UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO DE BALANCEAMENTO DE CARGA PARA O SISTEMA OPERACIONAL AURORA

Valdemir Ferreira de Almeida

Florianópolis – SC
Fevereiro – 2003

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO DE BALANCEAMENTO DE CARGA PARA O SISTEMA OPERACIONAL AURORA

Valdemir Ferreira de Almeida

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Florianópolis – SC
Fevereiro – 2003

UM MODELO DE BALANCEAMENTO DE CARGA PARA O SISTEMA OPERACIONAL AURORA

Valdemir Ferreira de Almeida

Esta Dissertação foi julgada adequada para a obtenção do título de **Mestre em Ciência da Computação**, Área de Concentração - **Sistemas Operacionais** e aprovada em sua forma final pelo **Programa de Pós-Graduação em Ciência da Computação.**

_____
Prof. Dr. Fernando O. Gauthier

Banca Examinadora

_____
Prof. Dr. Luiz Carlos Zancanella
Orientador

_____
Prof. Dr. Rômulo Silva de Oliveira

_____
Prof. Dr. Antonio Augusto Medeiros Frolich

_____
Prof. Dr. Thadeu Botteri Corso

“Uma caminhada de mil milhas, começa com o primeiro passo”. Provérbio Chinês

A minha esposa Juliana Aparecida Strobel Moreira da Silva Almeida e aos meus filhos, Lucas Strobel Moreira Ferreira de Almeida e Letícia Strobel Moreira Ferreira de Almeida, pelo apoio e entendimento de minha distância durante a elaboração deste.

A minha mãe Naide Araújo de Almeida e meu Pai Paulo Ferreira de Almeida, pelas constantes motivações.

Ao meu professor orientador, ***Prof. dr. Luiz Carlos Zancanella por aceitar o desafio e compartilhar os seus conhecimentos, que foram de vital importância para a conclusão desta dissertação.***

Aos meus irmãos e amigos que diretamente e/ou indiretamente tiveram sua parcela de contribuição, em especial aos colegas de trabalho Sandro Luis Brandão Campos, Robson Dolores Dias e Luciana Kondo Otaque.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

NOW	Network Workstations
SISD	Single Instruction Single Data
MIMD	Multiple Instruction Multiple Data
SIMD	Single Instruction Multiple Data
FIFO	First-In-First-Out
SJF	Shortest-Job-First
NUMA	NonUniform Memory Access
UMA	Uniform Memory Access.
BCA	Balanceamento de Carga do Sistema Operacional Aurora

RESUMO

O Balanceamento de Carga é um dos principais itens que influenciam o desempenho de uma plataforma computacional distribuída (composta por vários elementos de processamento). Independentemente se essa plataforma é utilizada para computação paralela ou para sistemas distribuídos, há várias maneiras de se distribuir e balancear os processos, cada uma apresentando vantagens e desvantagens. Para determinar corretamente qual política de balanceamento de carga empregar, o usuário deve ser capaz de comparar diferentes propostas e escolher aquela que lhe é mais apropriada. No entanto, essa escolha não é trivial porque a literatura não apresenta uma terminologia consistente e também porque as taxonomias divergem na abordagem empregada para classificar as diferentes políticas de balanceamento. Este trabalho tem por objetivo central, apresentar um modelo de Balanceamento de Carga de processos para um Sistema Operacional Reflexivo, no caso, o Aurora. Também serão destacadas outras questões relativas à área, como por exemplo: componentes de um algoritmo de Balanceamento de Carga com suas diferenças e algumas taxonomias da área.

.

ABSTRACT

The load balancing is the most main items that influence the performance of shared computation platform (compose by many processing elements). Independently if this platform is used by parallel computation or shared systems, there are many way how to shared out and to balance process, each one showing advantages and disadvantages. For determinate exactly which politics of load balancing to use it, the user must be able to compare differents offers and choice the better for him. Meanwhile, this choice isn't common because the literatures don't show a consistent terminology and because this taxonomy is different about the treatment appointed for classify the differents politics of balancing too. This work has how central objective, showing a model of Load Balancing of process for Reflexive Operational System, this case, the Aurora. Will be emphasize other questions about this area too, for example, components of a algorithm of Load Balancing with its differences and some taxonomies about this area.

Capítulo 1

INTRODUÇÃO

Segue uma pequena introdução sobre o tema abordado nesta dissertação, mostrando e enfocando as características de Redes de Computadores distribuídas e paralelas, alguns aspectos de seu histórico, as suas vantagens e a importância de Balanceamento de Carga nestes ambientes.

Com a crescente popularização dos Sistemas de Informação, com o uso cada vez mais acentuado de Sistemas Especialistas que requerem máquinas sempre mais poderosas, os Sistemas Operacionais, passaram e passarão por mudanças complexas e paradoxalmente novas para atender à crescente mudança no modelo tecnológico. Os Sistemas Operacionais inicialmente foram escritos para atender máquinas pequenas e centralizadas, migrando para máquinas maiores e com complexos sistemas de controle, tais como: Backup, Controles de Carga de Processamento, Controle de Concorrência, Controle de periféricos, etc... Mas, ainda assim máquinas que "rodavam centralizadas". Uma das alternativas para a resolução do problema foram os crescentes softwares para processamento Paralelo, o que exigiu que uma gama de diferentes Sistemas Operacionais fossem criados para atender a estes recursos. Com a crescente demanda para aproveitamento dos recursos instalados, surgiram os Sistemas Operacionais para Sistemas Distribuídos, estes utilizam e proporcionam o máximo de uso de recursos instalados em Redes de Computadores.

Hoje em dia é bem aceito o fato de que as redes de computadores estão presentes em qualquer instituição que utilize sistemas computacionais. No meio científico, inúmeros artigos mostram que nestas redes a taxa de ociosidade é muito grande [DOU96], visto que freqüentemente o número de estações disponíveis é maior que o de usuários.

Atualmente, nota-se que o potencial de processamento existente em uma rede de microcomputadores, custando algumas dezenas de milhares de dólares, é muito maior do que o encontrado em supercomputadores, com o custo na casa dos milhões [SIN94]. Porém faz-se necessário utilizar um sistema operacional que consiga fazer todas as máquinas da rede trabalharem em conjunto, aproveitando o máximo possível deste potencial.

Vários sistemas operacionais já foram criados dentro do conceito de Sistema Operacional Distribuído, um sistema que gerencia todas as máquinas da rede de modo a funcionarem como se fosse uma só. Como exemplos, podemos citar os sistemas Amoeba [TAN95], Sprite [OUS88] e o Accent. Nestes sistemas, as estações são tratadas como nós processadores de um sistema maior, e os processos do sistema podem

executar em qualquer deles, independentemente da máquina onde foram inicialmente gerados.

Mesmo com a existência destes sistemas mais avançados, o panorama encontrado ainda é o de redes de estações de trabalho independentes, tendo como sistema operacional algum tipo de clone do Unix. Neste ambiente se tem em geral uma alta taxa de ociosidade, pois nem todas as estações são utilizadas ao mesmo tempo e na mesma intensidade.

Os avanços observados no hardware têm tornado os Sistemas Computacionais Distribuídos cada vez mais viáveis para uma vasta comunidade de usuários. Concomitantemente, o software, através dos sistemas operacionais, tem fornecido uma visão mais apropriada de tais sistemas [ESK89] [COU94] [TAN95], facilitando o seu uso e favorecendo a sua disseminação.

É freqüentemente desejável atingir uma distribuição de carga (ou Balanceamento de Carga) uniforme nas máquinas pertencentes a um Sistema Computacional Distribuído, de tal forma que não haja máquinas ociosas enquanto outras estão sobrecarregadas. O problema de distribuição de carga pode envolver, entre outros fatores, a re-alocação dinâmica da carga de trabalho entre os elementos de processamento do sistema. Distribuição dinâmica de carga é usada, portanto, para obter um melhor tempo de resposta e para aumentar a utilização dos recursos atingindo assim, uma melhoria no desempenho global do sistema.

Devido então à exigência de ambientes mais escalares, vivemos um momento de grandes mudanças em Sistemas Operacionais, justamente visando um melhor uso do ambiente instalado, facilidade de administração e interfaces amigáveis. Para atender a estes requisitos, surge então um novo enfoque para paradigma de Sistemas Operacionais, este enfoque se demonstra nos Sistemas Operacionais Reflexivos, o qual possuímos um exemplo deste tipo de Sistema que é o Aurora, proposto por [Zan97] e que possui características de componentização, reutilização de códigos, encapsulamento e execução de computação sobre ele mesmo, a reflexão.

Diante do exposto, este trabalho tem por objetivo, estudarmos e refletirmos num modelo de controle de *Balanceamento de Carga para o Sistema Aurora.*

Capítulo 2

PLATAFORMAS COMPUTACIONAIS

Neste capítulo apresentamos uma visão sucinta dos ambientes tecnológicos atuais, com ênfase em ambientes distribuídos e paralelos, mostrando os conceitos, definições e caracterizações desses ambientes.

Plataforma computacional é a união do Hardware, software básico, topologia e demais elementos que compõe o ambiente computacional que se está utilizando.

2.1 Plataforma computacional distribuída

Uma plataforma computacional distribuída e heterogênea é uma coleção de elementos de processamento, fisicamente distribuídos, conectados por uma rede de comunicação e utilizada com objetivos específicos. Essa definição apesar de assemelhar-se à definição de redes de computadores é mais abrangente, porque os elementos de processamento podem pertencer a uma NOW (Network of Workstations) composta por computadores pessoais e/ou estações de trabalho [AND95], ou até mesmo a computadores paralelos, todos podendo estar interligados e formando uma única plataforma computacional.

Considerando as plataformas distribuídas, Tanenbaum [TAN95] cita que o termo "multiprocessadores" designa plataformas computacionais com mais de um processador, os quais compartilham o mesmo endereço de memória (possuem memória compartilhada ou centralizada). O termo "multicomputadores" designa plataformas onde cada processador possui a sua própria memória local, formando uma memória distribuída. Ainda segundo Tanenbaum [TAN95], as plataformas podem ser classificadas ou como fracamente acopladas (loosely-coupled) ou então como fortemente acopladas (tightly-coupled). Nas plataformas fortemente acopladas, o envio de uma mensagem entre dois processadores apresenta um atraso (delay) pequeno e uma taxa de transferência alta (número de bits enviados), quando comparadas às plataformas fracamente acopladas, onde o envio de mensagens possui um atraso maior e uma taxa de transferência mais baixa.

Em geral, os computadores paralelos são bons exemplos de plataformas fortemente acopladas e os sistemas distribuídos bons exemplos de plataformas fracamente acopladas (principalmente quando estes são constituídos por estações de trabalho conectadas por uma rede de comunicação padrão, como a ethernet).

Caso a plataforma distribuída e heterogênea possua na gerência das suas

atividades um sistema operacional distribuído, tem-se um "sistema distribuído", caso contrário tem-se uma plataforma computacional distribuída fisicamente e com possíveis arquiteturas e/ou capacidades de processamento. Essa plataforma computacional pode ter diferentes objetivos, os quais podem ser desde o simples compartilhamento de recursos (como unidades de disco e impressoras) ou então a execução de aplicações paralelas através da união lógica (ou virtual) dos processadores.

As plataformas distribuídas e heterogêneas possuem características interessantes, as quais contribuem enfaticamente para a sua grande utilização e, algumas vezes, particularizam a atividade de balanceamento de carga, ou seja, possuem uma solução de Balanceamento de Carga nativa. A melhor relação custo x benefício é um dos fatores responsáveis pela grande utilização, principalmente quando a plataforma é composta por Network WorkStations - NOWs. Russ [RUS97] cita que a principal razão para a grande utilização das NOWs é o seu baixo custo relativo e não o seu desempenho. No entanto, se forem consideradas as NOWs já instaladas, essa grande quantidade de recursos oferece um grande potencial para processamento. Anderson [AND95] demonstra em seus estudos que mesmo em horários normais de trabalho, mais de 60% dos computadores estavam 100% disponíveis, tornando as NOWs fortes candidatas ao processamento paralelo. A utilização das NOWs para o processamento paralelo agregou novos fatores ao balanceamento de carga de processos, os quais não apresentava a mesma importância tanto na computação seqüencial feita em plataformas distribuídas, quanto na computação paralela feita em computadores "realmente" paralelos. As características multiusuária e heterogênea das NOWs são bons exemplos desses fatores que obrigaram as políticas de balanceamento de carga adaptarem-se para uma melhor distribuição dos processos.

2.2 Plataforma Computacional Paralela

No que diz respeito à demanda por processamento computacional (desempenho) esta exigência é acirrada e vem impulsionando os patamares de desempenho que a tecnologia pode oferecer. Até recentemente, estes constantes aumentos por demandas de processamento eram atendidos pelo aumento do

desempenho tecnológico dos atuais computadores convencionais (computadores seqüenciais). Entretanto, há uma tendência de saturação das possíveis alternativas de aumento do desempenho tecnológico. Devido a essa tendência de saturação, concomitantemente aos esforços convencionais, pesquisadores passaram a investigar um novo paradigma computacional - Processamento Paralelo - objetivando atender à prevista tendência de crescimento da demanda por recursos computacionais.

