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

Alex Sandro Roschildt Pinto

ABORDAGEM DE ESCALONAMENTO DINÂMICO DE TAREFAS BASEADA EM SISTEMAS CLASSIFICADORES

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciências da Computação.

MARIO ANTONIO RIBEIRO DANTAS, Dr.
Orientador

Florianópolis - SC, Agosto de 2004.

ANEXO 5 - Página de Aprovação

Abordagem de Escalonamento Dinâmico de Processos Baseada em Sistemas Classificadores

Alex Sandro Roschildt Pinto

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

_____

Raul S. Wazlawick, Dr.
Coordenador

Banca Examinadora

_____

Mario Antonio Ribeiro Dantas, Dr.

_____

Alba C. Melo, Dra.

_____

Frank A. Siqueira, Dr.

_____

Luis Fernando Friedrich, Dr.

Dedico este trabalho a Fadinha
que me mostrou que é mais
fácil voar do que caminhar.
Te amo Fê.

Agradeço aos meus pais,
pela força que sempre me deram.
A todos os meus colegas da UFSC,
pela ajuda e críticas ao meu trabalho.
Aos meus colegas do LABWEB,
nosso convívio me ensinou muito.
Ao professor Mario Dantas,
por sempre acreditar no meu trabalho
e por me ensinar o
real “sentimento” da pesquisa.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

1PX	Cruzamento de um ponto
2PX	Cruzamento de dois pontos
AG	Algoritmos Genéticos
CBLB	*Classifier-Based Load Balancer*
COTS	*Components off the shelf*
COW	*Cluster of Workstations*
CPU	*Central Processing Unit*
CGTA	*Genetic Central Task Assigner*
NOW	*Network of Workstations*
OTS	*On-time assignment*

Resumo

A utilização de agregados de computadores está cada vez mais presente no contexto computacional atual. Um dos grandes problemas de tais ambientes é a má alocação dos recursos computacionais. O módulo de escalonamento de processos é um importante componente para a melhoria de distribuição das cargas do sistema. Enquanto o escalonamento estático é utilizado nos casos em que o comportamento dos programas é previamente conhecido, o escalonamento dinâmico torna-se necessário em casos onde o comportamento dos processos é desconhecido. As soluções de escalonamento adaptativas tomam decisões com base nos parâmetros atuais do sistema. Desta forma, são capazes de adaptarem-se às variações do ambiente. Nesta dissertação, apresentamos uma abordagem de escalonamento dinâmico de processos baseado em sistemas classificadores. Sistemas classificadores são algoritmos de aprendizado de máquina, baseados em algoritmos genéticos altamente adaptáveis. Em adição, apresentamos um modelo de sistema computacional que é testado sob o paradigma de um sistema classificador. Nossos resultados demonstram um diferencial na capacidade de adaptação do sistema classificador mediante o ambiente sob o qual está inserido.

Abstract

Cluster configurations are a cost effective scenarios which is becoming a reality to enhance several classes of applications in many organizations. The load distribution of processes to processors into this distributed environment is an interesting issue to be tackled. Therefore, the scheduling module is one important component that can improve the performance of any software package. In this article, we present an approach to enhance the load balancing of a distributed environment, using a system based on dynamic characteristics of a classifier system. Classifier systems are genetic-based machine learning algorithms with an interesting adaptive characteristic. Using this technique, we implemented a test bed contribution which had shown successful results to improve the load balancing of a distributed configuration.

1.Introdução

O crescente avanço das tecnologias de hardware e software, juntamente com a necessidade de desempenho computacional cada vez maior por parte das organizações, tem impulsionado o uso de sistemas paralelos e distribuídos de larga escala. Uma solução economicamente interessante e eficiente é a utilização de agregados de computadores, também conhecidos como *clusters* computacionais (CULLER & JASWINDER, 1999, DANTAS et al, 2000). Atualmente diversas opções de software são oferecidas para a construção de agregados de computadores cada qual com características distintas. Dentre alguns exemplos podemos citar o Oscar (PINTO et al, 2004a), o OpenMosix (MAYA et al, 2004) e o Linux Virtual Server (ZHANG, 2004).

Um dos grandes problemas em tais sistemas é o desenvolvimento de técnicas efetivas de distribuição de processos entre os nodos do agregado (CASAVANT & KHUL, 1988). Segundo (ZHOU, 1988), em um agregado de computadores existe uma grande probabilidade de um nodo ficar sobrecarregado, enquanto outros ficam ociosos (DANTAS & ZALUSKA, 1998, DANTAS et al, 2000). Tal problema, conhecido como desbalanceamento de carga, degrada o desempenho do sistema como um todo, uma vez que uma distribuição de processos mais efetiva diminui o tempo de resposta computacional dos processos.

O problema de escalonamento é reconhecidamente como NP-completo (PAPADIMITRIOU & STEILGLITZ, 1998). Por esta razão, é comum a utilização de métodos heurísticos ou estocásticos, uma vez que esses fornecem soluções quase ótimas em tempo razoável. Dentre os diversos métodos utilizados, pode-se notar uma expressiva utilização de algoritmos genéticos (ZOMAYA & TEH, 2001, HOU et al, 1994, GREENE, 2001 ,ZOMAYA et al, 1999, CORRÊA & MELO, 2001, BAUMGARTNER et al, 1995, WOO et al, 1997). Nesta dissertação apresentamos uma abordagem de escalonamento de processos através de sistemas classificadores (GOLDBERG, 1989, HOLLAND, 1984). Esses sistemas apresentam uma abordagem de aprendizado de máquina através de algoritmos genéticos, e possuem grande capacidade de adaptação mesmo quando submetidos a ambientes adversos.

Os resultados experimentais obtidos (PINTO & DANTAS, 2004c), demonstram o alto grau de adaptabilidade do método proposto. Além disso, o pacote de software implementado permitiu a execução de testes tanto em ambiente Windows quanto Linux. Em ambas configurações de agregado de computadores foi possível observar a capacidade de aprendizagem do sistema classificador proposto. Observa-se ainda um ganho de desempenho mediante os métodos *threshold* e *random*.

O capítulo 2 da presente dissertação, apresenta os conceitos de escalonamento de processos, balanceamento de carga e migração de processos. Também é apresentada uma breve introdução de agregados de computadores. Conceitos sobre algoritmos genéticos e sistemas classificadores são apresentados no capítulo 3. O sistema proposto, juntamente com a arquitetura utilizada, descrição do sistema implementado, modelo de sistema classificador utilizado e algoritmo de balanceamento de carga proposto são detalhados no capítulo 4. Os resultados experimentais e discussão são apresentados no capítulo 5. Finalmente as conclusões e sugestões de trabalhos futuros são apresentadas no capítulo 6.

2. Escalonamento de Processos e Balanceamento de Carga em Sistemas Distribuídos e Paralelos

2.1. Introdução

O problema de escalonamento de processos é uma das questões mais críticas na construção de um sistema distribuído(WANG & MORRIS, 1985). Segundo (CASAVANT & KHUL, 1988), podemos considerar um escalonador de processos como um componente que faz a gerência de recursos. O problema de escalonamento de processos pode ser divido em três principais componentes: consumidores, recursos e política de balanceamento. Neste caso podemos considerar os consumidores como os processos que esperam ser executados. Os recursos são os diversos processadores que irão executar os processos. Por fim, a política de escalonamento é o método pelo qual os processos serão encaminhados aos seus respectivos processadores.

Quando um sistema de escalonamento é avaliado, duas propriedades devem ser levadas em consideração: 1) a satisfação dos consumidores com relação ao modo como o escalonador gerencia os recursos disponíveis (desempenho), e 2) a satisfação dos consumidores em termos de quão difícil ou custoso é acessar o próprio mecanismo de gerenciamento de recursos (eficiência). Em outras palavras, os consumidores desejam estarem aptos a acessar de forma rápida e eficiente o recurso atual em questão, mas não desejam sofrer o atraso de utilizar a função de escalonamento.

De acordo com (ZHOU, 1988, KREMIEN & KRAMER, 1992), em um agregado de computadores existe uma grande probabilidade de um nodo ficar sobrecarregado, enquanto outros ficam ociosos. Tal problema, conhecido como desbalanceamento de carga, degrada a performance do sistema como um todo, uma vez que uma distribuição de processos mais efetiva diminui o tempo de resposta computacional dos processos, aumentando assim o desempenho do sistema. Desta forma, o módulo de balanceamento de carga juntamente com o módulo de escalonamento de processos, executam um papel fundamental em um agregado de computadores, pois aumentam o desempenho do sistema computacional como um todo.

2.2. Agregados de Computadores

Os recentes avanços nas tecnologias de microprocessadores e redes locais de alto desempenho têm criado a possibilidade de utilização de agregados de computadores como eficientes ambientes paralelos para execução de um grande número de aplicações. Essa abordagem baseia-se na idéia de utilização de diversos computadores comerciais facilmente encontrados no mercado (COTS – *Components Off The Shelf*) interligados por uma tecnologia de rede local, visando um aumento no desempenho das aplicações, através da exploração da distribuição ou paralelismo(DANTAS, 2002, CULLER & JASWINDER, 1999). O paradigma de agregado visa um aumento do desempenho das aplicações, através de uma maior taxa de execução dos aplicativos e um aumento no número de dados a serem considerados na execução. Em um ambiente de agregado de computadores, pode-se imaginar que para atingir o objetivo de melhoria de desempenho, primeiramente devemos considerar alguns esforços, tais como: aumento na velocidade do processador, uso de algoritmos mais otimizados e adoção de um ambiente de computação concorrente (ou paralela) (DANTAS, 2002).

Diversas configurações são empregadas na construção de agregados de computadores. A utilização de computadores comuns ligados por uma rede de interconexão, também conhecida como NOW(*Network of Workstations*), pode tirar vantagem de ciclos de CPU ociosos das máquinas de uma rede local (THOMAS et al, 1995). Neste caso as máquinas são utilizadas tanto como máquinas de usuários comuns quanto como nodos(máquinas) pertencentes do agregado. O escalonador de processos Condor (LITZKOW et al, 1988), tenta maximizar a utilização das máquinas de uma NOW com o mínimo de interferência nos processos dos usuários destas. O sistema identifica máquinas ociosas e escalona processos em segundo plano para elas. Quando o usuário volta a utilizar a máquina, o sistema migra o processo para outra máquina ociosa.

