quais foi alocada a aplicação está no arquivo GlobalIteracoes.TXT.

O controle do tipo de início de alocação é feito pela variável $INIT\_ALLOCATION$ do arquivo fonte ValuesAllocator.java. Para início simultâneo, esta variável deve ser igual a $CONCURRENT$, para início aleatório a $RANDOM$ e para início sequencial deve ser igual a $SEQUENTIAL$ (por exemplo $INIT\_ALLOCATION = RANDOM$ configurará o simulador para usar início aleatório[1]).

[1] Início aleatório é o valor por default.

Após feita uma modificação no código fonte do programa, todos os arquivos devem ser compilados novamente (usando o comando `java *.java Prices/*.java`). Estas instruções devem ser seguidas rigorosamente para obter sucesso na execução do programa simulador.

Apêndice C

Artigo publicado referente ao trabalho desta dissertação

- Pedro W. Chávez, Marco A. Amaral Henriques. *Alocação de Recursos Computacionais Baseada em Modelo Econômico para Sistemas de Processamento Massivamente Paralelo Virtuais*, Anais do XXVI Seminário Integrado de Software e Hardware (SEMISH), XIX Congresso da SBC, Rio de Janeiro, Jul. 19-22, 1999, pp. 211-200.