A postura do referido paradigma é a utilização de mais de uma máquina (processador), concorrentemente, na solução de um determinado problema (processamento paralelo). O atual estágio técnico-industrial é tal, que a duplicação de máquinas é possível a custos aceitáveis (em termos de hardware). Computadores individuais (máquinas) são agregados atingindo uma solução (máquina paralela) de custo/desempenho pré-estabelecido. Usando esta abordagem, o aumento por demanda de processamento é resolvido pela adição de novas máquinas ao sistema.

A taxonomia usualmente utilizada para classificar as arquiteturas paralelas foi introduzida por Flynn em 1972 [FLY72]. De acordo com esta notação, as duas principais características que definem um computador são o número de instruções que este executa em cada momento e o número de dados operados em simultâneo. Surge assim os modelos SISD (single instruction single data), MIMD (multiple instruction multiple data) e SIMD (single instruction multiple data).

Os ambientes de computação paralela disponíveis nos dias de hoje enquadram-se em vários tipos de arquiteturas, o que se traduz por soluções tecnológicas das mais variadas, preços e vasta gama de funcionalidade e desempenho. Desde as soluções de baixo custo e altamente versáteis oferecidas por clusters de workstations, até às máquinas massivamente paralelas de alto custo que podem dispor de Gbytes de RAM e centenas de processadores.

Os sistemas de computação paralela dividem-se em dois grandes grupos: as máquinas paralelas e os clusters de workstations. O que de fato distingue estes dois sistemas, é o estilo de acoplamento que utilizam na ligação entre as várias unidades de processamento. Os clusters de workstations caracterizam-se pela utilização de redes de comunicação local (Ethernet, ATM, FDDI), o que lhes confere capacidades para se

apresentarem fisicamente dispersas.

2.3 Multicomputador

O modelo de máquina paralela, chamado multicomputador atende aos requisitos mostrados. Um multicomputador é composto de vários computadores de Von Neumann, que são os nós, ligados por uma rede de interconexão. Cada computador executa seu próprio programa. Este programa pode acessar uma memória local e pode enviar e receber mensagens pela rede. Mensagens são utilizadas para comunicação com outros computadores ou, equivalentemente, a ler e escrever em memórias remotas. Na rede ideal, o custo de enviar uma mensagem entre dois nós é independente tanto da localização do nó quanto de outros tráficos na rede, porém depende do tamanho da mensagem.

O multicomputador, um modelo ideal de computador paralelo. Cada nó consiste em uma máquina de Von Neumann: uma CPU e memória. Um nó pode se comunicar com outros nós enviando e recebendo mensagens por uma rede de interconexão. Um atributo que define o modelo de multicomputador é que acessos à memória local (mesmo nó) são mais baratos que acessos remotos a memória (nó diferente). Ou seja, ler e escrever são menos custosos que enviar e receber. Portanto, é desejável que acessos a dados locais sejam mais freqüentes que acessos a dados remotos. Esta propriedade, chamada localidade, é também requisito fundamental em software paralelo, além de concorrência e escalabilidade. A importância da localidade depende da relação entre o custo de acesso remoto e local. Esta relação pode variar de 10:1 a 1000:1 ou mais, dependendo da performance relativa do computador local, da rede, e dos mecanismos utilizados para mover os dados na rede.

2.4 Máquinas Paralelas

Dentro das máquinas paralelas devemos distinguir as de pequenas dimensões (workstations multiprocessador) que disponibilizam poucos processadores (2-8) e memória central na ordem das dezenas de Mbytes, e as máquinas massivamente paralelas, que podem oferecer centenas de processadores e Gbytes de memória RAM. O

mercado destinatário para este tipo de máquinas é bastante diversificado. O mercado de workstations pretende acima de tudo fornecer ambientes flexíveis e serviços (ftp, http, I/O). As máquinas massivamente paralelas destinam-se ao mercado que exige cálculo intensivo em tempo hábil, como cálculo científico, simulação de embates, simulação de dinâmica de fluidos, simulação de reações químicas e nucleares, previsão meteorológica entre outras. Grande parte das arquiteturas paralelas adaptam soluções de memória partilhada ou de memória distribuída, porém, é muito comum encontrarem-se modelos híbridos.

2.5 Memória Partilhada

Estas arquiteturas são constituídas por um espaço de endereçamento que é acessível diretamente por um conjunto de processadores. Normalmente utilizam processadores vetoriais. As máquinas baseadas nesta arquitetura são pouco escaláveis devido ao estrangulamento que se verifica no acesso à memória quando o número de processadores aumenta. Devido ao fato de se utilizar memória partilhada, facilmente se constroem entre os processadores canais de comunicação que oferecem altas larguras de banda e pequenas latências. O fato de estas máquinas disponibilizarem memória partilhada, torna-as fáceis de programar.

2.6 Memória Distribuída

O modelo de memória distribuída propõe que a máquina paralela seja constituída à custa de vários elementos de processamento com memória local interligados por uma rede de comunicação. O êxito desta arquitetura deve-se principalmente a dois fatores: ser potencialmente mais escalável, e a tecnologia das redes de comunicação terem evoluído positivamente nos últimos anos.

As arquiteturas de memória distribuída apresentam-se nos dias de hoje como a tecnologia de base para o desenvolvimento de grandes sistemas paralelos.

2.7 Clusters de Workstations

À medida que as redes de comunicação se vão tornando cada vez mais rápidas, os clusters de workstations e de computadores pessoais apresentam-se cada vez mais como uma solução tecnológica para se integrarem em sistemas paralelos [GEI94]. Uma outra razão que torna esta solução bastante atrativa prende-se com o baixo custo das workstations e dos computadores pessoais, e a facilidade de expansão destes mesmos sistemas.

Capítulo 3

BALANCEAMENTO DE CARGA

Segue uma pequena introdução sobre o tema abordado nesta dissertação, mostrando e enfocando as características de Redes de Computadores distribuída e paralela, alguns aspectos de seu histórico, as suas vantagens e a importância de Balanceamento de Carga nestes ambientes.

.

Os principais objetivos dos Sistemas Distribuídos têm convergido para a busca de alto desempenho. Em geral, o balanceamento de carga (load balancing) e, em particular, a migração de processos (process migration), emergem como solução para esse impasse. Faz-se necessário garantir em sistemas com grande número de processadores ou de estações que todos os elementos de processamento sejam bem aproveitados, a fim de que a carga seja distribuída de maneira uniforme.

Pode acontecer de alguns processadores completarem suas tarefas antes de outros e tornarem-se ociosos porque a carga não está igualmente distribuída ou alguns processadores são mais rápidos que outros (ou as duas situações). Idealmente, o desejável é que todos processadores operem continuamente nas tarefas de modo a ter o tempo de execução mínimo. Balanceamento de carga é atingir este objetivo, pela distribuição regular das tarefas nos processadores, conforme figura 3.1.

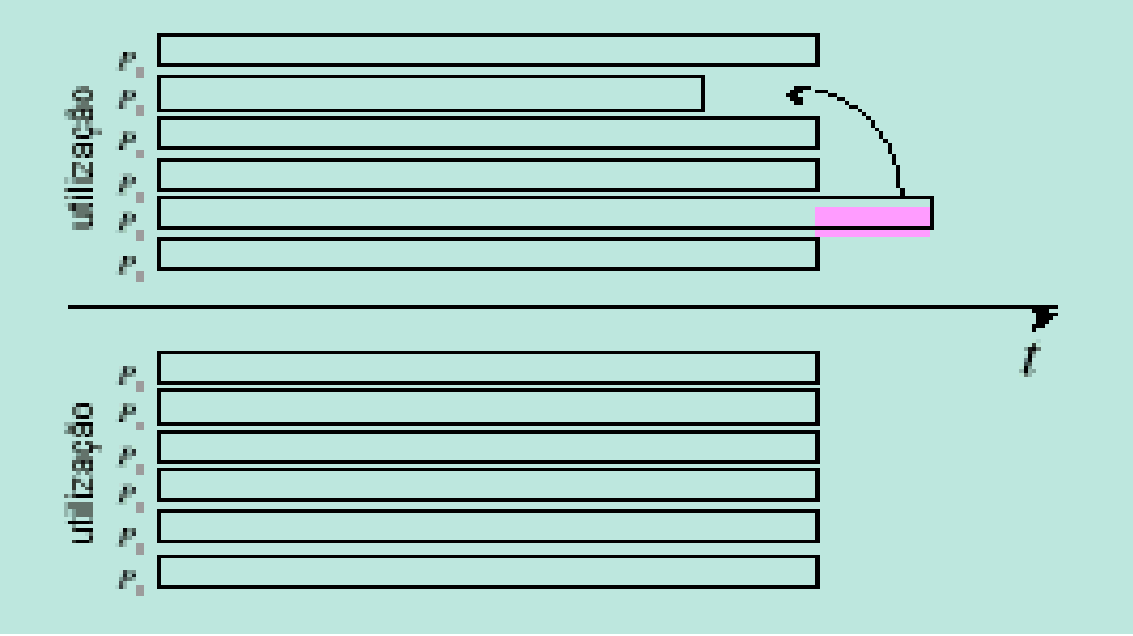


Figura 3.1 – Balanceamento de Carga

Balanceamento de carga é útil quando a quantidade de trabalho não é conhecida previamente à execução. Também serve para diminuir o efeito da diferença de velocidade dos processadores mesmo quando a quantidade de trabalho é conhecida previamente.

O Balanceamento de carga pode ser obtido estaticamente antes da execução de qualquer processo ou dinamicamente durante a execução dos processos. Balanceamento estático é conhecido como mapeamento.

Como dito anteriormente, um sistema multiprocessado por definição, consiste em diversos processadores que podem ser organizados como um conjunto de estações de trabalho, um pool de processadores, ou de alguma forma híbrida. Sistemas multiprocessados suportam multiprogramação e necessariamente precisam de um algoritmo para decidir qual dos processos prontos deve ser executado. A estes algoritmos de decisão é dado o nome de Balanceamento de Carga.

Segundo [TAN92], em todos os casos é necessário que haja algum algoritmo de Balanceamento para decidir qual processo deve ser executado. No caso de estações de trabalho, a questão é quando se deve executar um processo localmente, e quando devemos procurar por uma estação ociosa. Já para o pool de processadores deve ser tomada uma decisão a cada novo processo.

Os objetivos de um balanceador segundo [SHA93] são descritos a seguir:

- **Eficiência** - Uma maneira de definir eficiência é medir a produção. A produção é o número de processos que passam através de um sistema por unidade de tempo. Temos uma produção baixa quando foram completados poucos processos e, uma produção alta significa que muitos processos foram completados. No segundo caso, mais usuários conseguem ter seu trabalho realizado. Conclui-se que devemos tentar conseguir a produção mais alta possível.
- **Capacidade de Resposta** - Usuários interativos tem que possuírem respostas rápidas: ninguém é obrigado a ficar sentado diante de um terminal por meia hora esperando que um programa de cinco minutos seja executado.

Podemos pensar que a capacidade de resposta seja a mesma coisa que produção. Afinal, se o sistema pode responder rapidamente a cada um dos processos, mais processos poderão passar pelo sistema. Infelizmente, isso nem sempre é verdade. Um sistema operacional pode maximizar o resultado dando atenção especial aos processos pequenos e ignorando os grandes. Mas assim os processos demorados não vêem o sistema com alta capacidade de resposta**.**

- **Consistência** - As respostas de um sistema não devem variar, ou seja, a execução de um processo às 3 horas da tarde deve ter a mesma resposta se executado às 10 horas da noite. Se as respostas do sistema variarem muito, os usuários nunca saberão o que esperar então, o computador se torna não confiável.
- **Manter os Processadores Ocupados** - Um outro objetivo é que um sistema operacional deve manter seus recursos ocupados. Os recursos estão lá para serem utilizados e não poderiam ficar ociosos.
- Ainda podemos citar **Justiça,** para garantir que todos os processos do sistema tenham chances iguais de uso do processador.

Capítulo 4

TAXONOMIAS PARA BALANCEAMENTO DE CARGA

A seguir estaremos abordando as diferentes taxonomias para Balanceamento de Carga, com o objetivo de definirmos as políticas de Balanceamento do sistema Operacional Aurora.

Para Baumgartner; Wah [BAU91] o objetivo de uma taxonomia deve ser aumentar e organizar o "conhecimento" sobre uma classe de problemas, especificando-a e mostrando o relacionamento entre problemas. Os autores apontam ainda, no mínimo quatro características desejáveis de uma taxonomia:

(1) identificar características significantes: pois contribuem para uma solução eficiente;

(2) mostrar claramente o relacionamento entre os problemas: a solução de um problema pode ser semelhante à solução de outro problema que esteja relacionado com o primeiro;

(3) capaz de expandir e contrair: permite especializar a taxonomia a determinadas situações, onde pode haver a necessidade de um maior número de detalhes ou não, permitindo assim, que a taxonomia reduza a sua complexidade, caso seja possível;

(4) separar a especificação do problema da sua solução: essa separação permite comparar diversas soluções para o mesmo problema.