Os COW (*Clusters of Workstations)*, podem ser considerados como uma evolução dos NOW. Ao contrário das NOW, os nodos do agregado são projetados unicamente para executar aplicações paralelas. Atualmente diversas opções de software são oferecidas para a construção de agregados de computadores cada qual com

características distintas. Dentre alguns exemplos podemos citar o Oscar (PINTO et al, 2004a), o OpenMosix (MAYA et al, 2004) e o Linux Virtual Server (ZHANG, 2004). O pacote OSCAR, facilita a instalação de agregados de computadores e constrói agregados baseados na arquitetura mestre-escravo. Neste tipo de abordagem enquanto o nodo mestre coordena todo o funcionamento do agregado, os nodos escravos são utilizados unicamente para processamento. OpenMosix baseia-se na abordagem NOW, aproveitando os ciclos de CPU ociosos das máquinas. O Linux Virtual Server, por sua vez, é empregado para construção de agregados de servidores de internet (servidores que disponibilizam serviços http).

2.3. O Gerenciador OSCAR

Segundo (PINTO et al, 2004a), o OSCAR é um pacote de software que permite simplificar a complexa tarefa de utilização e gerenciamento de um agregado. OSCAR é essencialmente utilizado para a computação de alto desempenho, podendo perfeitamente ser utilizado por qualquer aplicação, que necessite das funcionalidades de um agregado, para obter um aumento em seu desempenho através da exploração do paralelismo. Cabe ressaltar alguns pacotes padrões instalados nativamente, como implementações de MPI (*Message Passing Interface*) (MPICH, 2004) e de PVM (*Parallel Virtual Machine*) (PVM, 2003).

As bibliotecas de troca de mensagens como MPI e PVM fornecem a capacidade necessária ao ambiente OSCAR para a criação de programas paralelos em um ambiente de computação distribuída.

2.4. Taxonomia do Escalonamento de Processos

A taxonomia de escalonamento proposta por (CASAVANT & KHUL, 1988) pode ser observada na figura 2.1.

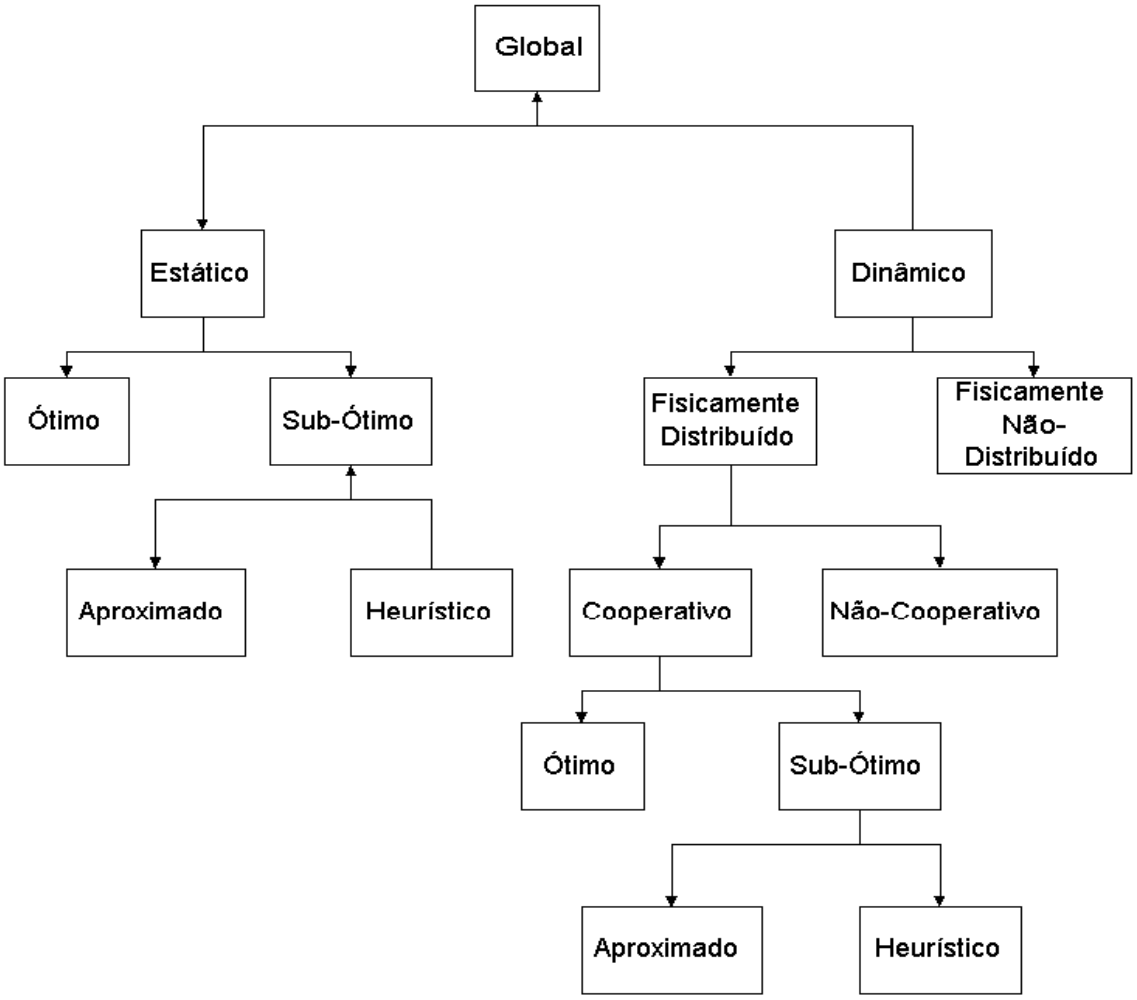

Figura 2.1: Taxonomia do escalonamento de processos (CASAVANT & KHUL, 1988)

O nível mais alto da taxonomia distingue o escalonamento de processos em local e global. O escalonamento local diz respeito aos tempos de alocação de processador que cada processo possui (*time-slice*). O escalonamento global por sua vez é o problema de decisão de onde executar um certo processo e o escalonamento local fica a cargo do sistema operacional do nodo para onde o processo foi alocado. Esta característica aumenta a autonomia dos processadores em um sistema multiprocessador, uma vez que reduz a responsabilidade (e o *overhead*) do mecanismo de escalonamento global. Note que isto não implica na transferência da responsabilidade do escalonamento para uma única autoridade centralizada (CASAVANT & KHUL, 1988).

2.4.1. Escalonamento Estático versus Dinâmico

O próximo nível da hierarquia (na parte do escalonamento global) é a escolha entre escalonamento estático e dinâmico. Tal escolha indica o momento em que as decisões de escalonamento são tomadas. O escalonamento estático define o balanceamento das cargas antes da execução. Essa abordagem apresenta boa eficiência em ambientes homogêneos e processos cujo comportamento pode ser previsto em tempo de compilação. Desta forma, as informações a respeito da natureza dos processos é conhecida antes dos mesmos serem executados.

O escalonamento dinâmico é empregado nos casos em que as necessidades dos processos não são previamente conhecidas. Desta forma, o algoritmo de escalonamento deve consultar o estado do sistema constantemente. De acordo com (CASAVANT & KHUL, 1988), no escalonamento dinâmico nenhuma decisão é tomada antes que o processo seja criado no ambiente. Entretanto, deve-se levar em conta o *overhead* das constantes consultas ao estado do sistema que não são executadas no escalonamento estático. Segundo (ZHOU, 1988), existem basicamente dois *overheads* principais apresentados em um escalonamento dinâmico. O primeiro seria em função dos índices medidos em cada nodo e a tarefa de enviar tais informações para o nodo que toma as decisões de escalonamento. O segundo seria o tempo de espera de cada processo até que a decisão de onde o mesmo será executado seja tomada. Uma vez que é utilizado neste trabalho o escalonamento dinâmico, vamos nos deter a especificar a taxonomia de classificação deste.

2.4.2. Escalonamento Distribuído versus Não-Distribuído

O próximo nível do escalonamento dinâmico, diz respeito à distribuição ou não da responsabilidade do escalonamento. Um único nodo pode ser responsável pela tarefa de escalonar os processos entre os diversos nodos de um sistema distribuído, sendo que neste caso o escalonamento é não-distribuído. A tarefa de escalonamento pode ser executada ainda por vários nodos, que podem cooperar ou não entre si (CASAVANT & KHUL, 1988). O método proposto nesta é não-distribuído, uma vez

que a tarefa de escalonamento fica a cargo do nodo mestre do agregado de computadores.

2.4.3. Escalonamento Adaptativo versus Não-Adaptativo

A taxonomia de Casavant (CASAVANT & KHUL, 1988), classifica ainda o escalonamento como adaptativo ou não-adaptativo. Uma solução adaptativa para o problema do escalonamento é aquela em que os algoritmos e parâmetros utilizados para implementar a política de escalonamento mudam dinamicamente de acordo com os comportamentos atuais e anteriores do sistema frente às decisões anteriores do sistema de escalonamento. Como exemplo, poderíamos tomar um escalonador de processos adaptativo que leva em consideração vários parâmetros antes de tomar suas decisões. Em resposta ao comportamento do sistema, o escalonador pode começar a ignorar determinado parâmetro se ele acredita que tal parâmetro está provendo informações inconsistentes com o resto das informações recebidas (KREMIEN & KRAMER, 1992).

De acordo com (KREMIEN & KRAMER, 1992), o intervalo com que os dados de estado do sistema são coletados assim como a periodicidade com que os parâmetros são ajustados, estão intimamente ligados com a eficiência do método. Ao contrário de um escalonador adaptativo, um escalonador não-adaptativo não modifica o controle básico de seu funcionamento com base no histórico de atividades do sistema. Um exemplo seria um escalonador que sempre considera o estado recebido do sistema da mesma forma, independente do histórico de comportamento do sistema.

2.5. Balanceamento de Carga

A idéia básica do balanceamento de carga é tentar equilibrar o nível de carga de todos os nodos de determinado sistema distribuído, de forma que o tempo de execução dos processos seja aproximadamente igual em todos os nodos (CASAVANT & KHUL, 1988). De acordo com (ZHOU, 1988), um algoritmo de balanceamento de carga é composto basicamente de três componentes:

- Política de informações: especifica a quantidade de informação de carga e processos disponível ao módulo responsável por tomar a decisão do nodo em que os processo irão executar, e o modo pelo qual a informação é distribuída no sistema;
- Política de transferência: determina a possibilidade dos processos serem transferidos de acordo com a natureza destes e estado atual dos nodos;
- Política de localização: decide, dentre os processos escolhidos pela política de transferência, para onde os processos serão transferidos.

Os três componentes citados acima não agem isoladamente uns dos outros, mas interagem de diversas formas: a política de localização utiliza as informações de carga disponibilizadas pelo módulo de política de informações, e age somente sobre os processos considerados aptos a serem transferidos pelo módulo de política de transferência.

A política de seleção também faz parte de algumas soluções. Segundo (SHIVARATRI et al, 1992), tal política é responsável pela determinação dos processos que devem ser transferidos. Quando a política de transferência decide que um determinado nodo é transmissor, a política de seleção deve escolher que processo deve ser transferido. A abordagem mais simples é escolher sempre os processos recém chegados ao nodo, ou seja, os processos que tornaram o nodo um transmissor. Desta forma não é considerada a transferência preemptiva (quando o processo é transferido no meio da sua execução).

A chegada aleatória de processos a serem executados em um ambiente distribuído pode acarretar em um desbalanceamento de carga, com isso, enquanto alguns nodos ficam sobrecarregados, outros ficam ociosos. O papel do módulo de balanceamento de carga é justamente aumentar o desempenho de sistemas distribuídos através da transferência de processos de nodos sobrecarregados para nodos ociosos (SHIVARATRI et al, 1992). A figura 2.3 ilustra o caso de um agregado de computadores com carga desbalanceada.

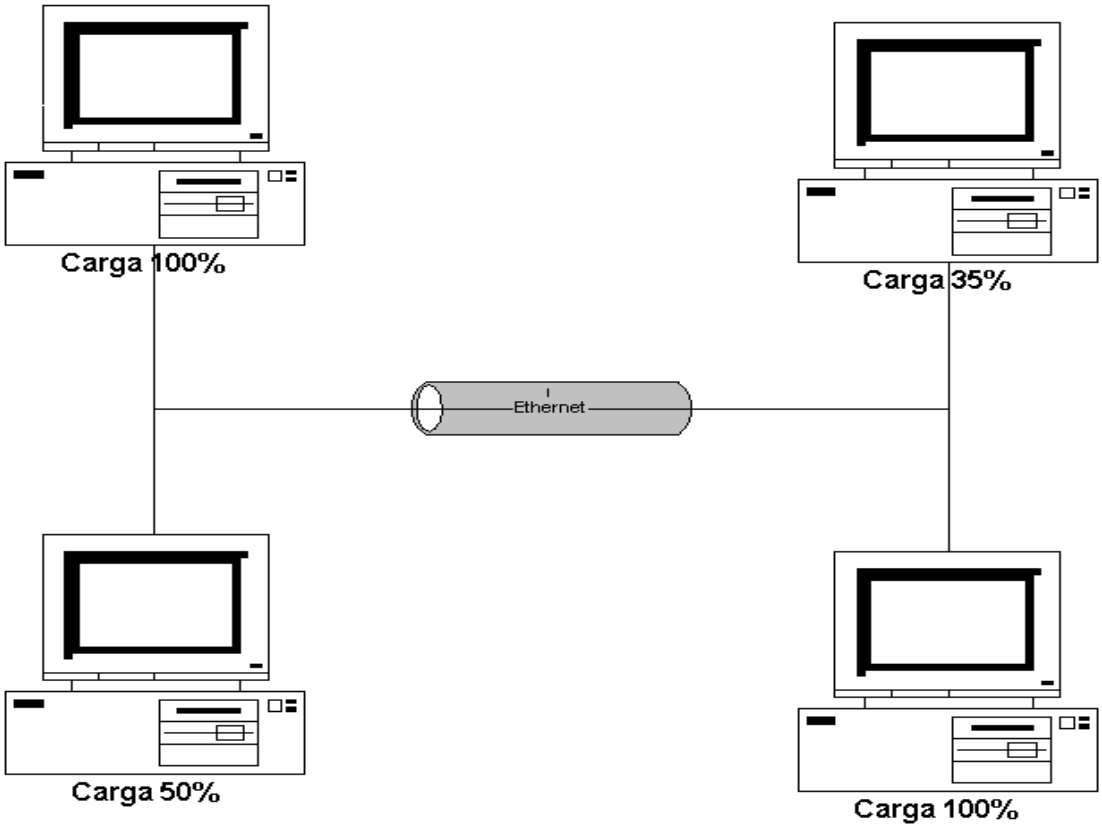

Figura 2.2: Agregado de Computadores com Carga Desbalanceada

2.5.1. Política de Informação

A política de informação é responsável por decidir quando a informação dos nodos deve ser coletada, de que lugar esta informação será coletada e que tipo de informação será coletada. Segundo (SHIVARATRI et al, 1992), existem três tipos de política de informação:

1. Políticas baseadas na demanda: de acordo com esta política descentralizada, determinado nodo coleta o estado de outros nodos somente quando este se torna transmissor de processos ou receptor de processos. Tal política é considerada dinâmica, uma vez que suas ações estão ligadas ao estado do ambiente. A política baseada na demanda pode ser inicializada pelo transmissor, receptor ou simetricamente inicializada. Na política inicializada pelo transmissor, o transmissor procura por receptores para os quais ele possa transferir sua carga. A abordagem de política inicializada pelo receptor, transfere para o receptor a função de solicitar processos dos transmissores. A política simetricamente

inicializada é a combinação das duas políticas demonstradas acima. Tanto pode ser inicializada pelo transmissor quanto pelo receptor.

2. Políticas periódicas: tais políticas, as quais podem ser tanto centralizadas quanto descentralizadas, coletam informações sobre o sistema periodicamente. De acordo com a informação coletada, a política de transferência decide sobre a transferência dos processos. Políticas de informações periódicas geralmente não adaptam-se ao estado atual do sistema. Por exemplo, os benefícios do balanceamento de carga são mínimos em sistemas com grande carga, uma vez que a maioria dos nodos do sistema estão sobrecarregados. Além disso, o custo da coleta periódica do estado do sistema o sobrecarrega ainda mais.

3. Políticas baseadas na mudança de estado: De acordo com tais políticas, os nodos disseminam seus estados toda vez que seus estados mudam em determinado valor. A política baseada na mudança de estado difere da política baseada em demanda, uma vez que dissemina a informação a respeito de um nodo, ao invés de coletar informações a respeito dos outros nodos. Em uma política baseada na mudança de estado centralizada, os nodos enviam seus estados a um nodo centralizador de informação. Na política descentralizada, a informação é enviada a todos os nodos.

2.5.2. Política de Transferência

A política de transferência determina se um nodo está em estado apropriado de participar de uma transferência de processos, tanto assumindo o papel de transmissor quanto de receptor (SHIVARATRI et al, 1992). A literatura apresenta diversas políticas de transferência baseadas na abordagem de intervalo (*threshold*) (DANTAS et al, 2000, EAGER et al, 1986). Nesta abordagem, os intervalos são expressados em unidades de carga, ou ainda segundo o número de processos. Quando um novo processo é gerado em um nodo, a política de transferência determina se este nodo é um transmissor, se a carga daquele nodo exceder um intervalo T_1 (máximo). Por

outro lado, se a carga do nodo cair abaixo de T_2 (mínimo), a política de transferência decide que o nodo será receptor de processos (SHIVARATRI et al, 1992).

De acordo com (EAGER et al, 1986), a escolha do "melhor" *threshold* é dependente da carga do sistema e do custo de transferência dos processos. Um *threshold* baixo é mais apropriado quando a carga é baixa, uma vez que muitos nodos estão ociosos. *Threshold*s altos são mais apropriados quando a carga do sistema é alta, pelo motivo que os nodos possuem grande número de processos na fila de processos prontos.

Segundo (EAGER et al, 1986), o objetivo da política intervalo é evitar transferências inúteis de processos (aquelas em que o nodo receptor já está no limite máximo ou acima deste). Outra política de transferência é a política aleatória (*random*), que é considerada uma das políticas mais simples pois não utiliza nenhuma informação para a tomada de decisão. Nesta abordagem, um nodo é selecionado aleatoriamente e o processo é transferido para este. Nenhuma troca de informação de estado entre os nodos é requerida para decidir o local de transferência do processo (EAGER et al, 1986).

2.5.3. Política de Localização

A responsabilidade da política de localização é encontrar um "parceiro" apropriado para a transferência de processos (transmissor ou receptor) para determinado nodo, uma vez que a política de transferência já tenha decidido que o nodo é um transmissor ou receptor (SHIVARATRI et al, 1992). Uma política descentralizada amplamente utilizada é a que encontra o nodo mais apropriado através de eleição (*polling*): um nodo elege outro a fim de pesquisar seu estado e verificar a possibilidade de transferência de processo. Os nodos podem ser eleitos tanto de forma serial quanto de forma paralela (*broadcast*, por exemplo). Um nodo pode ser eleito aleatoriamente, com base nas informações coletadas na última votação ou ainda de forma a tentar eleger os nodos mais próximos. Um modo alternativo de realizar a votação é enviar uma pergunta (*query*) procurando qualquer nodo disponível para participar do compartilhamento de carga (SHIVARATRI et al, 1992).

2.5.4. Balanceamento de Carga iniciado pelo Transmissor

As abordagens de balanceamento de carga inicializadas pelo transmissor deixam a cargo dos nodos sobrecarregados a inicialização da transferência dos processos. Desta forma, os nodos sobrecarregados tentam enviar seus processos para os nodos com carga menor (SHIVARATRI et al, 1992).

O trabalho de (EAGER et al, 1986), estudou três tipos de algoritmos de balanceamento de carga inicializados pelo transmissor. Nesta seção, iremos descrever estes algoritmos e suas respectivas políticas de transferência, seleção, localização e informação.

- **Política de transferência**: todos os três algoritmos utilizam a mesma política de transferência, uma abordagem *threshold* que leva em consideração o tamanho da fila de CPU. Segundo esta abordagem, um nodo é considerado transmissor se um novo processo que chega a este nodo faz com que o tamanho da fila de CPU ultrapasse o *threshold T*. Da mesma forma, um nodo se identifica como receptor se seu tamanho de fila de CPU está abaixo de *T*.

- **Política de seleção**: todos os três algoritmos utilizam a mesma política de seleção, considerando somente tarefas recém-chegadas ao sistema. Não consideram migração preemptiva.

- **Política de localização**: Os algoritmos propostos por (EAGER et al, 1986), diferem na sua política de localização, as quais são descritas a seguir.