4.1 Taxonomia de Casavant; Kuhl (1988)

O trabalho de Casavant; Kuhl [CAS88] teve por objetivo fornecer uma estrutura adequada para que os trabalhos até então desenvolvidos (1988) pudessem ser comparados, e também para que trabalhos futuros pudessem ser classificados e discutidos. A taxonomia proposta é uma taxonomia híbrida entre hierárquica e plana. São definidos dois conjuntos, um com termos que se relacionam entre si (hierárquica) e outro conjunto de termos que pode relacionar-se com quaisquer outros, sem que haja obrigatoriamente numa hierarquia. Segundo os autores, essa taxonomia além de ser direcionada ao balanceamento de processos também é aplicável para qualquer conjunto de ferramentas de software de gerenciamento de recursos.

A classificação hierárquica é mostrada na Figura 4.1 e a seguir são feitas algumas discussões tanto sobre a taxonomia hierárquica quanto sobre a plana.

4.2 Taxonomia Hierárquica

- Local: Tanenbaum [TAN92] descreve que o escalonamento local é realizado

por um escalonador (*scheduler*), o qual é implementado por um algoritmo denominado algoritmo de escalonamento. Com as atuais plataformas monoprocessadas, multiusuária e multitarefa. Este algoritmo de escalonamento deve preocupar-se com vários fatores que não existiam nas antigas plataformas *batch* e monousuárias, onde o processador deveria simplesmente executar o próximo processo, formado através de cartões perfurados e posteriormente apresentar os resultados obtidos.

Além de permitir que os processos sejam executados no processador, um algoritmo de escalonamento local deve possuir alguns objetivos para poder ser considerado um bom algoritmo de escalonamento. Embora algumas vezes contraditórios, são exemplos de objetivos: (1) imparcialidade: ter certeza que todos os processos terão acesso adequado ao processador; (2) eficiência: manter o processador 100% do tempo ocupado; (3) tempo de resposta: diminuir o tempo de resposta para os usuários interativos; (4) *turnaround time*: diminuir o tempo que os usuários de aplicações *batch* devem esperar pela saída dos resultados, desde a submissão; e (5) *throughput* do sistema: aumentar o número de processos executados por unidade de tempo.

A preempção permite que os processos concorrentes ao processador recebam uma fatia de tempo para serem executados (*time slice*) e, assim, compartilhem o processador (*time sharing*) proporcionalmente à demanda das aplicações. Com a técnica de *time sharing* as aplicações que necessitam mais processamento que entrada/saída tem mais acesso ao processador quando comparadas às aplicações que necessitam mais entrada/saída que processamento.

A preempção também é uma técnica empregada no Balanceamento de Carga Global, onde um processo pode ser suspenso em um processador e colocado em execução em outro processador. A atividade de mudar um processo em execução para outro processador é conhecida como *migração de processos* e é bem mais complexa que a preempção feita apenas localmente, por ter que tratar problemas como a *localização de um processador destino*, uma possível transferência do estado do processo para o processador destino, entre outros.

Exemplos de algoritmos não-preemptivos podem ser: o First-In-First-Out

(FIFO) e o Shortest-Job-First (SJF). Exemplos de algoritmos preemptivos são: Round Robin, por Prioridades, por Múltiplas Filas e por Múltiplas Filas com Realimentação. Este trabalho não abordará os detalhes desses algoritmos, os quais encontram-se descritos em Machado; Maia [MAC92] e Tanenbaum [TAN92].

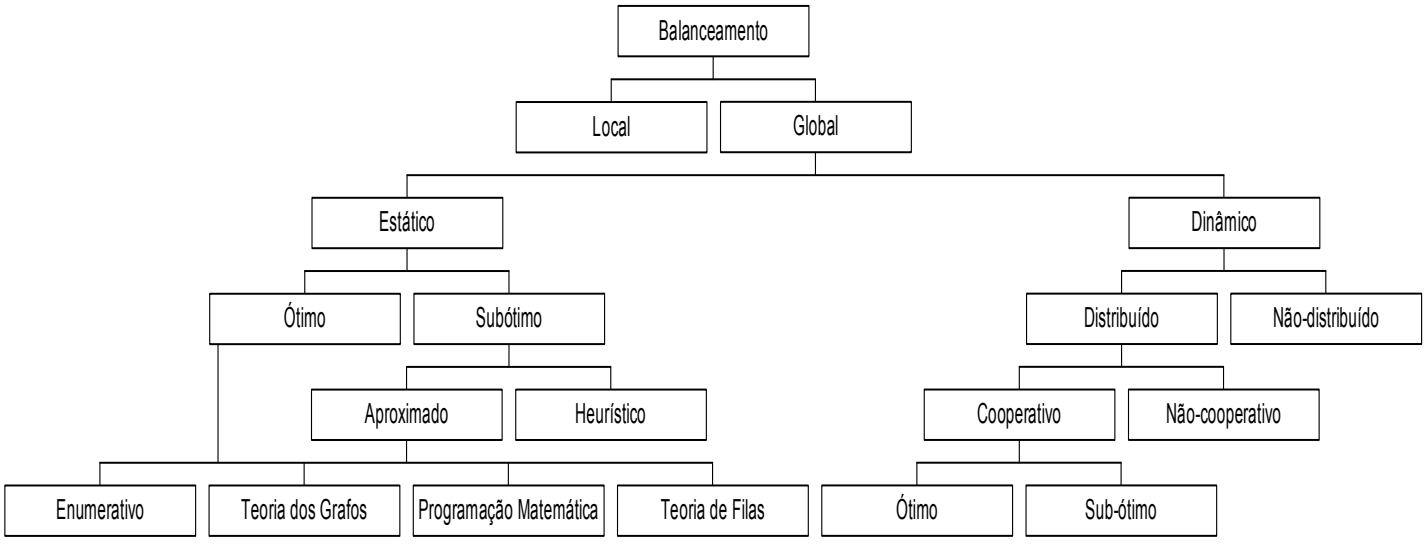
Global: Vários autores definem a atividade de Balanceamento de Carga de processos para plataformas computacionais distribuídas. Tanenbaum [TAN92] refere-se ao Balanceamento Global como "Alocação de Processadores", definindo essa atividade como sendo o estudo dos "algoritmos usados para determinar quais processos serão atribuídos para quais processadores". Mullender [MUL93] define o Balanceamento Global como sendo um serviço de um sistema distribuído, o qual atribui processos para processadores. Esse serviço é denominado de serviço de gerenciamento de processos, enfatizando assim, os processos e não os processadores.

Saphir [SAP95] define "escalonamento global" como sendo "a atribuição de recursos específicos para uma aplicação paralela". Essa definição é voltada aos processadores e não aos processos, fato encontrado principalmente nos trabalhos para a computação paralela. Shirazi; Hurson [SHI92] definem a área de Balanceamento Global como sendo o estudo da distribuição dos processos entre os elementos de processamento, para atingir alguma meta de desempenho, tais como: maximizar o *throughput* ou encontrar o menor *turnaround time*. Essa definição, além de ser voltada à aplicação, agrega também outra importante característica dos algoritmos de Balanceamento: os objetivos a que eles se propõem. Esses objetivos inseridos no algoritmo determinam a sua "personalidade" e restringem a sua área de atuação. Casavant; Kuhl [CAS88] define o Balanceamento Global, como um recurso para o gerenciamento de recursos. Esse gerenciamento é composto basicamente por uma política (ou mecanismo) usada para permitir o acesso e o uso de um recurso por seus vários consumidores, de modo eficiente. Os recursos significam, por exemplo: processadores, memórias, rede de comunicação, entre outros; e os consumidores são os usuários representados por suas aplicações seqüenciais distribuídas ou paralelas. Os eventos e o ambiente são as entradas para a Política de Balanceamento e o Balanceamento (ou mapeamento) é à saída do algoritmo.

Os eventos representam a menor entidade no Balanceamento, como por exemplo, processos. O ambiente engloba todas as demais características que podem afetar a escolha do Balanceamento. Dentre essas características estão: número de processadores, características físicas dos processadores, meio de comunicação, e os objetivos de desempenho. Para os autores os objetivos podem ser, por exemplo, o prazo máximo para a conclusão de uma aplicação de "tempo real" ou então determinar se é possível obter alguma melhoria de desempenho com o Balanceamento.

Estático & Dinâmico: as políticas de balanceamento estático e dinâmico pertencem ao balanceamento global e indicam o tempo que o balanceamento ou a decisão de atribuição serão efetuadas. A característica estática ou dinâmica de um balanceamento é uma das mais importantes e destacadas pela literatura, embora não haja um consenso a respeito desses termos. No balanceamento estático, o máximo de informações a respeito dos processos e dos processadores devem estar disponíveis em tempo de compilação/"linkedição". Assim, quando a aplicação for executada o balanceamento já está determinado e não mudará em tempo de execução.

Figura 4.1 – Taxonomia hierárquica de Casavant; Kuhl [CAS88].



Ótimo & Subótimo: um balanceamento ótimo pode ser realizado quando se possuem todas as informações necessárias a respeito dos processos que serão escalonados, assim como da plataforma computacional utilizada. Entretanto, a geração

de uma solução ótima é, na maioria dos casos, um problema NP-completo, inviabilizando, portanto, a sua realização. Segundo Shirazi [SHI95] balanceamentos ótimos são possíveis apenas em casos específicos, como por exemplo, quando o tempo de execução de todos os processos são os mesmos e apenas dois processadores são utilizados. Uma alternativa para o problema NP-completo das soluções ótimas são as soluções subótimas, as quais não se prendem apenas à melhor solução, ficando satisfeitas se "um bom balanceamento" for realizado. O problema aqui é determinar o que é "um bom balanceamento" e qual métrica avalia a solução encontrada.

Aproximado & Heurístico: o balanceamento aproximado utiliza algum modelo computacional formal para o algoritmo, porém, tem por objetivo alcançar um balanceamento aceitável, ao invés de buscar a melhor solução para o balanceamento. O balanceamento heurístico, por sua vez, utiliza parâmetros "especiais" os quais afetam a plataforma computacional de maneiras indiretas. Normalmente os parâmetros estão relacionados indiretamente com o desempenho e são mais simples para monitorar e calcular. Casavant; Kuhl [CAS88] cita como um exemplo de heurística, grupos de processos I/O Bound que são escalonados no mesmo processador e grupos de processos CPU Bound que são escalonados em processadores distintos. Essa heurística diminui diretamente a sobrecarga envolvida na troca de informações entre os processos, enquanto se beneficia do paralelismo. A intuição indica que esse balanceamento melhorará o desempenho da plataforma computacional, mas afirmar que existe um relacionamento direto entre a política empregada e os resultados desejados é uma tarefa um pouco mais complexa.

Ótimos & Subótimos Aproximados: independente se a política de balanceamento é ótima ou uma subótima aproximada, há quatro categorias básicas para os algoritmos de balanceamento: (1) enumeração do espaço da solução e busca, (2) teoria dos grafos, (3) programação matemática e (4) teoria das filas.

Distribuído & Não Distribuído: quando um balanceamento dinâmico é realizado, a responsabilidade por essa atividade pode ser apenas de um processador (balanceamento não distribuído) ou fisicamente distribuída entre os processadores (balanceamento distribuído). Casavant; Kuhl [CAS88] distingue os termos distribuído e

descentralizado, através de dois componentes: a responsabilidade e a autoridade. Quando a responsabilidade pelo balanceamento é compartilhada entre os diferentes processadores, o balanceamento é distribuído. Quando a autoridade de realizar o balanceamento for distribuída entre esses processadores, o balanceamento é descentralizado. As plataformas que possuem autoridade descentralizada devem possuir a responsabilidade distribuída, no entanto, é possível distribuir responsabilidade para, por exemplo, coletar informações e efetuar determinadas políticas, sem fornecer a autoridade necessária para mudar nenhuma decisão já tomada nem tão pouco realizar futuras decisões.

Cooperativo & Não Cooperativo: o balanceamento distribuído pode ser dividido considerando o grau de autonomia que cada processador possui para determinar como o balanceamento deve ser feito. No caso do balanceamento não cooperativo, um processador toma as suas decisões de maneira independente das ações dos outros elementos de processamento, não se preocupando com os efeitos das suas decisões sobre o restante da plataforma. No balanceamento cooperativo cada processador é responsável por realizar a sua porção do balanceamento, onde todos os processadores agem seguindo um único objetivo global. Assim, com o balanceamento cooperativo, os processadores não se preocupam apenas em melhorar o seu desempenho local, mas sim o desempenho do sistema como um todo.

4.3 Taxonomia de Wang; Morris (1985).

Wang; Morris [WAN85] propõem uma taxonomia direcionada apenas ao Balanceamento de carga em sistemas distribuídos locais, apresentando algumas limitações como: o uso de plataformas homogêneas, execução de processos independentes, seqüenciais e que, uma vez iniciados não podem ser suspensos e transferidos de processador (sem migração). A taxonomia divide os algoritmos de balanceamento através de duas características. A primeira distinção é em relação a quem tem a iniciativa de iniciar o Balanceamento. Para tanto, os processadores foram classificados ou como origem ou servidores (destino). Os processadores origem são aqueles que possuem processos para serem executados, enquanto os processadores

servidores representam aqueles que desejam executar os processos remotos. Um processador pode ser simultaneamente origem e servidor, também conhecido como simétrico [SHI95]. Se um processador origem é quem toma a iniciativa de procurar onde executar um processo, essa política é classificada como "iniciada pela origem" (*source-initiative*). Se um processador servidor é quem toma a iniciativa de "procurar por trabalho" (processos), essa política é classificada como "iniciada pelo servidor" (*server-initiative*). A segunda distinção refere-se à dependência por informação que a política de Balanceamento possui. Essa distinção indica a quantidade de informação que um processador origem possui (ou necessita) sobre os processadores servidores e vice-versa. Para determinar a quantidade de dependência por informação os autores criaram sete níveis (1-7), onde o número "um" representa a menor e o número "sete" a maior dependência. Para Wang; Morris [WAN85] no nível um estariam as políticas de balanceamento estáticas e a partir do nível cinco estariam as políticas de balanceamento dinâmicas.