  Random: esta política de localização não utiliza nenhuma informação de estado dos nodos. O processo é simplesmente transferido para um nodo selecionado aleatoriamente, sem que nenhuma troca de informação entre os nodos que estão participando da transferência seja efetuada para auxiliar na tomada de decisão. Transferências inúteis são freqüentes; estas podem ocorrer quando é escolhido um nodo que já possui tamanho de fila de CPU igual ou maior que o *threshold T* (determinado pela política de transferência). O problema é como os nodos tratam um processo recém-

chegado, já que um processo recém-transferido pode ser novamente transferido para outro nodo. Desta forma, os nodos do sistema podem gastar todo seu tempo transferindo processos e não os executando. Uma saída simples é limitar o número de vezes que um processo pode ser transferido.

Threshold: esta política de localização evita transferências inúteis, consultando o estado dos nodos antes de tomar a decisão. Desta forma, se a nova transferência for ultrapassar o *threshold* do receptor, a transferência não é executada. Quando a transferência do processo for acarretar em uma *threshold* maior que o permitido, outro nodo é escolhido ao acaso. A fim de diminuir o *overhead* do sistema, o número de consultas aos nodos é determinado pelo parâmetro *poll limit*(limite de consultas). Os nodos são consultados até o limite de consultas ser atingido. Se nenhum nodo for encontrado até o limite de consultas ser atingido, o próprio transmissor deverá executar a tarefa. Esta abordagem consegue melhor desempenho que a abordagem *random*, uma vez que evita transferências inúteis.

Shortest: As duas abordagens apresentadas anteriormente, não tentavam encontrar o nodo mais apropriado para a transferência. A abordagem *shortest* seleciona um determinado número de nodos (de acordo com o *poll limit*) ao acaso, e determina seus respectivos tamanhos de fila de CPU. O nodo com menor tamanho de fila de CPU é escolhido, a menos que a transferência faça o *threshold* do receptor escolhido ser ultrapassado. O nodo destino irá executar o processo independentemente do tamanho de sua fila no momento da chegada do processo. Esta abordagem não apresentou melhoras de desempenho significativas em relação à abordagem *threshold*. Desta forma, a utilização de informações mais detalhadas não implica necessariamente em um melhor desempenho (EAGER et al, 1986).

- **Política de Informação**: ambas as abordagens (*threshold* e *shortest*), utilizam política de informação baseadas em demanda. Ou seja, quando

um nodo é identificado como transmissor, a pesquisa de estados é executada.

2.5.5. Balanceamento de Carga iniciado pelo Receptor

Os algoritmos de balanceamento de carga iniciado pelo receptor levam em consideração os nodos ociosos (ou com uma baixa carga) na inicialização do balanceamento de carga. Nesta abordagem, um nodo com baixa carga tenta "receber" processos de nodos sobrecarregados. Nesta secção iremos descrever o algoritmo apresentado nos trabalhos (EAGER et al, 1986b, LIVNY & MELMAN, 1982).

- **Política de transferência**: a política utilizada é baseada no algoritmo *threshold*, baseado no tamanho da fila de CPU. A política de transferência é acionada sempre que um novo processo chega ao sistema. Se o tamanho da fila de CPU local ficar abaixo do *threshold*, então o nodo é identificado como receptor de processos de um determinado nodo (transmissor). O nodo transmissor é determinado pela política de localização. Um nodo é identificado como transmissor se o tamanho de sua fila de CPU ultrapassar o *threshold T*.
- **Política de seleção:** O algoritmo considera todos os processos elegíveis para transferência, podendo utilizar tanto abordagens preemptivas quanto não-preemptivas.
- **Política de Localização:** A política de localização seleciona um nodo aleatoriamente e consulta o mesmo para determinar se a transferência de um processo iria deixar seu tamanho de fila de processos abaixo do *threshold*. Caso não seja o caso, o nodo transfere o processo. Em caso afirmativo, outro nodo é selecionado aleatoriamente, este processo é repetido até um transmissor ser encontrado ou o limite de consultas (*poll limit*) ser ultrapassado.

  O problema com este tipo de política de localização é que se todas as consultas falharem em encontrar um transmissor, o poder de processamento disponível no receptor será totalmente desperdiçado. Este problema afeta seriamente o desempenho de sistemas onde somente alguns nodos geram a maioria da carga de trabalho. Nestes sistemas, a

consulta aleatória dos receptores pode facilmente falhar. Umas das alternativas, se todas as consultas falharem, seria ou o receptor esperar um período pré-determinado de tempo para iniciar uma nova consulta ou esperar até que outro processo chegue.

- **Política de informação:** A política de informação é baseada em demanda uma vez que, as consultas são iniciadas somente depois que um nodo torna-se receptor.

2.5.6. Balanceamento de Carga Simetricamente Inicializado

As atividades de distribuição de carga em algoritmos de balanceamento simetricamente inicializados, são iniciadas tanto por receptores quanto por transmissores (SHIVARATRI et al, 1992). Tais algoritmos possuem tanto as vantagens dos algoritmos inicializados pelos receptores quanto pelos transmissores. Em sistemas com baixa carga de trabalho, o componente inicializado pelo transmissor obtém maior sucesso em encontrar nodos com baixa carga. Sistemas com alta carga, podem se beneficiar das características do componente inicializado pelo receptor, pois deste modo é mais fácil encontrar transmissores.

Apesar das vantagens, este tipo de balanceamento de carga pode sofrer de desvantagens de ambos os tipos de balanceamento (iniciados pelo receptor e pelo transmissor). Em algoritmos inicializados pelo transmissor, a consulta em sistemas sobrecarregados pode resultar em instabilidade do sistema. Em algoritmos inicializados pelo receptor, torna-se necessária a migração preemptiva. Um algoritmo simetricamente inicializado simples pode ser construído pela combinação das políticas de transferência e localização de algoritmos inicializados pelo transmissor e receptor.

2.5.7. Algoritmos de Balanceamento de Carga Adaptativos

O grande atrativo das políticas de balanceamento de carga estática é sua simplicidade: ” transfira todas as compilações originadas no nodo X para o nodo Y” ou “ ... para os nodos Y e Z com probabilidades 0,8 e 0,2 respectivamente”.

Entretanto, tais políticas são limitadas, uma vez que não reagem ao estado atual do sistema (EAGER et al, 1986).

Segundo (EAGER et al, 1986), o principal atrativo das políticas adaptativas é que elas respondem ao estado do sistema, e são mais aptas a evitar os estados do sistema de baixo desempenho. Entretanto, a implementação de tais políticas é mais complexa. Geralmente estes algoritmos coletam grande quantidade de informação a respeito do sistema. Diversos pontos devem ser observados em tais abordagens. Uma das preocupações é com o *overhead* que a coleta de informação do sistema pode acarretar. Outro problema, são as decisões errôneas tomadas pelo algoritmo, que são inevitáveis em tais políticas.

Um algoritmo de balanceamento de carga adaptativo desempenha basicamente duas funções: a coleta do estado do sistema e a tomada de decisão (KREMIEN & KRAMER, 1992). As coletas de estado devem ser implementadas de forma a minimizar o *overhead* e ainda conseguir representar a real situação do sistema. As tomadas de decisão devem tentar diminuir o tempo médio de resposta dos processos através de migração preemptiva e não-preemptiva.

2.5.8. Migração de Processos

O módulo de balanceamento de carga é responsável por transferir processos de entre nodos (SHIVARATRI et al, 1992). Quando a migração é preemptiva (o processo pode ser transferido para outro nodo durante a sua execução), o processo deve ser migrado juntamente com seu contexto (espaço de endereçamento, *links* e arquivos abertos) (POWELL & MILLER, 1983). Os mecanismos utilizados na migração de processos podem ser úteis também na recuperação de falhas. A migração de processos provê a habilidade do sistema de parar determinado processo, transportar o estado corrente do processo para outro processador, e reiniciar o processo. Se a informação necessária para transportar determinado processo é salva em um sistema de armazenamento confiável, é possível migrar um processo de um processador que estiver com problemas para outro processador operante. A migração de processos torna-se mais complexa em sistemas que não compartilham memória

principal, como é o caso dos agregados de computadores. A migração de processos em sistemas que compartilham memória principal é trivial (SMITH, 1988).

Apesar dos benefícios pela migração de processos pode oferecer, a implementação de tal módulo é difícil e custosa (ARTSY & FINKEL 1989). O mecanismo de realocação de processos deve mover os processos de maneira confiável e eficiente, transferir o processo juntamente com o seu contexto, e ainda inserir o processo em um novo contexto (a máquina destino). A migração pode ser interrompida por causa de falhas nas máquinas ou na comunicação entre elas. Entretanto, é possível implementar uma abordagem menos rígida de escalonamento dinâmico chamada *on-time assignment (OTS)* (CASAVANT & KHUL, 1988). A técnica *OTS* é considerada uma abordagem dinâmica, no qual a decisão do nodo aonde o processo vai ser executado é feita logo após a criação do processo e não considera migração preemptiva. Nesta dissertação foi utilizada a técnica *OTS*.

2.6. Métricas Utilizadas no Balanceamento de Carga

Uma grande variedade de índices de carga foram explícita ou implicitamente utilizados na literatura, a maioria em esquemas de balanceamento de carga. O trabalho de (EAGER et al, 1986) utiliza como métrica o custo de transferência dos processos. Outras pesquisas utilizam a carga da CPU ou ainda o tamanho da fila de CPU (FERRARI & ZHOU, 1988). Vários trabalhos apresentaram com meta a redução do tempo médio de resposta dos processos (*mean response time*) (EAGER et al, 1986, KREMIEN & KRAMER, 1992, PINTO & DANTAS, 2004c ZHOU, 1988). Em alguns destes trabalhos uma estimativa do tempo médio de resposta foi utilizada como índice de carga. Embora vários destes trabalhos tenham apresentado resultados positivos, uma justificativa científica para a escolha de tais métricas geralmente não é demonstrada (FERRARI & ZHOU, 1988).

Segundo (FERRARI & ZHOU, 1988), um bom índice de carga deve satisfazer os seguintes requisitos:

- Ser capaz de refletir estimativas qualitativas da carga corrente em um nodo;
- Ser útil na previsão da carga em um futuro próximo, desde que os tempos de resposta dos processos sejam mais afetados pela carga futura do que pela carga presente do sistema;

- Ser relativamente estável , flutuações de alta-freqüência na carga podem ser descontadas, ou desconsideradas;
- Ter uma relação simples (idealmente linear) com o índice de desempenho: deste modo o valor pode ser facilmente traduzido para o desempenho esperado.

3. Algoritmos Genéticos e Sistemas Classificadores

3.1. Introdução

Algoritmos genéticos são algoritmos de busca, baseados nos mecanismos de seleção natural (GOLDBERG, 1989, HOLLAND, 1984, MITCHELL, 1996). No início dos anos 70, John Holland, um inovador no campo da ciência computacional, inspirou-se na teoria da evolução para criar um algoritmo para computador. O objetivo de Holland não era, simplesmente, desenvolver outro método de otimização, mas sim criar uma abordagem teórica, assim como procedimentos para desenvolvimento de programas genéricos e máquinas com capacidade ilimitada de adaptação a ambientes arbitrários.

De acordo com os mecanismos de seleção natural os indivíduos mais adaptados possuem mais chances de sobreviver e deste modo repassar seu código genético para seus descendentes. Em um algoritmo genético os indivíduos de uma população são representados pelos seus cromossomos (genótipo), que usualmente são representados por um conjunto de caracteres. A cada nova geração um novo conjunto de criaturas artificiais (conjuntos de caracteres) são gerados, com base nos fragmentos de material genético dos indivíduos mais aptos das gerações passadas.

O foco central da pesquisa dos algoritmos genéticos é a robustez. O balanço entre eficácia e eficiência é requerido para a sobrevivência nos mais diversos ambientes. Se os sistemas artificiais forem mais robustos, o custo de redefinição de tais sistemas pode ser reduzido, ou eliminado. Sistemas que atingem níveis mais altos de adaptação, são capazes de executar melhor e por mais tempo. Características para auto-reparo, auto-orientação, e reprodução são regras em sistemas naturais, visto que eles raramente existem nos mais sofisticados sistemas artificiais (GOLDBERG, 1989).

3.2. Funcionamento dos Algoritmos Genéticos

Para que os algoritmos genéticos sejam capazes de sobrepujar outras técnicas de busca em termos de robustez, esses devem diferir em alguns pontos fundamentais. Diferenças existentes entre os algoritmos genéticos e outros processos de busca, podem ser entendidos através dos seguintes aspectos (GOLDBERG, 1989):

- Os algoritmos genéticos trabalham com a codificação do conjunto de parâmetros, não com os próprios parâmetros;
- A busca é efetuada em uma população de pontos, não em um único ponto;
- Os algoritmos genéticos utilizam informações da função objetivo, não necessitando de conhecimentos adicionais;
- Os algoritmos utilizam regras de transição probabilísticas e não determinísticas.

A maioria dos métodos de otimização se move ponto a ponto no espaço de busca. Tal método ponto a ponto é extremamente perigoso uma vez que pode localizar ótimos locais em espaços de busca multimodais. Algoritmos genéticos contornam tal problema trabalhando com vários pontos em paralelo, deste modo a probabilidade de retornar um ótimo local é reduzida. Enquanto a maioria dos métodos de busca necessita de informações auxiliares para funcionar, a única informação que os algoritmos genéticos necessitam é o valor de adaptabilidade (*fitness*) dos indivíduos. Deste modo, algoritmo genético é considerado um método de busca cego.

O processo de evolução de um algoritmo genético começa com a definição de uma população inicial de possíveis respostas para o problema, na qual geralmente os indivíduos são inicializados ao acaso. A partir desta população inicial, três operadores são utilizados: seleção, reprodução(*crossover*) e mutação. A cada iteração do processo evolutivo são escolhidos pares de reprodutores (através do método de seleção); tal seleção é feita com base na adaptabilidade dos indivíduos. Uma vez selecionados os pares de reprodutores, estes são cruzados (através do operador de reprodução). Os descendentes são gerados com base no material genético dos seus descendentes e podem ou não ter seu cromossomo mutado (através do operador de mutação). A população de respostas é então renovada, de acordo com critério de reposição previamente escolhido e uma nova iteração é iniciada. O critério de parada da evolução pode ser por convergência (quando a adaptação média da população de soluções não aumenta a um

certo número de gerações) ou por um número de gerações previamente definido. A função de um algoritmo genético simples é demonstrado na figura 3.1.

O mecanismo de funcionamento de um algoritmo genético comum é relativamente simples, envolvendo nada mais complexo do que cópias e trocas parciais de cadeias de caracteres. A simplicidade do método aliada ao poder de busca dos algoritmos genéticos tem atraído um número cada vez expressivo de pesquisadores para a área. Apesar da facilidade de implementação, dois pontos são considerados críticos na construção de um algoritmo genético: a escolha de uma representação para o cromossomo e a função de cálculo de aptidão. A estrutura de dados que representa o cromossomo deve ser capaz de representar todas as possíveis instâncias de resposta para o problema. O cálculo de aptidão, por sua vez, deve ser capaz de traduzir com o máximo de precisão a qualidade da possível resposta (indivíduo).

```
AlgoritmoGeneticoSimples(){
Inicializa população;
Avalia população;
Enquanto critério de parada não atingido{
    Seleciona pares para reprodução;
    Executa reprodução e mutação;
    Avalia população;
    }
}
```

Figura 3.1: Algoritmo Genético Simples

3.3. Representação Cromossômica

A escolha de uma estrutura de dados capaz de representar a solução para o problema a ser tratado no algoritmo genético é uma das etapas fundamentais na construção deste (SRINIVAS & PATNAIK, 1994). A escolha da estrutura de dados está intimamente ligada com a natureza do problema a ser tratado. Além da estrutura de dados que mais se adapta ao problema, deve ser levado em conta a escolha do alfabeto a ser utilizado. Originalmente foi utilizada a representação binária, tal representação é ainda a mais utilizada atualmente (MITCHELL, 1996). Apesar disto, outras formas de representação são empregadas (exemplo: valores reais ou inteiros).

3.4. Função de Adaptabilidade

A função objetivo (a função a ser otimizada) provê o mecanismo para avaliação de cada indivíduo da população (SRINIVAS & PATNAIK, 1994). Entretanto, a faixa de valores utilizada varia de problema para problema. A fim de manter a uniformidade sobre vários problemas, a função de adaptabilidade é utilizada para normalizar a função objetivo para uma faixa conveniente (geralmente de 0 a 1). O valor normalizado da função objetivo é a adaptabilidade (*fitness*) do indivíduo; tal valor é utilizado para avaliar cada individuo na etapa da seleção de reprodutores.

3.5. Seleção de reprodutores

A escolha do método de seleção de reprodutores é um dos pontos críticos na implementação de sistemas que utilizem algoritmos genéticos. Tal etapa tem como objetivo garantir que indivíduos mais adaptados tenham maior probabilidade de serem escolhidos como reprodutores e repassar seu código genético para as próximas gerações. A seleção de reprodutores é um dos fatores que guia o processo evolucionário. Quando critérios de seleção muito rígidos são utilizados, a dominância de indivíduos mais adaptados é induzida e desta forma o processo tende a convergência prematura. Métodos muito relaxados de seleção, por sua vez, podem acarretar em uma desaceleração do processo evolucionário (MITCHELL, 1996). Diversos métodos de seleção de reprodutores são apresentados na literatura(GOLDBERG, 1989, PINTO & BORGES, 2004b). Dentre os métodos utilizados podemos citar:

- Seleção por giro de roleta: Neste método, cada indivíduo da população é representado na roleta proporcionalmente ao seu índice de aptidão. Assim, aos indivíduos com alta aptidão, é dada uma porção maior da roleta, enquanto aos de aptidão mais baixa, é dada uma porção relativamente menor da roleta. Finalmente, a roleta é girada um determinado número de vezes, dependendo do tamanho da população, e são escolhidos, como indivíduos que participarão da próxima geração, aqueles sorteados na roleta.

- Seleção por ranking: os indivíduos são ordenados de acordo com seu valor de adequação e então sua probabilidade de escolha é atribuída conforme a posição de origem. O propósito de tal método é tentar previnir a convergência prematura do algoritmo genético, quando os genes dos indivíduos dominantes dominam a população e a evolução para.

- Seleção por torneio: a seleção por torneio utiliza o método de roleta para selecionar dois indivíduos e então seleciona o mais adaptado entre eles. Este procedimento é realizado diversas vezes até que todos os reprodutores sejam escolhidos.

- Seleção por exclusão social: foi criado com base em uma sociedade hipotética que além de monogâmica tivesse plena noção da aptidão de seus indivíduos. Desta forma após a população ser ordenada decrescentemente em função da aptidão, cada indivíduo terá o direito de cruzar com o indivíduo de aptidão imediatamente inferior a sua e gerar um ou dois filhos. Se tomarmos como exemplo uma população de 500 indivíduos, com taxa de 10% de reposição, aonde cada par de reprodutor gerasse dois descendentes, somente os 50 indivíduos mais adaptados teriam oportunidade de gerar descendentes. Neste caso o indivíduo mais adaptado da população cruzaria com o segundo mais adaptado, o terceiro com o quarto e assim por diante (PINTO & BORGES, 2004b).

3.6. Operadores de cruzamento

Após os pares serem escolhidos, estes irão gerar um ou dois descendentes de acordo com uma taxa de probabilidade de cruzamento. Caso não gerem descendentes, os pares podem ser clonados. Existem diversas formas de combinar o genótipo dos indivíduos selecionados para o cruzamento. A escolha do operador de cruzamento mais apropriado deve levar em conta as características do problema a ser resolvido. Os principais operadores de cruzamento são (GOLDBERG, 1989):

- cruzamento de um ponto (1PX) : cruzamento utilizado para cromossomas de tamanho fixo. Os cromossomos de cada par de indivíduos a serem cruzados são

particionados em um ponto, chamado ponto de corte, sorteado aleatoriamente. Um novo cromossomo é gerado permutando-se a metade inicial de um cromossomo com a metade final do outro. Ou seja, dados dois genomas x e y de tamanho *tam*, sorteia-se um número *num* entre 0 e tam, o primeiro filho receberá todos os genes de x de 1 até num e todos os genes de y de num + 1 até tam: o segundo filho por sua vez receberá o inverso do primeiro filho.