4.4 Taxonomia de Baumgartner; Wah (1991).

O trabalho de Baumgartner; [BAU91] apresenta um levantamento sobre as pesquisas em algoritmos de balanceamento, analisa algumas taxonomias propostas até então (entre elas Casavant; Kuhl [CAS88] e Wang; Morris [WAN85] e propõe uma nova taxonomia para a área).

A primeira distinção feita é entre os balanceamentos determinísticos, não determinísticos e indecidíveis. O balanceamento determinístico tem, previamente, todas as informações necessárias para o balanceamento. Um balanceamento é determinístico se ele pode ser completamente definido e também se é possível de ser feito em função do tempo (processamento) e espaço (memória e disco). O balanceamento não determinístico pode ser completamente definido, entretanto, encontrar uma solução exata leva muito tempo ou necessita muito espaço ou ambos. Um balanceamento indecidível é aquele impossível de ser realizado, mesmo que haja infinitas capacidades de processamento e recursos de armazenagem. Baumgartner; Wah [BAU91] cita que na década de 60 e início de 70 havia um grande interesse sobre o balanceamento em

“ambientes de manufatura”, os quais eram predominantemente determinísticos, com um conhecimento completo sobre o tempo de chegada e de atendimento. Em sistemas de computação o balanceamento deve ser visto como não determinístico, em função de raramente haver uma informação exata sobre os processos, plataforma e objetivos que devem ser considerados. Para apresentar a nova taxonomia (chamada ESR), os autores discutem sobre dois grupos de taxonomias. O primeiro grupo relaciona algumas taxonomias existentes até então, abordando tanto balanceamentos determinísticos quanto não determinísticos. O enfoque desse primeiro grupo é dado para os algoritmos não determinísticos. O segundo grupo é voltado para o balanceamento determinístico e é a base para a taxonomia proposta, porque, segundo os autores, as taxonomias do primeiro grupo apresentam limitações no que tange os determinísticos.

O “nível do balanceamento” distingue o balanceamento em intraprocessador e interprocessador. Esses termos correspondem, respectivamente aos balanceamentos local e global, destacados na taxonomia de Casavant; Kuhl [CAS88]. A “flexibilidade das normas” pode ser estática ou dinâmica e refere-se à flexibilidade inserida nas regras do balanceamento para reagir ao estado atual da plataforma computacional. Os termos estático e dinâmico são iguais aos usados por Casavant; Kuhl [CAS88].

4.5 Taxonomia de Luling; et al. (1993).

Lüling [LUL93] introduz uma nova taxonomia para algoritmos de balanceamento dinâmicos com o objetivo exclusivo de **balanceamento de carga**. Os termos estático, dinâmico, distribuído e adaptável são (apenas) utilizados de maneira análoga aos utilizados por Casavant; Kuhl [CAS88]. Para os autores, as taxonomias propostas até então, foram consideradas complexas para uma comparação genérica entre os algoritmos para balanceamento de carga proposta. Assim sendo, os algoritmos de balanceamento foram separados em duas partes: uma parte de decisão e outra de migração. Como o próprio nome diz, à parte de decisão é “responsável pela decisão de migrar a carga de trabalho”. À parte de migração é a responsável por enviar a carga para outro processador, **com o intuito de balancear a plataforma computacional usada**. À

parte de decisão pode ser baseada na situação local do processador e/ou no máximo dos seus vizinhos (decisão local) ou então depender de qualquer subconjunto dos processadores (decisão global). À parte de migração pode enviar cargas apenas para seus vizinhos diretos (migração local) ou então enviar essas cargas para quaisquer processadores da plataforma (migração global). Essa distinção entre local e global é à base da taxonomia proposta por Lüling [LUL93].

4.6 Taxonomia de Xu; Lau (1997).

Xu; Lau [LAU97] utilizam o termo balanceamento de carga como sinônimo de balanceamento de processos em todo o trabalho, portanto, esta taxonomia é apresentada exclusivamente para a função de balancear a carga da plataforma computacional, vide figura 4.2. Antes de apresentarem a taxonomia, os autores também citam que o balanceamento possui algumas "questões chave" (*key issues*). Dentre essas questões aparece à regra de iniciação, a qual pode ser iniciada pelo transmissor, pelo receptor, simetricamente ou ainda periodicamente. A iniciação periódica (ou remapeamento) induz todos os processadores a participarem de um rebalanceamento periódico com a intenção de atingir o objetivo do balanceamento. Para fundamentar a taxonomia, os autores definem o "domínio do balanceamento", o qual representa o conjunto de processadores envolvidos no balanceamento dinâmico. O domínio do balanceamento pode ser global ou local. O domínio global permite que todos os processadores pertencentes à plataforma façam parte do balanceamento. O domínio global ou local indica não somente quais são os possíveis parceiros para o balanceamento, mas também determina a quantia de informação trocada entre os processadores. Dessa forma, há duas questões envolvidas no domínio do balanceamento:

- As regras de localização de um parceiro para o balanceamento e as
- Regras para a troca de informações.

Algoritmos de balanceamento que trocam informações e procuram parceiros apenas no domínio local são conhecidos como algoritmos de balanceamento dinâmicos baseados nos vizinhos mais próximos. Os algoritmos baseados nos vizinhos mais próximos também são conhecidos como iterativos (repetitivos), porque transferem

processos de um processador para seus vizinhos e assim sucessivamente até atingir os objetivos propostos para o balanceamento. Cada salto feito pelos processos é determinado por uma decisão local, no processador transmissor dos processos. Assim, em um balanceamento iterativo o processador origem (aquele que estava inicialmente com os processos) não sabe qual será o processador destino dos processos, por não fazer essa atribuição (origem/destino) diretamente. Os algoritmos de balanceamento dinâmicos que localizam seus parceiros e que utilizam informações através do domínio global, são denominados algoritmos de balanceamento globais (ou diretos). O termo direto deve-se ao fato desses algoritmos determinarem o destino dos processos diretamente entre o processador transmissor e o receptor. Segundo os autores, os algoritmos globais são mais indicados em plataformas que possuem *broadcast* ou políticas de balanceamento centralizadas, em função do custo para determinar e para distribuir de maneira eficiente à carga de trabalho de todos os processadores. Já os algoritmos iterativos são mais flexíveis, onde a cada salto dos processos uma nova decisão sobre o balanceamento é tomada por um processador. Os algoritmos iterativos também são mais apropriados para plataformas que possuem a rede de comunicação direta (ou ponto-a-ponto) e com o roteamento do tipo *store-and-forward*, devido ao fato de cada processador receber a incumbência de executar processos e ainda assim poder enviá-los para outro processador. Os algoritmos determinísticos e estocásticos indicam quais serão as regras de distribuição aplicadas ao balanceamento. Os algoritmos de balanceamento iterativos determinísticos são caracterizados por possuírem regras predefinidas, as quais determinam para qual processador vizinho a carga de trabalho deve ser transferida, além de também determinar qual a quantia dessa carga deve ser transferida. Os algoritmos iterativos estocásticos têm por objetivo conduzir a plataforma utilizada a um estado que possua alta probabilidade de se atingir os objetivos propostos no caso o balanceamento de carga.

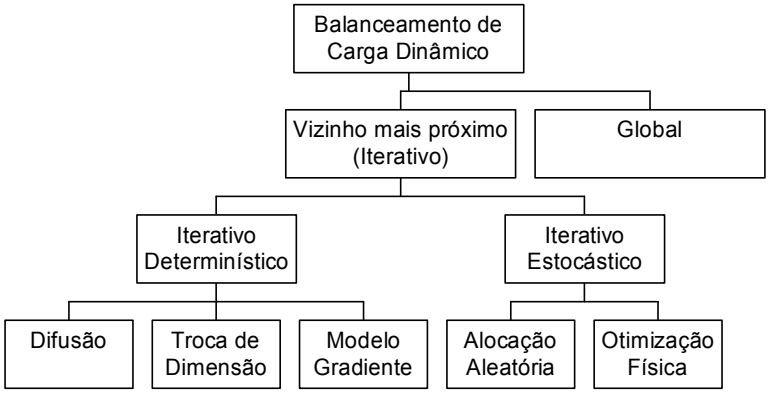


Figura 4.2 – Taxonomia de Xu; Lau [LAU95].

Os três tipos de algoritmos de balanceamento pertencentes ao balanceamento iterativo e determinístico são:

- Difusão (*diffusion*),
- Troca de dimensão (*dimension exchange*) e
- Modelo gradiente (*gradient model*).

Os algoritmos por difusão e por troca de dimensão são parecidos, em função de examinarem todos os seus vizinhos em toda operação de balanceamento. Porém, um algoritmo por difusão permite que, em cada operação de balanceamento, um processador contate todos os seus vizinhos ao mesmo tempo, propagando porções da sua carga de trabalho para um ou mais processadores, enquanto solicita carga de trabalho de outros vizinhos. Já com o algoritmo troca de dimensão, um processador também realiza o balanceamento com seus vizinhos e entra em contato com apenas um processador vizinho por vez. Depois de uma operação de transferência e/ou recebimento de carga de trabalho com um vizinho, o processador comunica a sua carga atual para o próximo vizinho e então uma nova distribuição de carga é iniciada. Maiores detalhes sobre a política de balanceamento baseada em difusão podem ser encontrados no trabalho de Corradi [COR97]. Os algoritmos de balanceamento baseados no modelo gradiente, propostos inicialmente por Lin; Keller [LIN87], restringem o balanceamento

apenas ao vizinho que apresenta a menor carga de trabalho. Segundo Lüling [LUL93] gradientes são vetores que possuem a menor distância (em saltos) entre o processador atual e um processador ocioso (com carga de trabalho baixa ou nenhuma), permitindo assim, que os processos a serem balanceados saibam a distância e saltem sobre os processadores até encontrarem esse processador ocioso.

4.7 Taxonomia de Quinn (1994)

O livro de Quinn [QUI94] destaca três grupos de balanceamentos para a computação paralela:

- Mapeamento em multicomputadores e em *arrays* de processadores,
- Balanceamento de carga dinâmico em multicomputadores e
- Balanceamento estático em multiprocessadores com acesso uniforme à memória.

O primeiro grupo, mapeamento em multicomputadores e *arrays* de processadores, refere-se ao problema de realizar o mapeamento prévio de uma aplicação paralela (representada por intermédio de um grafo) especificamente sobre uma plataforma computacional distribuída e com acesso à memória não uniforme. O acesso não uniforme à memória (NUMA – *Nonuniform Memory Access*) indica que cada processador é capaz de obter dados locais muito mais rápido que se estivesse acessando dados vindo de uma memória remota (em outro processador). Assim, com informações detalhadas sobre o hardware e sobre a aplicação paralela, as decisões sobre o balanceamento levam em consideração principalmente fatores como a localização dos dados, de modo que seja otimizado o acesso aos mesmos pelos processos balanceados.

O segundo grupo, balanceamento de carga dinâmico em multicomputadores, refere-se "à atividade de mudar a distribuição do trabalho entre os processadores durante a execução". Essa definição indica que esse grupo utiliza a migração de processos para mudar o balanceamento durante a execução, o que segundo a taxonomia de Casavant, Kuhl [CAS88] refere-se ao balanceamento dinâmico e com preempção (com migração). Nesse segundo grupo Quinn [QUI94] classifica os algoritmos em:

- Centralizados,
- Completamente distribuídos,

- Semidistribuídos.

Os algoritmos centralizados foram definidos de maneira análoga à definição apresentada em Casavant; Kuhl [CAS88]. Os algoritmos completamente distribuídos permitem que cada processador possua a sua própria imagem do estado da carga da plataforma, onde os processadores trocam informações e processos (carga de trabalho) com seus vizinhos mais próximos. Para a taxonomia de Xu; Lau [LAU97], apresentada anteriormente, esses algoritmos que trocam informações com seus vizinhos são chamados de iterativos. Os algoritmos semidistribuídos dividem os processadores em grupos, onde cada grupo possui um coordenador que trata o balanceamento de maneira centralizada. O terceiro grupo de balanceamento é chamado de estático em multiprocessadores e com acesso uniforme à memória (UMA – U*niform Memory Access*). Esses são os balanceamentos realizados antes da execução e em plataformas com memória compartilhada, onde todos os processadores têm acesso físico e lógico a mesma porção de memória principal. Como a velocidade de acesso à memória é considerada uniforme para todos os processadores, a discussão desses algoritmos enfoca principalmente o tempo de execução dos processos escalonados e não a distribuição dos dados entre os processos, como foi o caso do primeiro grupo de balanceamentos.