Exemplificando, considerado-se dois genomas "A" e "B", com as seguintes decodificações binárias:

Pai "A" = 0 1 1 0 1

Pai "B" = **1 1 0 0 0**

Se for escolhida a posição de corte em K = 4, ou seja, após a quarta posição do genoma, após o cruzamento teremos a seguinte configuração, em decorrência da troca, conforme destacado em negrito:

Filho "A" = 0 1 1 0 **0**

Filho "B" = **1 1 0 0** 1

- cruzamento por 2 pontos (2PX): utiliza metodologia semelhante ao cruzamento por um ponto, a diferença é que neste caso dois pontos de corte são sorteados. O exemplo a seguir utiliza-se de dois genomas, codificados na forma alfa-numérica, com a seguinte codificação:

  Pai “1” = **A B C D E F**

  Pai “2” = G H I J L M

  Considerando que sejam sorteados os pontos de corte 1 e 4, logo os filhos de tal cruzamento terão a seguinte configuração:

  Filho “1” = **A** H I J **E F**

  Filho “2” = G **B C D** L M

3.7. Operador de Mutação

Após os descendentes serem gerados, existe uma probabilidade destes receberem mutação em seu cromossoma. Geralmente é escolhido um gene ao acaso e o valor deste é substituído por outro do alfabeto válido. O operador de mutação possui fundamental importância para o algoritmo genético, pois aumenta a biodiversidade da população de respostas.

3.8. Atualização

A etapa de atualização é responsável pela manutenção da população, após as etapas de reprodução e de mutação, seleciona-se, segundo critérios pré estabelecidos, os indivíduos que irão compor a próxima geração e os que serão descartados da população atual. Deste modo, tenta-se garantir a convergência para uma solução ótima do problema. Convencionalmente, o AG mantêm um número fixo de indivíduos na população sendo estes substituídos por completo por seus descendentes. Além desta abordagem de atualização de população, existem metodologias que garantem que os melhores indivíduos sejam sempre mantidos, ou ainda, que admitam um número variado de indivíduos na população.

3.9. Finalização

O critério de parada do AG não envolve o uso de nenhum operador genético, simplesmente determina-se um teste que dá fim ao processo de evolução. Pode-se limitar o número de gerações ou definir o grau de convergência da atual população, ou seja, o grau de proximidade dos valores de avaliação de cada indivíduo da população (GOLDBERG, 1989).

3.10. Parâmetros Genéticos

Além do funcionamento do AG em si, é importante também, analisar de que maneira alguns parâmetros influem no comportamento destes, para que se possa estabelecê-los conforme as necessidades do problema e dos recursos disponíveis.

- Tamanho da População: o tamanho da população afeta o desempenho global e a eficiência dos AGs. Com uma população pequena o desempenho pode cair, pois, deste modo, a população fornece uma pequena cobertura do espaço de busca do problema. Uma grande população geralmente fornece uma cobertura representativa do domínio do problema, além de prevenir convergências prematuras para soluções locais ao invés de globais. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais, ou que o algoritmo trabalhe por um período de tempo muito maior.

- Taxa de Cruzamento: quanto maior for esta taxa, mais rapidamente novas estruturas serão introduzidas na população. Mas se esta for muito alta, estruturas com boas aptidões poderão ser retiradas mais rapidamente, a maior parte da população será substituída. Com um valor baixo, o algoritmo pode tornar-se muito lento.

- Taxa de Mutação: Altas taxas de mutação introduzem uma grande quantidade de indivíduos distintos (genótipo diferente) na população e assim aumentam a biodiversidade.

- Intervalo de Geração: controla a porcentagem da população que será substituída durante a próxima geração. Com um valor alto, a maior parte da população será substituída, podendo ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

3.11. Sistemas Classificadores

Sistema classificadores são sistemas de aprendizado de máquina, capazes de aprender regras sintaticamente simples chamadas classificadores, com o propósito de guiar sua performance em um ambiente arbitrário (GOLDBERG, 1989, HOLLAND, 1984). Um sistema classificador consiste de três componentes principais:

- Sistema de regras e mensagens;
- Sistema de divisão de créditos (*apportionment of credit*);
- Algoritmo Genético.

O sistema de regras de mensagens de um sistema classificador é um tipo especial de sistema de produção. Um sistema de produção é um esquema computacional que usa regras como seu único dispositivo algorítmico, as regras são geralmente da seguinte forma:

se<condição>então<ação>

O significado da regra de produção é que a ação deve ser tomada, quando a condição é satisfeita. Os classificadores geralmente possuem um alfabeto ternário {0,1,#}, aonde # é considerado o símbolo *don't care,* pode significar tanto 0 quanto 1.

Uma mensagem recebida do sistema pode ativar um ou mais classificadores. Podemos tomar como exemplo o conjunto de classificadores apresentados na tabela 3.1:

Tabela 3.1: Exemplo de conjunto de classificadores

Condição	ação
10#01#	100
11#1#0	111
0#1111	001
100001	110

Se o sistema receber do ambiente a mensagem "101011", o primeiro classificador será ativado e a ação "100" será executada. Um sistema classificador é capaz de adaptar seus classificadores, de forma a privilegiar os classificadores que enviam

ações positivas para o ambiente em que este está inserido. Desta forma, o sistema classificador é capaz de adaptar-se a ambientes adversos.

Na criação do sistema classificador todos os classificadores possuem a mesma aptidão. Quando um classificador é escolhido, este deve "pagar" uma parte de sua aptidão ao Sistema de divisão de créditos. A quantia a ser paga ao sistema de divisão de créditos é determinada através de uma taxa pré-definida. Quando mais de um classificador satisfizerem uma determinada condição, será escolhido aquele que tiver a maior quantia a pagar ao sistema de divisão de créditos. Esta quantia será paga ao sistema classificador que gerou tal mensagem, se a ação que este executou no sistema tiver sido positiva. Caso contrário, o montante é armazenado e pago ao próximo classificador que enviar uma ação positiva para o sistema. De tempos em tempos o algoritmo genético é ativado, e os classificadores são renovados. Geralmente um número de consultas ao sistema classificador é pré-definido. O algoritmo genético deve agir após a aptidão dos classificadores ter sido ajustada pelo uso do sistema classificador.

3.12. Trabalhos Relacionados

A utilização de algoritmos genéticos (AG) no escalonamento de processos já foi amplamente discutida na literatura(ZOMAYA et al, 1999, ZOMAYA & TEH, 2001, HOU et al, 1994, CORRÊA & MELO, 2001, BAUMGARTNER et al, 1995, WOO et al, 1997). Nesta seção iremos discutir as abordagens de (BAUMGARTNER et al, 1995) e (CORRÊA & MELO, 2001) que baseiam-se em sistemas classificadores. A utilização de métodos estocásticos ou heurísticos para a resolução de problema do balanceamento dinâmico de processos pode ser justificada pela conhecida complexidade de tal problema (NP-completo) (PAPADIMITRIOU & STEILGLITZ, 1998).

As abordagens que utilizam AG podem ser utilizar o algoritmo genético diretamente (ZOMAYA et al, 1999, ZOMAYA & TEH, 2001, HOU et al, 1994, ou ainda utilizar sistemas classificadores (BAUMGARTNER et al, 1995, CORRÊA & MELO, 2001, PINTO & DANTAS, 2004c). Embora as abordagens baseadas em

algoritmos genéticos apresentem bom desempenho, estas possuem a desvantagem de ter que evoluir a população de respostas para efetuar o balanceamento (ZOMAYA & TEH, 2001). Desta forma, esta abordagem é mais eficiente em balanceamento estático(ZOMAYA et al, 1999, HOU et al, 1994). Sistemas classificadores não possuem esta desvantagem, já que os classificadores são armazenados e podem ser submetidos para o ambiente sem a necessidade de ter que evoluir a população a cada balanceamento de carga. Tal característica é importante, pois o tempo de evolução da população de resposta pode degradar o desempenho do balanceamento de carga e conseqüentemente do sistema.

No trabalho de (BAUMGARTNER et al, 1995) , são apresentados dois algoritmos de balanceamento de carga: GCTA (*Genetic Central Task Assigner*) e CBLB (*Classifier-Based Load Balancer*). Ambas as abordagens demonstraram um aumento de desempenho de sistemas paralelos e distribuídos sobre uma grande variedade de domínios de aplicação. As duas abordagens foram testadas em um simulador e o método CBLB foi testado ainda em uma máquina Intel Hypercube.

O método GCTA é utilizado em um algoritmo centralizado, aonde um único nodo do sistema periodicamente coleta informações de estado dos outros nodos. Tal informação consiste do número de tarefas (NT_i) esperando serem executadas. O número total de processos em espera do sistema é dado por NT_s. Após receber a informação de carga, o processador central executa o algoritmo GCTA para determinar a melhor distribuição de cargas do sistema. Esta nova configuração é enviada para todos os nodos que então transferem os processos para seus novos nodos e continuam a execução. Um dos maiores problemas de tal abordagem é a espera pela evolução do algoritmo genético para gerar uma nova configuração.

O método CBLB emprega um sistema classificador simples em conjunto com o algoritmo Centex (ZHOU, 1988). O algoritmo Centex é executado em todos os nodos do sistema e ainda em um nodo central, mas os parâmetros do algoritmo são manipulados no nodo central. O sistema classificador recebe como condição de entrada dos nodos o tamanho das filas de CPU (NT_i), o tempo médio de resposta dos processos(Rup_i) , o tempo médio de serviço (Sup_i)e o número de processos recebidos (Aup_i). Todos estes parâmetros são enviados pelos nodos segundo um

período de tempo e refletem o estado do sistema desde a ultima atualização. O nodo central de posse destes valores calcula três parâmetros:

1. Tempo médio de resposta desde a última atualização Rup:

$$Rup = \frac{1}{N}\sum_{i=0}^{N} \mathrm{Rup_i}$$

Este parâmetro é utilizado como métrica que indica o sucesso ou não da última condição submetida pelo sistema classificador.

2. Utilização média por nodo desde a última atualização Uup

Uup é calculada primeiramente pela taxa de recebimento de processos do sistema $\lambda = \frac{1}{Up}\sum_{i=0}^{N} Aup_i$ Uma vez que $1/\lambda = S/U$ para todo o sistema, a utilização média de cada nodo é calculada por:

$Uup = \frac{S\lambda}{N}$. A utilização por nodo indica uma medida da carga de processos no sistema desde a úlima atualização.

3. O inverso do desvio padrão de processos recebidos desde a última atualização $1/\delta(A)$, tal métrica produz uma estimativa do desbalanceamento de carga baseado no recebimento de processos Aup desde a ultima execução do sistema classificador.

Os classificadores propostos por (BAUMGARTNER et al, 1995) possuem três ações:

1. Threshold da fila da processos Tq;
2. Período de atualização do estado do sistema UP;
3. Threshold de carga da CPU;

De acordo com as ações e condições apresentadas acima a configuração dos classificadores é a seguinte:

<c1>+<c2>+<c3>:<a1>+<a2>+<a3>

A tabela 4.1 descreve os possíveis valores de cada condição e ação.