4.8 Taxonomia usada por Tanenbaum (1995)

A primeira classificação citada refere-se aos algoritmos migratórios e não migratórios, indicando respectivamente os algoritmos que realizam migração de processos (preemptivos) e aqueles que não a realizam (não preemptivos). O segundo grupo de taxonomias, as quais são apresentadas como "questões de projeto" de algoritmos de balanceamento, são: determinísticos x heurísticos, centralizados x distribuídos, ótimos x subótimos, locais x globais e iniciados pelo transmissor x iniciados pelo transmissor. Os algoritmos determinísticos indicam o balanceamento que possui conhecimento total sobre o comportamento de todos os processos a serem escalona dos e sobre a plataforma como um todo. Os algoritmos determinísticos são aqueles que podem teoricamente analisar todas as possíveis variações para o balanceamento e escolher a melhor. Os algoritmos de balanceamento heurísticos, por

outro lado, são os que estão preparados para variações arbitrárias e significativas na plataforma e nos processos em tempo de execução.

.

Capítulo 5

REFLEXÃO COMPUTACIONAL

O objetivo deste capítulo é apresentar os principais conceitos de reflexão computacional, destacando aqueles que são utilizados nesta dissertação.

5.1 Introdução

Intuitivamente, os sistemas de reflexão computacional permitem que as computações observem e modifiquem as propriedades de seu próprio comportamento. Por analogia, o conceito de reflexão talvez seja mais bem explicado pelo estudo de autoconsciência (self-awareness) da Inteligência Artificial: "Aqui estou andando rua abaixo na chuva. Uma vez que eu estou ficando ensopado, devo abrir meu guarda-chuva". Esse fragmento de pensamento é um exemplo que revela uma autoconsciência de comportamento e estado, que por sua vez leva à alteração nos mesmos comportamento e estado [SOB96]. Mais especificamente, a reflexão computacional é o processo em que um sistema pode analisar seu próprio comportamento e atuar sobre o mesmo. Em um sistema convencional, a computação é realizada em dados que representam entidades externas ao sistema. No entanto, um sistema reflexivo deve conter dados que representam aspectos estruturais e computacionais do próprio sistema. Também deve ser possível acessar e manipular tais dados a partir do sistema e, principalmente, tais dados devem estar integrados ao comportamento do sistema, ou seja, alterações nesses dados devem causar alterações no comportamento do sistema e vice-versa [FAB95].

5.2 Protocolos de Meta-Objetos

O objetivo no desenvolvimento do protocolo de meta-objeto (Meta-Objeto Protocolo) foi o de permitir que uma linguagem fosse extensível o suficiente para admitir a inclusão de novas características, mantendo ao mesmo tempo sua simplicidade, poder de programação, compatibilidade e desempenho com versões anteriores. Assim, o Protocolo de Meta-Objeto permite a extensão de uma linguagem, que então passa a abrir sua abstração e implementação à intervenção do programador. Para isso, são usadas as técnicas de orientação a objetos e de reflexão computacional, que organizam uma arquitetura de nível meta, onde é possível adicionar as características que estendem a linguagem de programação [KIC93]. Por conseguinte, quando a reflexão computacional é aplicada à programação orientada a objetos, tem-se a abordagem de meta-objetos. Ela estrutura os objetos em dois níveis: (1) nível base

(objeto-base) e (2) nível meta (meta-objeto) [LAU96]. Cada objeto-base x está associado com um meta-objetox, que representa tanto aspectos estruturais quanto computacionais de x, aspecto este que pode ser gerenciado dinamicamente através de computações realizadas em x Figura 5.1. As chamadas aos métodos do objeto-base são desviadas com o propósito de ativar meta-métodos que permitem a modificação do comportamento do objeto-base ou a adição de funcionalidades a seus métodos. Essa abordagem possibilita a separação dos aspectos funcionais e não funcionais de uma aplicação, permitindo que os métodos e procedimentos da aplicação em si sejam tratados no nível base, enquanto o controle e o gerenciamento da aplicação são tratados no nível meta. Dessa forma, a reflexão torna possível abrir a implementação de um sistema sem revelar detalhes desnecessários de sua implementação, fornecendo flexibilidade de implementação [FAB95].

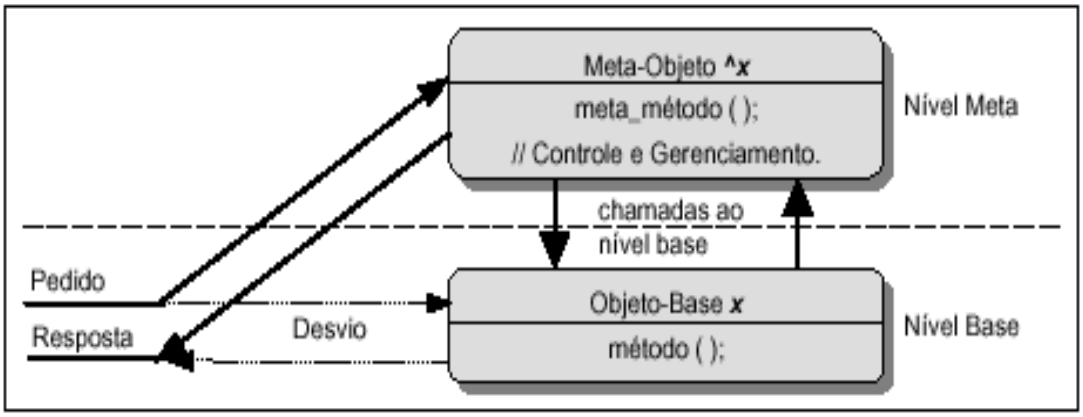


Figura 5.1 – Reflexão Computacional

No processo de Reflexão Computacional, basicamente possuímos três estágios de realização da Reflexão Computacional, como segue:

- O primeiro estágio, conhecido como reificação (traduzido do termo *reification*, utilizado na literatura sobre computação reflexiva), consiste em obter uma descrição abstrata do sistema tornando-a suficientemente concreta para permitir operações sobre ela.

- No segundo estágio, a reflexão computacional, utiliza esta descrição concreta para realizar alguma manipulação.

- Finalmente, no terceiro estágio, é modificada a descrição reificada com os resultados da reflexão computacional, retornando a descrição modificada ao sistema. Desta forma, as operações subseqüentes refletirão as alterações efetuadas sobre a descrição reificada do sistema [ZAN97].

5.3 Reificação

A reificação (reification), ou materialização, é o ato de converter algo que estava previamente implícito, ou não expresso, em algo explicitamente formulado, que é então disponibilizado para manipulação conceitual, lógica ou computacional. Portanto, é através do processo de reificação que o nível meta obtém as informações estruturais internas dos objetos do nível base, tais como métodos e atributos. Contudo, o comportamento do nível base, dado pelas interações entre objetos, não pode ser completamente modelado apenas pela reificação estrutural dos objetos-base. Para tanto, é preciso intermediar as operações de nível base e transformá-las em informações passíveis de ser manipulada (analisadas e possivelmente alteradas) pelo nível meta [OLI98].

5.4 Tipos de Reflexão

A reflexão computacional pode ser particionada genericamente em dois tipos de funções [TAT99] [GOL98]: a introspecção e a intercessão, descritas nas subseções a seguir. O programa que realiza a reflexão computacional, introspecção e / ou intercessão, é chamado de programa de nível meta, enquanto o programa executado na computação reflexiva é chamado de programa de nível base [TAT99].

- **Introspecção:**

A introspecção (introspection), também referenciada como reflexão estrutural (structural reflection), refere-se ao processo de obter informação estrutural do programa e usá-la no próprio programa. Isso é possível através da representação, em nível meta,

dessa informação do programa, feita por um processo de reificação.

- **Intercessão:**

A intercessão (intercession), também referenciada como reflexão comportamental (behavioral reflection), ou ainda como interceptação, refere-se ao processo de alterar o comportamento do programa no próprio programa. Isso pode ser realizado, por exemplo, através do ato de interceder, intermediar, ou ainda interceptar [FAB95] operações de nível base – como operações de invocação de método, ou operações de estabelecimento *(setting)* / obtenção *(getting)* de valores de atributos –, que podem assim ser reificadas e, conseqüentemente, manipuladas pelo nível meta [OLI98].

Capítulo 6

SISTEMA OPERACIONAL AURORA

O Sistema Operacional Aurora a seguir mostrado, foi criado no trabalho de Doutorado do Prof. Dr. Luiz Carlos Zancanella, o qual esta dissertação esta inserida mostrando uma solução de Balanceamento de Carga.

6.1 Introdução

Pressmam comenta sobre o impacto que a Programação Orientada a Objetos, exercerá sobre futuras arquiteturas computacionais baseada nessa tecnologia que tem tomado força a partir de meados de 80. Alguns autores prevêem sistemas operacionais específicos a esta finalidade, e por causa disso tem-se despontado muito interesse na comunidade acadêmica à Computação Orientada a Objetos com a utilização de Reflexão Computacional.

O paradigma de orientação a objetos na engenharia de software tem demonstrado a grande importância de sistemas orientados a objetos. Na década de 90 foi marcada com pesquisas e muitos projetos em andamento na busca de soluções a esta nova tecnologia, neste período também, começaram a aparecer pesquisas associando a Orientação a Objetos com a Reflexão Computacional que estão em fase bem adiantada, pode-se citar como exemplo os sistemas operacionais (Apertos, 1991, CHOICES, 1992, Deubbler, Koestler, 1994, e o próprio AURORA, 1997) [ZAN97].

6.2 Modelo de Estrutura Reflexiva do Aurora

O modelo de plataforma reflexivo orientado a objeto, multiprocessado, paralelo e utilizável em ambiente distribuído foi proposto juntamente com AURORA por [ZAN97].

Este modelo de estrutura reflexiva vem de encontro à necessidade de criar um ambiente computacional direcionada a orientação a objetos, e é de se observar que os sistemas operacionais existentes não foram planejados para suportar as linguagens modernas.

No modelo computacional reflexivo Aurora é caracterizado por objetos, meta-objetos e meta-espaços, onde, objetos são instâncias de uma classe, que contém todas as informações e características dos dados, enquanto que Meta-objetos refletem o comportamento desses objetos. Cada objeto possui seu Meta-Espaço, que é a composição de mais meta-objetos do objeto. [ZAN97]

6.3 Arquitetura do Sistema Operacional Aurora

A arquitetura resultante da utilização do modelo de estrutura reflexiva sobre Aurora pode ser vista como definida dentro de uma hierarquia de meta-espaços, onde no topo da hierarquia, está implementado o suporte ao modelo estrutural, a partir do qual, o sistema operacional utilitário e aplicações do usuário são construídos. As implementações deste suporte são realizadas através de meta-espaços que implementam basicamente mecanismos para: gerenciamentos de meta-espaços, suporte a ativações e multiprocessamento [ZAN97]. A figura 6.1 apresenta uma visão simplificada deste modelo onde o suporte ao modelo estrutural constitui o topo da hierarquia. Deve-se observar que alguma meta-espaço não possui objetos associados. Esta particularidade ocorre principalmente nos meta-espaços que implementam basicamente serviços do sistema operacional.

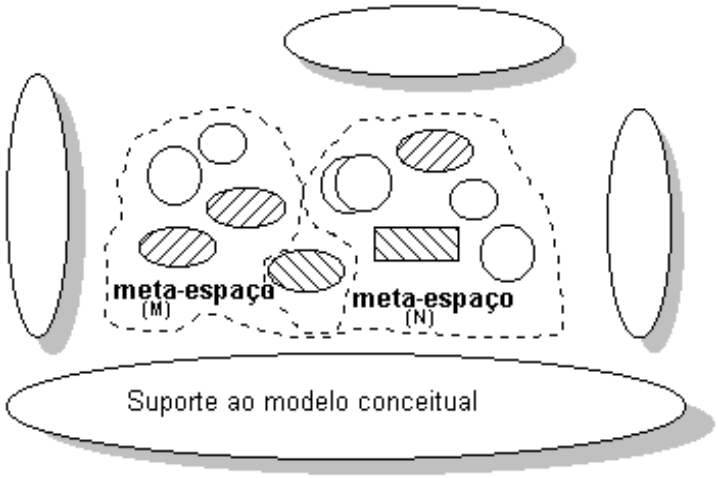


Figura 6.1 Visão simplificada da arquitetura de Aurora

Capítulo 7

BALANCEAMENTO DE CARGA NO SISTEMA OPERACIONAL AURORA

Diversas técnicas de balanceamento de carga têm sido propostas, a fim de obter melhoria no desempenho de aplicações paralelas e distribuídas. Nesse trabalho, será mostrada uma solução de Balanceamento de Carga para o Sistema Operacional Aurora, como segue.

7.1 Introdução

O modelo de Balanceamento de Carga a seguir proposto, foi constituído para ser implementado no Sistema Operacional Reflexivo Aurora. Sistema este que trabalha diretamente com Reflexão Computacional e define Objetos puros, no Sistema Operacional Aurora, segundo Zancanella [ZAN97], existem dois problemas principais inerentes ao Balanceamento de Carga, como segue: "*Administrar o uso dos processadores é uma função muito importante do sistema. O objetivo básico é maximizar a taxa de throughput (quantidade de processamento) do sistema, minimizando o tempo que um objeto tem que esperar para ser alocado a um processador. A tarefa de atribuir objetos a processadores é dificultada por* ***dois objetivos conflitantes****:* ***primeiro*** *objetos devem ser atribuídos a diferentes processadores, levemente carregados, de modo a executarem concorrentemente;* ***segundo*** *objetos que interagem freqüentemente devem ser atribuídos ao mesmo ou a processadores próximos de modo a reduzir o custo de comunicação. Assim, o benefício de executar objetos de um programa em múltiplos processadores é contrabalanceado pelo custo adicional da comunicação". Para obter um desempenho ótimo, os objetos, de um programa baseado em objetos, devem ser atribuídos a um grupo de processadores levemente carregados e próximos.[ZAN97].* A solução para estes problemas foi encontrada através do Balanceamento de Carga distribuído Global.