Tabela 4.1: Formato dos Classificadores em CBLB(BAUMGARTNER et al, 1995)

Campo	Numero de Bits	Descrição	Codificação
C1	1	*Flag* do tempo médio de resposta do sistema	0 decrementou 1 incrementou
C2	2	Nível médio de utilização Uup	00 < 0.25 01 < 0.50 10 < 0.75 11 < 1.00
C3	2	Nível de desbalanceamento de carga	00 < 0.25 01 < 0.50 10 < 0.75 11 < 1.00
A1	3	Modificar o *threshold* da fila de processos	Tq = valor binário
A2	3	Modificar o período de envio de estado do sistema	Up = valor binário
A3	3	Modificar o *threshold* de CPU	Tcpu=valor binário*0.25

A recompensa dos classificadores que enviam ações positivas para o ambiente no método CBLB, baseia-se no tempo médio de resposta dos processos assim como o método apresentado neste trabalho. O classificador que conseguir diminuir o tempo médio de resposta recebe como recompensa $1/Rup$, sendo que inicialmente todos os classificadores criados possuem uma aptidão de 100.

O sistema proposto por (CORRÊA & MELO, 2001) utiliza um tipo especial de sistema classificador XCS (DEJONG, 1975). Tal sistema foi utilizado em conjunto com um simulador de agregado de computadores que utilizava um algoritmo de balanceamento de carga baseado no tempo de vida do processo, o sistema utiliza a

equação tempo_vida_processo >=α*(custo_migração), se o tempo de vida de determinado processo for maior ou igual a α*(custo_migração) então este será migrado. O problema é a determinação do melhor α, que depende das condições do agregado de computadores. O sistema classificador proposto por (CORRÊA & MELO, 2001) tenta então ajustar α da melhor maneira possível.

As abordagens de (BAUMGARTNER et al, 1995, CORRÊA & MELO, 2001) basearam-se basicamente na simulação. Ambos trabalhos apresentaram resultados que demonstraram a eficiência de sistemas classificadores no balanceamento de carga. A abordagem apresentada nesta dissertação tem como diferencial a utilização do sistema classificador em um ambiente real (PINTO & DANTAS, 2004c). Além disso, utilizamos como métrica o tempo médio de resposta do sistema. Esta métrica tem como vantagem proporcionar a escalabilidade do balanceamento de carga. Uma vez que tal métrica não mudará de acordo com o número de nodos escravos. Outro diferencial é a utilização de uma arquitetura mestre-escravo e migração não-preemptiva de processos.

4. Descrição do Sistema Proposto

4.1. Introdução

A grande maioria das pesquisas de escalonamento de processos através de algoritmos genéticos utiliza-se da simulação de sistemas distribuídos para testar o comportamento do escalonador (ZOMAYA & TEH, 2001, CORRÊA & MELO, 2001, BAUMGARTNER et al, 1995) . Neste contexto, a validação do método é facilitada já que os experimentos podem ser repetidos inúmeras vezes. Apesar disto, o comportamento real do sistema só será conhecido quando este for testado em um ambiente real.

A literatura apresenta algumas simulações que comprovam a eficiência de algoritmos genéticos na resolução tanto do problema do escalonamento estático quanto o do escalonamento dinâmico. Visto que somente algumas abordagens apresentam testes práticos, optamos por implementar um protótipo que seja capaz de demonstrar todas os reais problemas da construção de um escalonador de processos dinâmico que utilize um sistema classificador. Uma vez que os parâmetros do sistema deverão ser analisados em tempo de execução e que estes variam com o passar do tempo, acreditamos que a utilização de um algoritmo de aprendizado de máquina, como é o caso de um sistema classificador, é a mais apropriada solução para tal problema.

O primeiro passo para a implementação de um sistema distribuído que utilize um escalonador de processos com base em sistemas classificadores, é a definição do ambiente. Nosso ambiente de teste é baseado em uma arquitetura mestre-escravo (CULLER & JASWINDER, 1999, DANTAS, 2002), aonde o nodo mestre recebe os processos e repassa estes para serem executados nos nodos escravos. Um esquema mais explicativo do ambiente pode ser visto na figura 4.1.

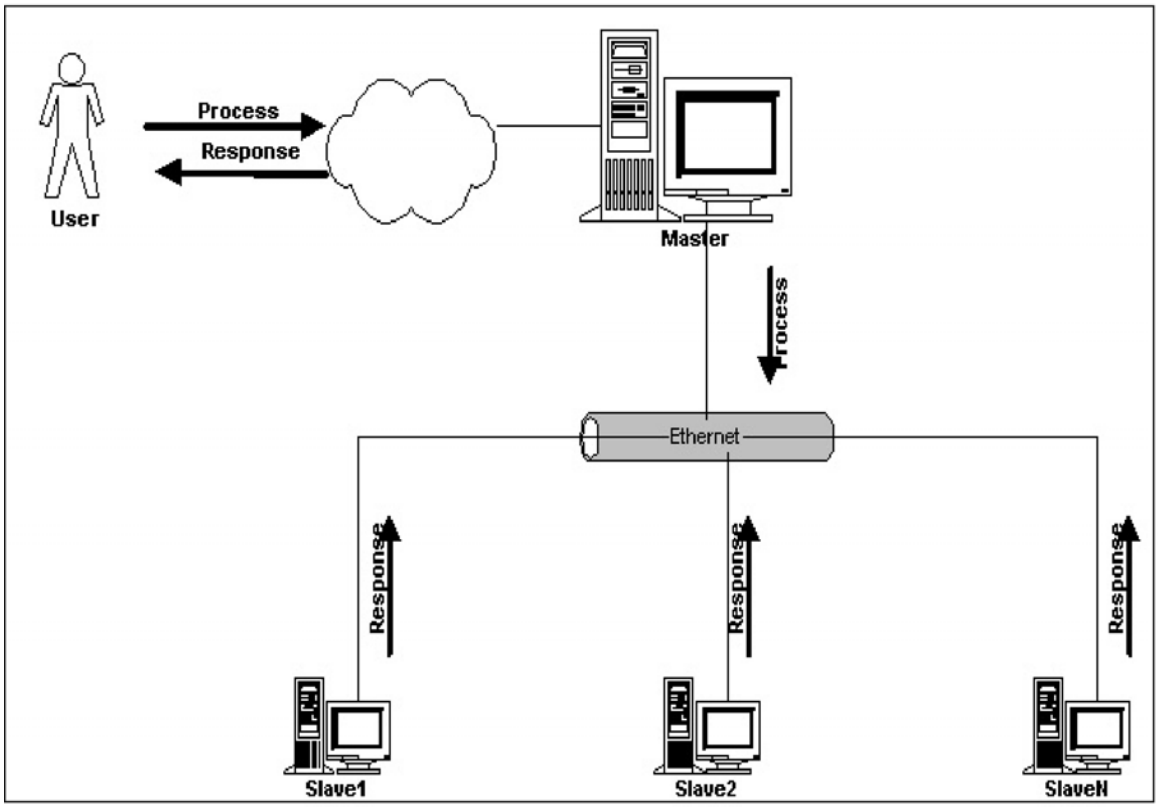

Figura 4.1: Arquitetura do Sistema Proposto

A idéia inicialmente é utilizar as máquinas disponíveis no laboratório LABWEB para executar os testes. Desta forma, será utilizada a abordagem de NOW. Tal arquitetura permite que os processos sejam recebidos de clientes, sejam eles móveis ou fixos. O nodo mestre é responsável por gerenciar todo o escalonamento do sistema, e ainda receber o estado dos nodos escravo. Uma vez que a tarefa de gerenciamento do sistema é totalmente centralizada no nodo mestre, os nodos escravos terão como única função a execução dos processos. Na segunda fase do projeto, depois que o sistema for devidamente testado, o escalonador poderá ser implementado em um ambiente de agregado que utilize arquitetura semelhante tais como OSCAR (PINTO et al, 2004a) e Linux Virtual Server (ZHANG, 2004).

Assim com em (WANG & MORRIS, 1985), assumimos que os processos são logicamente independentes uns dos outros. Deste modo, um processo que chega ao nodo mestre pode ser executado em qualquer nodo escravo. Tal situação é correspondente a um sistema de banco de dados, aonde os dados sejam replicados entre os nodos escravos. Neste caso as consultas chegariam ao nodo mestre e

poderiam ser executadas em qualquer nodo escravo, já que todos possuem a mesma cópia dos dados.

4.2. Descrição dos Módulos do Sistema Proposto

O ambiente implementado pode ser dividido basicamente em dois módulos: módulo mestre e escravo. O módulo mestre recebe os processos dos usuários e armazena estes em uma fila de processos do sistema. O módulo mestre permite a conexão de diversos nodos escravos. Quando o nodo mestre é inicializado, é especificado o número de nodos escravos que este irá atender. Cada nodo escravo é conectado através da inicialização dos serviços do nodo. Uma vez que todos os nodos escravos estão conectados ao nodo mestre, o sistema está pronto para a execução dos processos. O sistema está sendo implementado na linguagem Java devido as facilidades de *threads* e *sockets* que a mesma apresenta. Além disso, aplicativos Java são portáveis e de livre distribuição.

4.2.1. Módulo Mestre

Para cada nodo conectado no nodo mestre, é criada uma *thread* **ThMasterSon**, uma **ThMasterSonStatus** e uma fila de processos. A **ThMasterSon** é responsável por retirar os processos da fila de processos do nodo presente no nodo mestre e enviar para o seu respectivo nodo escravo. **ThMasterSonStatus** recebe o status dos nodos escravos e atualiza o **SystemStatus**, aonde é armazenado o status geral do sistema.

Para simular o recebimento de processos pelo sistema foi criada a ThUser, *thread* que de acordo com um intervalo de tempo pré-determinado insere um processo na fila do sistema. O balanceamento de carga é de responsabilidade do serviço **ThCollector**, que de acordo com o algoritmo de balanceamento utilizado retira os processos da fila de sistema e envia para uma fila de nodo. Serão implementados dois algoritmos clássicos: no método randômico, o nodo destino que irá executar o processo é escolhido aleatoriamente e *threshold* aonde é escolhido um limite de

processos que cada nodo pode executar e o nodo recebe processos até atingir este limite. A figura 4.2 apresenta o diagrama de classes do módulo mestre.