7.2 Políticas de Funcionamento

Através da análise das diferentes Taxonomias existentes de Balanceamento de Carga em Sistemas Operacionais, e levando em consideração alguns princípios que nortearam este trabalho, a seguir, detalharemos as características básicas do Modelo de Balanceamento de Carga do Aurora criado neste trabalho.

O Balanceamento de Carga do Aurora funcionará em um ambiente com o uso de plataformas homogêneas, com execução de processos independentes e seqüenciais.

O Balanceamento de Carga do Aurora a partir de agora denominado simplesmente de BCA, será **não-preemptivo,** ou seja, a preempção é uma técnica

empregada no Balanceamento de Carga, onde um processo pode ser suspenso em um processador e colocado em execução em outro processador. O BCA foi construído sem a característica de migração de processo, o processo uma vez iniciado em um "nó" da Rede processará até o seu término. **Vantagem**: Esta técnica tem a vantagem de não haver desperdício de tempo entre a comunicação entre os nós entre a migração dos processos. **Desvantagem**: Sem a possibilidade do processo não ser migrado em tempo de execução, isto gera a desvantagem de que em certos períodos na rede de processadores haja ociosidade em processos com possuem um tempo de atividade muito longo.

Um Balanceamento "Ótimo" pode ser realizado quando se possui toda a informação necessária a respeito dos processos que serão balanceados, assim como da plataforma computacional utilizada. Entretanto, a geração de uma solução ótima é, na maioria dos casos, um problema NP-completo, inviabilizando, portanto, a sua realização. Os balanceamentos "ótimos" são possíveis apenas em casos específicos, como por exemplo, quando o tempo de execução de todos os processos é o mesmo e apenas dois processadores são utilizados. Uma alternativa para o problema NP-completo das soluções ótimas são as soluções "**Sub-ótimas**", as quais não se prende apenas à melhor solução, ficando satisfeitas se "um bom balanceamento" foi realizado.

O BCA será **distribuído** (Descentralizado), ou seja, quando a responsabilidade sobre o balanceamento será compartilhada entre os diferentes processadores. A plataforma do Sistema Operacional Aurora possuirá a autoridade de balanceamento de carga descentralizada, pois será possível distribuir a responsabilidade para coletar informações e efetuar a decisão de balanceamento em todos os "nós" do sistema. Os Objetos que conterão e gerenciarão estas informações, estarão baseados nos vários Objetos "Activity" instanciados nos vários "nós" da rede. **Vantagem**: O sistema não dependerá de um "nó" centralizador ou de uma parte dos "nós" (solução global parcial) para a solução do balanceamento, com isto aumentará muito a disponibilidade efetividade da solução, gerando um índice muito bom quanto à tolerância à falhas. **Desvantagem**: O sistema perderá em agilidade para definir a solução de balanceamento, pois deverá percorrer todos os "nós" da rede coletando informações para a sua

avaliação, para depois definir a melhor solução.

O Balanceamento de Carga do Aurora – BCA, será **Iniciado pela Origem** (Source-Initiative), ou seja, quem terá a iniciativa de iniciar o balanceamento. Os processadores origem são aqueles que possuem processos para serem executados, enquanto os processadores servidores representam aqueles que desejam executar os processos remotos. Se um processador origem é quem toma a iniciativa de procurar onde executar um processo, essa política é classificada como **"Iniciada pela origem"**. Neste caso os processos na origem terão uma dependência por informação que a política de Balanceamento resolverá, a dependência causada será sanada através da utilização da estrutura criada dentro do trabalho do M. Sc. José Rodrigo Balzan que foi "Uma Solução Reflexiva para Gerenciamento de Objetos Distribuídos em Aurora", neste trabalho desenvolveu-se uma estrutura chamada AVLO – Access Virtual List Object, nesta estrutura o Sistema Operacional Aurora possuirá as informações necessárias ao Balanceamento de Carga, como detalharemos a frente. **Vantagem**: A rede de servidores de processamento iniciará a execução do balanceamento no momento exato de uma solicitação de execução de uma atividade de processo de um determinado servidor, com isto, não ocorrerá desperdício de tempo e recurso desnecessários de verificações constantes de "leveza" da rede. **Desvantagem**: Se o sistema possuísse as informações previamente da rede, antes da solicitação de criação de processo de um determinado servidor, não haveria desperdício de tempo para se obter essas informações no momento da solicitação.

Utilizaremos no BCA um tipo de Balanceamento comumente utilizado em sistemas de computação, o Balanceamento **Não Determinístico** em função de raramente haver uma informação exata sobre os processos, plataforma e objetivos que devem ser considerados.

O Balanceamento de Carga do Aurora – BCA, possuirá a característica de envio dos processos para criação das "Activity" para qualquer "nó" da Rede ou processador que esteja mais leve, isto lhe dá a funcionalidade de Balanceamento **global**, ou seja, pode enviar cargas para quaisquer processadores da plataforma. **Vantagem**: A solução desenvolvida analisará todos os processadores da rede, com isso a solução

gerará equidade para todos, ou seja, todos possuirão a mesma responsabilidade pelo balanceamento, o sistema ficará totalmente independente de qualquer processador para as tomadas de decisões, ocasionando com isso maior disponibilidade do sistema. **Desvantagem**: Sempre que houver uma solicitação de balanceamento, o sistema terá que analisar todos os processadores da rede, sem levar em consideração à proximidade do nó que estiver mais leve, isto ocasionará a utilização desnecessária de recursos de comunicação entre os nós.

7.3 Especificação da Solução

1. Descrição

Esta solução tem por objetivo criar processos em um "nó" da Rede, utilizando técnicas de Balanceamento de Carga, ou seja, criação de atividades de processos Balanceados. Estes processos serão denominados de "Activity".

2. Atores

Sistema Operacional Aurora – Ator que dá início a Activity definido pela solução. Esta solução representa um processo condicionada a requisição recebida pelo BCA do Sistema Operacional Aurora.

3. Solução

3.1 Objetos atuantes no Processo de Balanceamento de Carga

Seguem os objetos pertencentes ao BCA e suas funções básicas, estes objetos serão instanciados em todos os servidores ("nó") da Rede, ou seja, existem quantos objetos e instâncias quanto o número de servidores ("nó").

Objetos:

- "metaActivity" – Objeto que trabalha no meta espaço do sistema operacional, responsável por capturar as mensagens e exercer

computações sobre estas. Ao interceptar uma mensagem de criação de activity, esta será tratada conforme descrito nos métodos a seguir listados do objeto.

- "verificaBalanceamento" – atividade de verificação de existência de processos em execução no servidor de origem da solicitação de criação de activity, ou seja, servidor ("nó") de origem.

- "realizaBalanceamento" – atividade de balanceamento no servidor indicado, ou seja, verifica quantos processos estão em atividade no servidor, através de verificação do Objeto AVLO – Access Virtual List Object. Este método possui a seguinte interface:
  - Identificação do servidor que possui a activity a ser instaciada, ou seja, servidor que originou o balanceamento de carga;
  - Número de processo do servidor mais leve;
  - Identificação do servidor mais leve.

- "AVLO" – Access Virtual List Object – objeto do sistema operacional que efetua o Gerenciamento de objetos no sistema, dentre as varias informações que o objeto possui, utilizaremos as seguintes:
  - Identificação do processo em atividade,
  - Servidor de origem do processo em atividade,
  - Servidor de destino que o processo foi instanciado.

  Através destas informações, conseguiremos identificar o número de processos em execução no servidor em Balanceamento.

  - "Activity" Objeto responsável por fazer a instanciação da activity no servidor "leve", indicado pelo metaActivity

3.2 Fluxo da Solução

Identifica o melhor caminho para conclusão da solução, ou seja, identifica o "caminho feliz" para realização da tarefa.

3.1.1 Início do processamento – Criação de Activity.

O processamento inicia quando o Sistema Operacional Aurora – SOA solicita a criação de uma Activity. O SOA deverá acionar a execução da rotina toda vez que houver a submissão de novos processos para o Sistema, poderá acontecer em intervalos irregulares.

3.1.2 Obtenção da Solicitação de Criação de Activity

O Sistema, através do Meta Objeto "metaActivity" captura a mensagem de criação de Activity com as seguintes informações: Solicitação de criação de Activity e a Activity a ser criada (Objeto a ser instanciado);

3.1.3 Verificação se o servidor solicitado para hospedar a Activity possui processo em execução.

Após capturar a mensagem de criação de Activity, o BCA, através do "metaActivity" verificará através do método "verificaBalanceamento" se no servidor solicitado possui processos em atividade (execução), ou seja, o servidor candidato para hospedar a Activity a ser instanciada, se caso negativo, isto é, o servidor candidato não possui processos instanciados, está vazio, então, o metaActivity enviará mensagem para o objeto "activity", para que este instancie o processo neste servidor.

Neste caso não há necessidade do balanceamento de carga na Rede, pois, o servidor candidato esta completamente "vazio".

3.1.4 Realização Balanceamento Global na Rede, método "realizaBalanceamento".

Na hipótese do servidor candidato possuir atividades em execução, então o BCA verificará o quanto este servidor esta "leve" (qual o número de processos em

execução), e assim sucessivamente até a conclusão do balanceamento global na Rede.

A chamada inicial para realizar o balanceamento será pelo método "verificaBalanceamento" do objeto metaActivity daquele servidor, este método verifica se há processos em execução. Se não possuir processos em execução, como dito anteriormente, então este é o servidor indicado para o processo. Com isto, o "metaActivity" informará ao objeto "activity" que o servidor que receberá o processo será este (sem nenhum processo, totalmente vazio). Se ao contrário, o servidor possuir processos, então o "metaActivity" irá chamar o método "realizaBalanceamento". Este método realiza o balanceamento através do acesso ao objeto AVLO daquele servidor. No AVLO podemos identificar o número de processos em execução do servidor naquele momento.

Uma vez identificado o número de processos no servidor, o "metaActivity" comparará quais dos servidores esta mais leve, se aquele anteriormente medido ou o que acabou de ser. A mensagem que trafega entre os servidores durante o processo de realização de balanceamento de carga Global, será atualizada com a indicação do servidor mais leve, esta mensagem possui a seguinte interface:

- Activity a ser instânciada,
- Identificação do servidor de origem ("nó") mais leve,
- Identificação do servidor de destino da mensagem,
- Número de processos do servidor mais leve,
- Identificação do servidor que deu início ao balanceamento.

3.1.5 Criar activity no servidor mais leve da Rede.

Ao interceptar a mensagem para verificação de carga de dado servidor, o objeto "metaActivity" verifica a origem do servidor que solicitou o balanceamento, se coincidir com o servidor hora em análise. Então, haverá uma última medição através do método "realizaBalanceamento". Se a sua carga for menor que o servidor mais leve da Rede, então haverá a instanciação da activity localmente. Se não, o metaActivity enviará uma mensagem para criação da activity no servidor mais leve da Rede, através

do objeto “Activity”. Em seguida atualizará a AVLO do servidor de origem e o AVLO do servidor de destino, informando para estes o número da activity instanciada, identificação do servidor de origem da activity e a identificação do servidor de destino onde foi instanciado a activity.

3.3 Diagramas da Solução

3.3.1 Diagrama de Classe da Solução

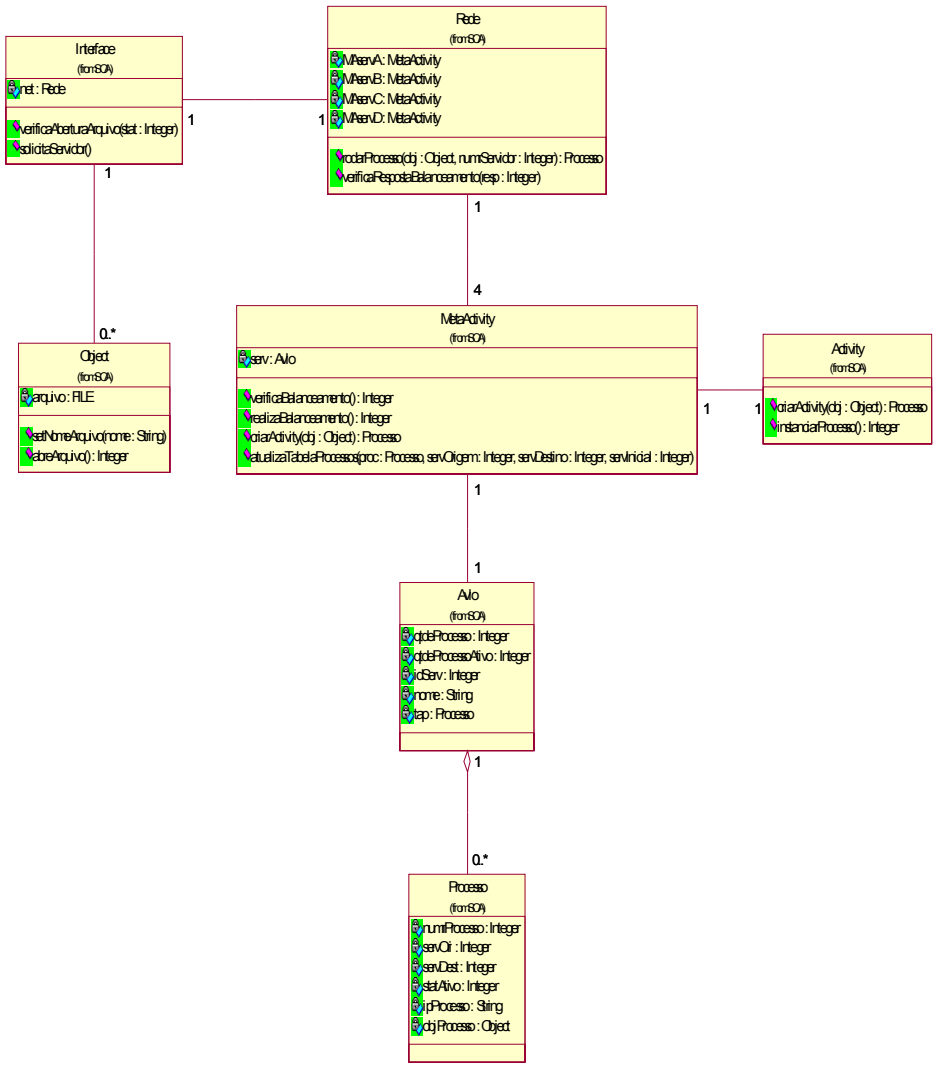


Figura 7.1 Diagrama de Classe da Solução

3.3.2 Diagrama de Seqüência: Realiza Balanceamento

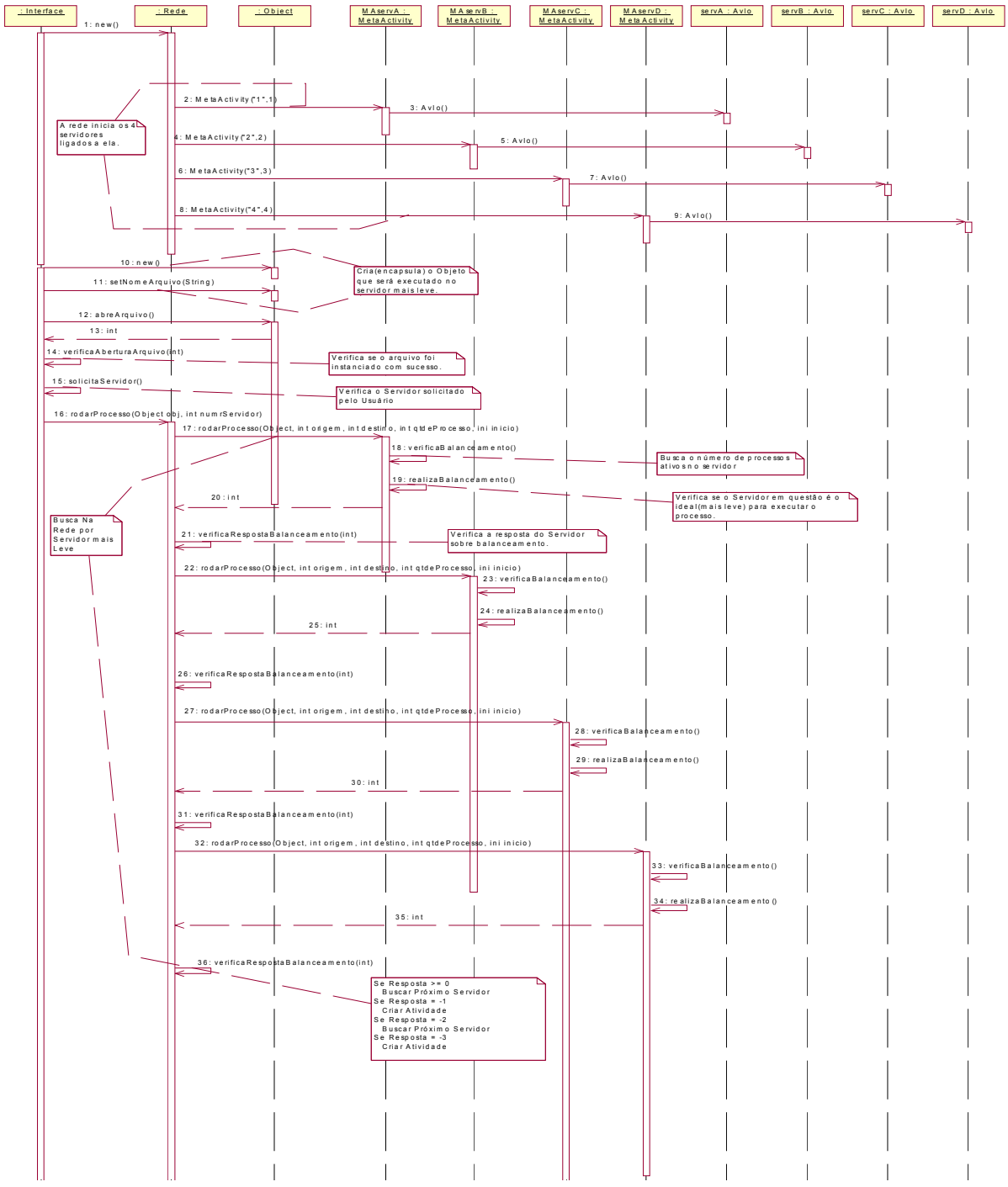


Figura 7.2 Diagrama de Seqüência – Realiza Balanceamento

3.3.3 Diagrama de Seqüência: Criar Activity

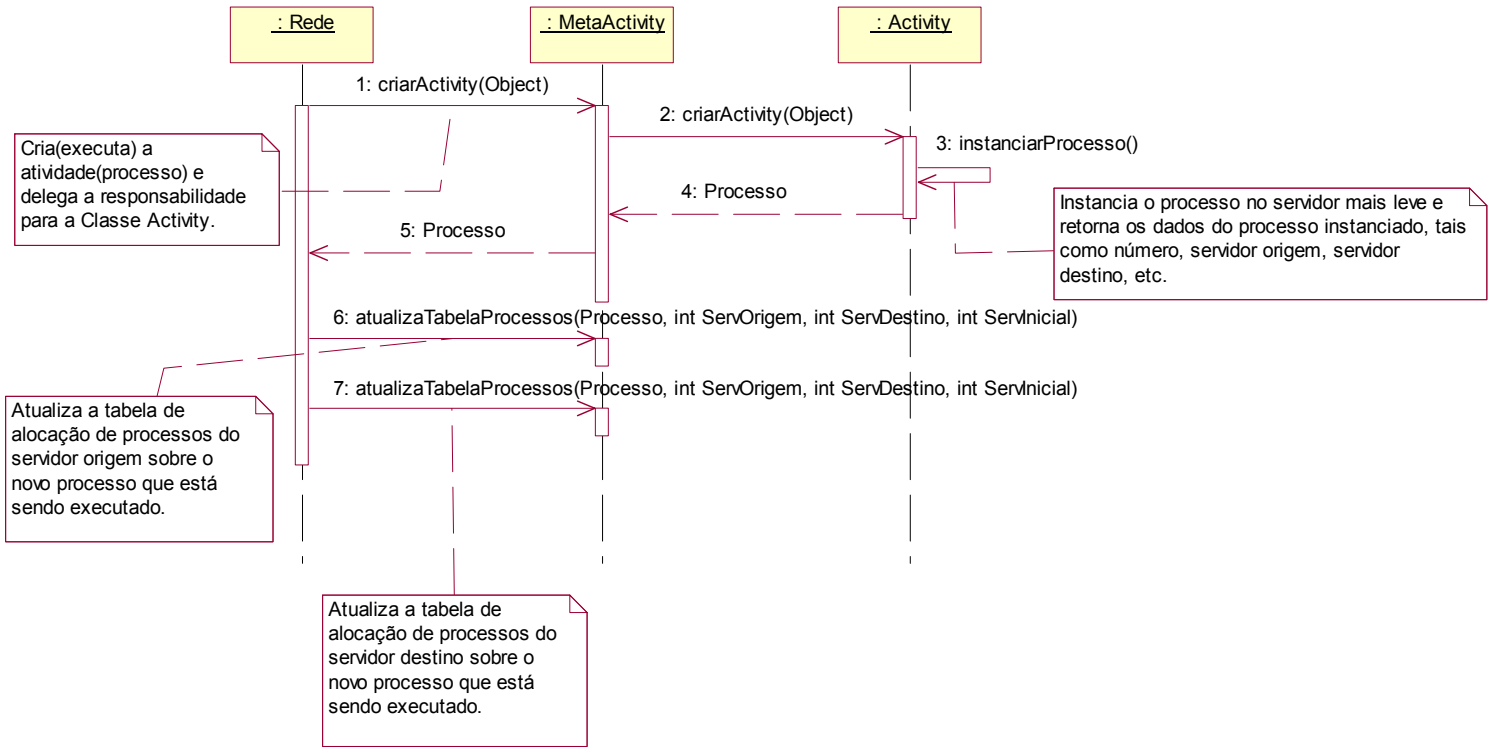


Figura 7.3 Diagrama de Seqüência – Criar Activity

3.3.4 Diagrama de Colaboração: Realiza Balanceamento

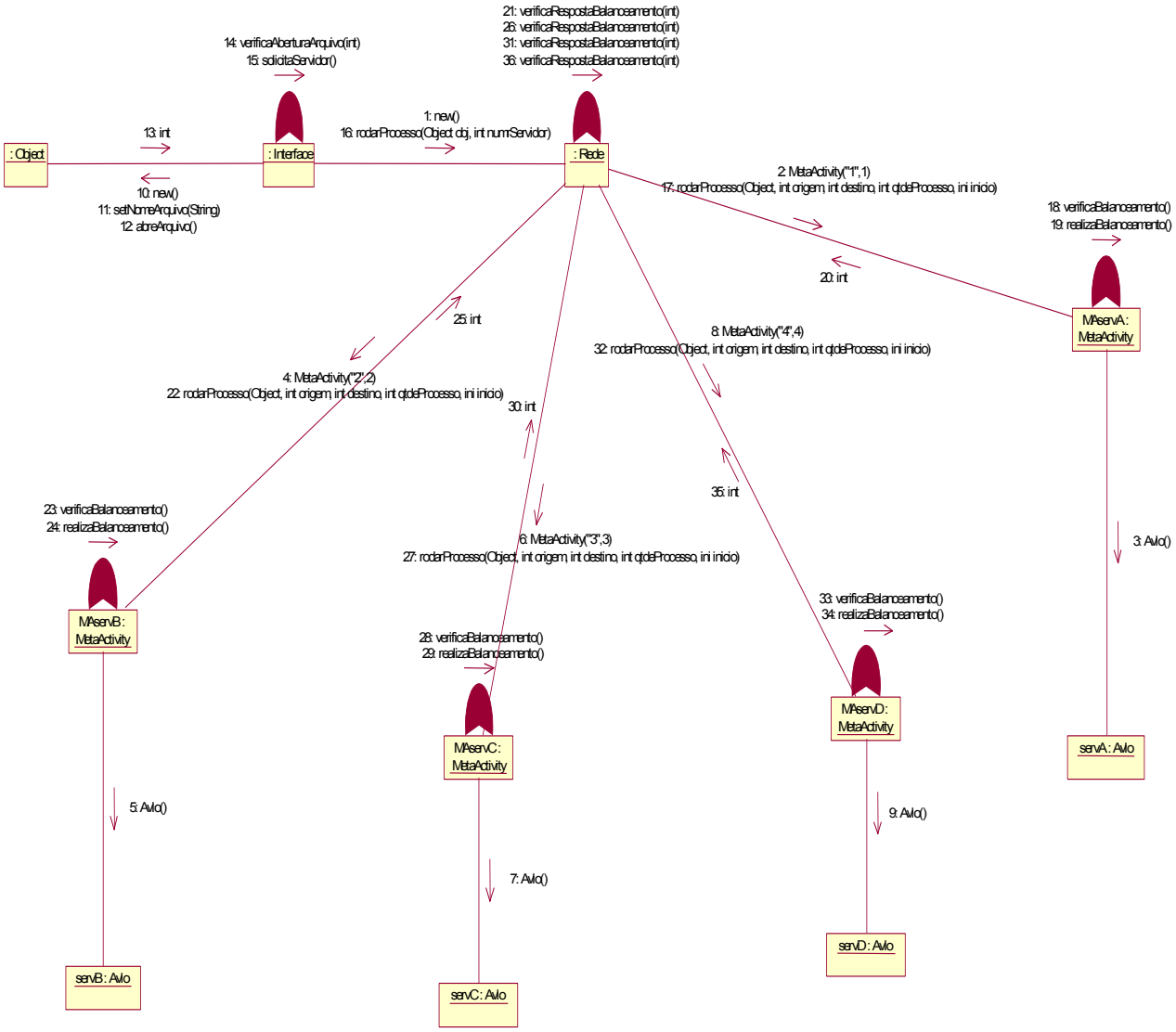


Figura 7.4 Diagrama de Colaboração – Realiza Balanceamento

3.3.4 Diagrama de Colaboração: Criar Activity

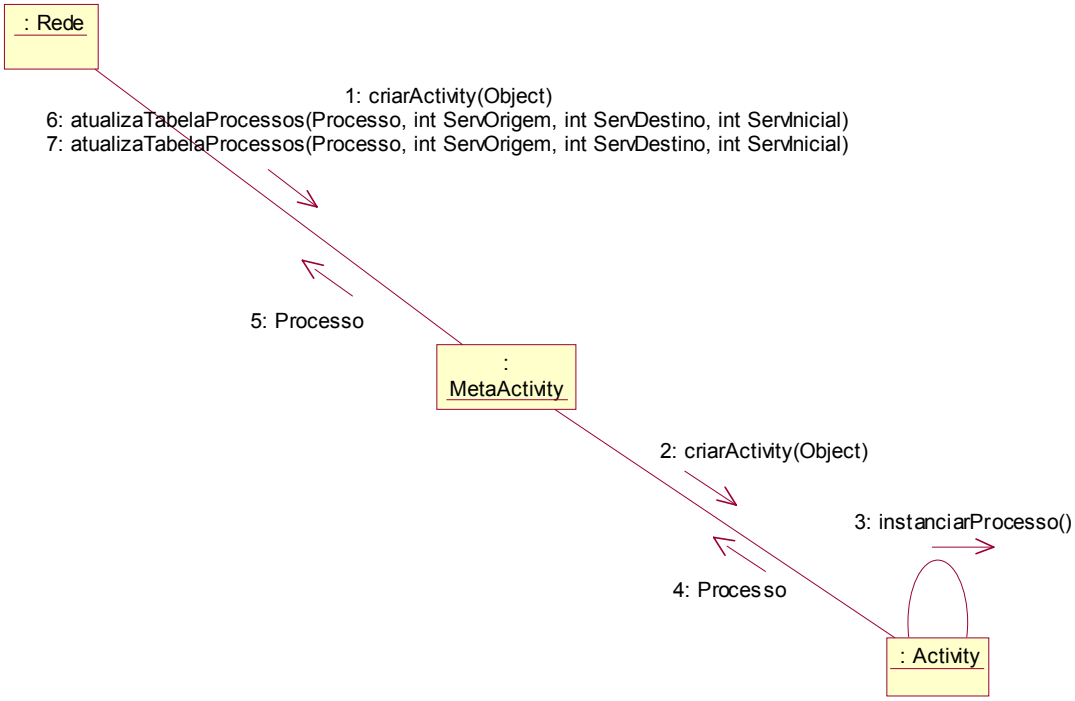


Figura 7.5 Diagrama de Colaboração – Criar Activity

Capítulo 8

CONCLUSÃO

Nesse trabalho abordou-se uma solução para o balanceamento de carga global do sistema operacional Aurora. Segue a conclusão desta técnica em seu contexto.

Esta dissertação apresentou uma solução para o problema de balanceamento de carga do sistema Aurora, esta solução foi constituída através da análise de diferentes taxonomias de Balanceamento existentes, suas políticas de funcionamento, suas características, suas vantagens e desvantagens.

O modelo proposto se encaixa dentro da arquitetura do sistema Aurora, ou seja, sistema operacional distribuído, orientado a objetos com reflexão computacional; O Balanceamento de Carga do Aurora – BCA, é distribuído, fazendo a verificação de carga de processamento globalmente, pois, a faz em todos os processadores da Rede; O BCA é orientado a objetos, pois possui uma arquitetura definida com objetos, o qual foi implementado com linguagem de programação nesta arquitetura – "C++"; Possui reflexão computacional através de seu meta objeto "criarActivity" – faz computações sobre si mesmo.

O BCA trata a verificação de carga nos processadores através da quantidade de processos em execução no mesmo. O BCA faz incursão em todos os processadores da Rede para verificação do mais leve, esta verificação se dá através da verificação de existência de processos e da sua quantidade. Para melhorarmos o modelo, o BCA poderia verificar a proximidade do "nó" servidor de destino em relação à origem, ou seja, evitarmos o tempo gasto de transferência do processo do "nó" origem para o "nó" destino.

Dentre os processos que envolvem a execução do Aurora, o BCA vem a ser um dos requisitos importantes para atender sua funcionalidade e flexibilidade. Um protótipo da implementação do BCA foi construído, o qual atingiu as metas propostas no trabalho.

O modelo proposto torna a exploração de ambientes distribuídos mais dinâmico, pois não fica a cargo somente de uma máquina concentrar as informações do Balanceamento da Rede, e sim, de todas que participam do ambiente.

9 REFERENCIAS BIBLIOGRÁFICA

[AND95] ANDERSON, T.E.; CULLER, D.E.; PATTERSON, D.A. A Case for Now (Networks of Workstations). IEEE Micro, p.54-64, fevereiro, 1995.

[BAL01] BALZAN, JOSÈ RODRIGO, Uma Solução Reflexiva para Gerenciamento de Objetos Distribuídos em Aurora, Dissertação de Mestrado, Florianópolis, 2001, UFSC.

[BAU91] BAUMGARTNER, K.M.; WAH, B. W. Computer Scheduling Algorithms: Past, Present and Future. Information Sciences, Elsevier Science Pub. Co., Inc., New York, NY, v.57 & 58, p.319-345, setembro/dezembro, 1991.

[BES97] BESTAVROS, A. Load Profiling in distributed real-time systems. Information Sciences, v.101, Iss 1-2, p.1-27, 1997.

[CAS88] CASAVANT; T.L.; KUHL, J.G. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. IEEE Transactions on Software Engineering, p.141-154, fevereiro, 1988.

[CHA95] CHANGCHIEN S.W. et alli, "A Dynamic Control Model of Flexible Manufacturing Cells Using the Information Processing Object Hierarchy".International Journal FAIM, Vol. To appear, No., 1995.

[COR97] CORRADI, A.; LEONARDI, L.; ZAMBONELLI, F. Performance Comparison of Load Balancing Policies based on a Diffusion Scheme. In: Euro-Par' 97 Parallel Processing - Third International Euro-Par Conference, Lecture Notes in Computer Science v.1300, Passau, Germany, agosto, 1997.

[COU94] COULORIS, G.; DOLLIMORE, J.; KINDBERG T. Distributed Systems: Concepts and Design, Addison-Wesley, 1994.

[DOU96] DOU, J-M; French Small Business Information Through the Internet: A Comparison with US Organizations, *International Journal of Information Management*, Vol. 16, No. 4, pp. 289-298, 1996.

[ESK89] ESKICIOGLU, M.R. Design Issues of Process Migration Facilities in Distributed Systems. IEEE Technical Committee on Operating System Newsletter, p.3-13, 1989.

[FAB95] FABRE J.; NICOMETTE, V. Implementing Fault Tolerant Applications Using Reflective Object-Oriented Programming. Proceedings of the 25 th IEEE International Symposium on Fault-Tolerant Computing, Pasadena, CA, EUA, Jun. 1995.

[FLY72] FLYNN, M.J. Some Computer Organizations and Their Effectiveness. IEEE Transactions on Computers, v.C-21, p.948-960, 1972.

[GEI94] GEIST, A.; BEGUELIN, A.; DONGARRA J.J.; JIANG, W; MANCHEK, R.; SUNDERAM, R. PVM 3 Users Guide and Reference Manual. Oak National Laboratory, setembro, 1994.

[GOL98] GOLM, MICHAEL; KLEINÖDER, JURGEN. meta and the Future of Refletion. OOPSLA'98 Workshop on Reflective Programming in C++ and Java, Vancouver, Canadá, Out. 1998..143

[ISL96] ISLAM, N.; PRODROMIDIS, A.; SQUILLANTE, M.S. Dynamic Partitioning in Diferent Distributed-Memory Environments. In: IPPS' 96 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.1162, Santa Barbara, Honolulu, Hawaii, abril, 1996.

[KIC93] KICZALES, GREGOR; ASHLEY, J. MICHAEL Metaobject Protocols: Why We Want Them, and What Else They Can Do, publicado no Object-Oriented Programming: The CLOS Prospective, págs. 101-118, Andreas Paepcke, Ed., MIT Press, Cambridge, MA, EUA, 1993.

[LAU97] XU, C.; LAU, F.C.M. Load Balancing in Parallel Computers: Theory and Practice. Kluwer Academic Publishers, Boston, USA, 1997.

[LIN87] LIN, F.C.H.; KELLER, R.M. The Gradient Model Load Balancing Method. IEEE Transactions on Software Engineering, v.SE-13, n.1, janeiro, 1987.

[LUL93] LÜLING, R.; MONIEN, B. A Dynamic Distributed Load Balancing Algorithm with Provable Good Performance. In: Proceedings of ACM Symposium on Parallel Algorithms and Architectures (SPAA-93), 1993.

[MAC92] MACHADO, F.B., MAIA, L.P. Introdução à Arquitetura de Sistemas Operacionais. Rio de Janeiro, LTC-Livros Técnicos e Científicos, 1992.

[MUL93] MULLENDER, S. Distributed Systems. 2a. ed., New York, Addison-Wesley Publishing Company, 1993.

[OLI98] OLIVA, ALEXANDRE. Guaraná: Uma Arquitetura de Software para Reflexão Computacional Implementada em Java™. Dissertação submetida à UNICAMP para obtenção de grau de Mestre em Ciência da Computação. Campinas, Ago. 1998.

[OUS88] OUSTERHOUT, J.K. Scheduling Techiniques for Concurrent Systems. In: Proceedings Third International Conference on Distributed Computing Systems, IEEE, p22-30, 1982.

[QUI94] QUINN, M.J. Parallel Computing: Theory and Practice. 2.ed., New York, McGraw-Hill Inc., 1994.

[RAP98] RAPINE, C.; SCHERSON, I.D.; TRYSTRAM, D. On-line sheduling of parallelizable jobs. In: Euro-Par' 98 Parallel Processing - Fourth International Euro-Par Conference, Lecture Notes in Computer Science v.1470, Southampton, UK, p.322-327, setembro, 1998.

[RUS97] RUSS, S.H.; MEYERS, B.; ROBINSON, J.; GLEESON, M.; RAJAGOPALAN, L.; TAN, C-H.; HECKEL, B. User-Transparent Run-Time Performance Optimization. In: EHPC'97 - the 2nd International Workshop on Embedded HPC Systems and Applications at the 11th IEEE International Parallel Processing Symposium, 1997.

[SAP95] SAPHIR, W.; TANNER, L.A.; TRAVERSAT, B. Job Management Requirements for NAS Parallel Systems and Clusters. In: IPPS' 95 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.949, Santa Barbara, CA, USA, abril, 1995.

[SHA93] SHAY , WILLIAM A. **Introdution to Operating Systems**. HarperCollins College Publishers, 1993.

[SHI92] SHIRAZI, B.; HURSON, A.R. Special Issue on Scheduling and Load Balancing: guest editor's introduction. Journal of Parallel and Distributed Computing, v.16, Iss 4, p.271- 275, 1992.

[SHI95] SHIRAZI(b), B.A.; HURSON, A.R.; KAVI, K. M. Mechanisms for Process Migration. Introdução do Sexto Capítulo do Livro Scheduling and Load Balancing in Parallel and Distributed Systems, IEEE Computer Society Press, Los Alamitos, CA, USA, p.411-413, 1995.

[SHV92] SHIVARATRI, N.G.; KRUEGER, P.; SINGHAL, M. Load Distributing for Locally Distributed Systems. IEEE Computer, dezembro, 1992.

[SIN94] SINGLETON, J.; SHARTZ, M. Data access in the information warehouse framework. IBM System Journal, 1994.

[SNI96] SNIR, M.; OTTO, S.; STEVEN, H.; WALKER, D.; DONGARRA, J.J. MPI: he Complete Reference. The MIT Press, Massachusetts, 1996.

[SOB96] SOBEL, JONATHAN; FRIEDMAN, DANIEL P. Introduction to Reflection-Oriented Programming. Reflection'96, San Francisco, CA, EUA, Abr. 1996. [online] http://www.cs.indiana.edu/hyplan/jsobel/rop.html

[TAN92] TANENBAUM, ANDREW S. Modern Operating Systems. Prentice Hall, 1992.

[TAN95] TANENBAUM, ANDREW S., Modern Operating Systems. Prentice-Hall, Inc, 1995.

[TAT99] TATSUBORI, MICHIAKI An Extension Mechanism for the Java Language. Dissertação de Mestrado em Engenharia, Universidade de Tsukuba, Ibaraki, Japão. Fev. 1999. [online] http://www.hlla.is.tsukuba.ac.jp/~mich/openjava/papers/mich_thesis99.pdf

[WAN85] WANG, Y.T.; MORRIS, R.J.T. Load Sharing in Distributed Systems. IEEE Transactions on Computers, p.204-217, março, 1985.

[ZAN97] ZANCANELLA, Luiz C. Estrutura Reflexiva para Sistemas Operacionais Multiprocessados. Dissertação de Doutorado, Universidade Federal de Rio Grande do Sul, , Instituto de Informática, UFRGS, Porto Alegre, RS, 1997.