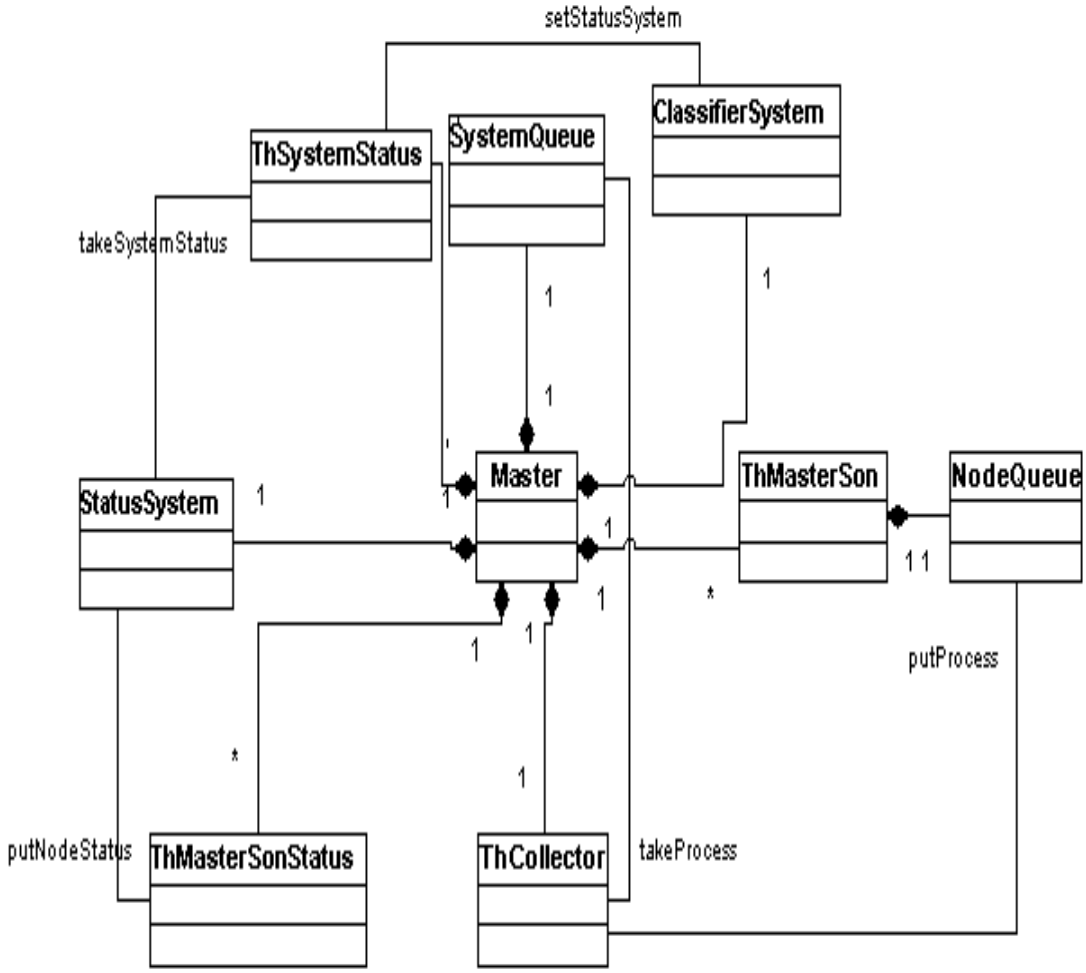

Figura 4.2: Diagrama de Classes do Módulo Mestre

A seguir serão descritas as classes do módulo Mestre.

- **Master**: Responsável por inicializar todos os serviços do sistema. Recebe como parâmetro o número de nodos escravos que serão conectados ao mestre. Cria uma *thread* **ThMasterSon** e uma *thread* **ThMasterSonStatus** e ainda uma fila de processos (**NodeQueue**) para cada nodo escravo conectado. Instância uma classe de **StatusSystem** responsável por armazenar o estado do sistema. Inicializa ainda: **ThCollector**, **ThSystemStatus** e dependendo da política de balanceamento utilizada inicializa o sistema classificador ou não.

- **ThMasterSon**: Responsável por enviar os processos a serem executados pelos nodos escravos (retira os processos da fila do nodo e envia para o nodo propriamente dito). Abre a conexão com o nodo escravo e envia os processos a serem executados.
- **ThMasterSonStatus**: Recebe o estado dos nodos. Logo após, atualiza o estado do nodo em **SystemStatus**.
- **NodeQueue**: Armazena os processos a serem executados pelos nodos escravos.
- **StatusSystem**: Armazena o estado dos nodos escravos (tempo médio de resposta,desempenho e número de processos ativos). Responsável por armazenar ainda o *threshold* do sistema.
- **SystemQueue**: Armazena os processos que chegam ao sistema. Tais processos são submetidos através de **ThUser**(processos que simulam a submissão de tarefas por usuários).
- **ThSystemStatus**: Atualiza os valores de tempo médio de resposta do sistema e gera os arquivos de resultados.
- **ThCollector**: Serviço responsável pelo balanceamento de carga. Tem acesso às filas dos nodos e às filas dos sistemas. De acordo com o balanceamento de carga utilizado no momento, coloca o processo na fila de processos de um determinado nodo.

O processo no sistema proposto foi implementado como um objeto serializável. Deste modo, o mesmo pode ser enviado pela conexão aberta com os nodos escravos para ser executado. Esta implementação permite que a tarefa a ser executada seja facilmente modificada.

4.2.2. Módulo Escravo

Cada nodo do sistema possui um **NodeStatus**, aonde é armazenado o estado atual do nodo (Tempo médio de resposta e numero de processos ativos). O status do nodo é enviado pela **ThNodeStatus**, que possui um intervalo de envio de status pré-determinado. Para cada processo recebido pelo nodo-escravo é criada uma *thread* **ThProc**, ao fim da de cada processo submetido o tempo médio de resposta é

atualizado pela **ThProc**. A figura 4.3 demonstra a o diagrama de classes do módulo escravo.

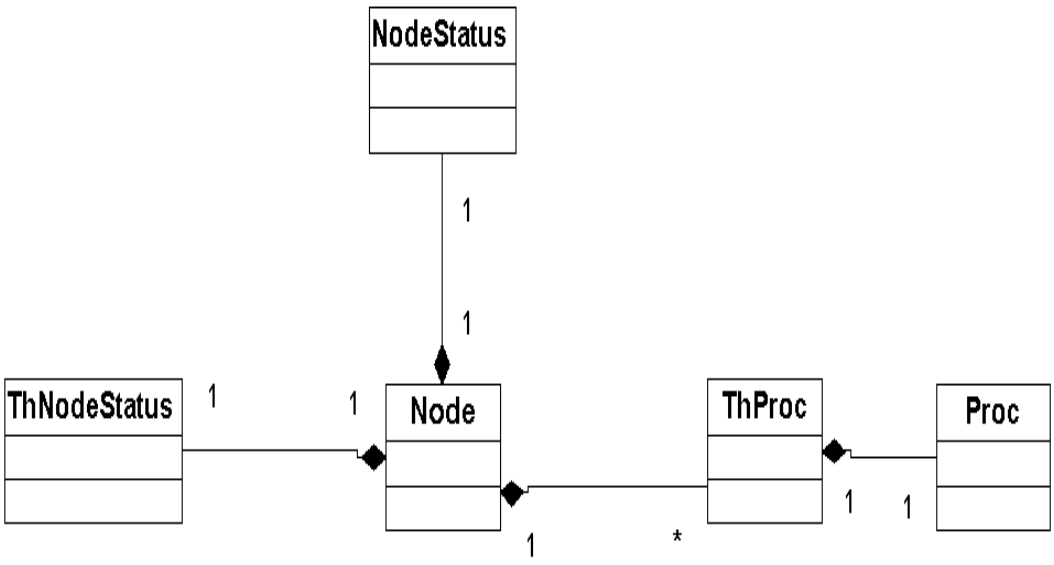

Figura 4.3: Diagrama de Classes do Módulo Escravo

A seguir serão descritas as classes do módulo escravo.

- **Node**: Responsável por inicializar todos os serviços do nodo escravo. Assim como abrir a conexão (*socket*) com o nodo mestre. Recebe os processos a serem executados.
- **ThNodeStatus**: Responsável por atualizar o estado do nodo escravo em questão. Atualiza tempo médio de resposta, desempenho e número de processos ativos. Envia o estado do nodo de acordo com um intervalo pré-determinado.
- **ThProc**: Serviço responsável por executar o processo (**Proc**) recebido por **Node**. Este serviço possibilita que vários processos sejam executados paralelamente nos nodos escravos.

4.3. Balanceamento de Carga Proposto

O desempenho do algoritmo de balanceamento de carga proposto deve ser comparado com o desempenho de algoritmos consagrados. Desta forma, será possível determinar sua eficiência em aumentar o desempenho do agregado de

computadores no qual está inserido. Nesta seção iremos descrever as políticas utilizadas no balanceamento de carga proposto (PINTO & DANTAS, 2004c), assim como outras características do mesmo.

O balanceamento de carga pode ser considerado centralizado, uma vez que todas as decisões são tomadas pelo nodo mestre. Uma vez que estamos trabalhando com uma abordagem mestre-escravo, o módulo mestre nunca irá executar um processo. Desta forma, todos os processos serão recebidos pelo mestre e repassados para os escravos executarem. O nodo mestre centraliza todas as decisões quanto a transferência de processo de acordo com o balanceamento utilizado no momento.

A política de informação é periódica, sendo definido o intervalo de atualização do estado dos nodos na inicialização do sistema. O estado dos nodos é enviado pelo serviço **ThNodeStatus** e recebido pelo serviço **ThMasterSonStatus**. As informações enviadas pelo ThNodeStatus são tempo médio de resposta dos processos, processos ativos e desempenho. O tempo médio de resposta é utilizado pelo sistema classificador para atualizar o *threshold*. O número de processos ativos é utilizado pela política de localização para escolher o nodo que receberá o processo em questão. A política de transferência que todos os processos recebidos pelo nodo mestre deverão ser transferidos para os nodos escravo.

A política de localização pode ser *random*, *threshold* e Sistema Classificador. A política *random* utilizada escolhe um nodo aleatoriamente para receber o processo em questão. A política random utilizada não leva em consideração o estado do nodo que receberá o processo. A política *threshold* envia processos para os nodos escravos até o limite (*threshold*) ter sido atingido. Deve-se ressaltar ainda que esta política sempre escolhe o nodo com menor número de processos. Caso não seja possível transferir o processo para nenhum nodo, uma nova pesquisa é iniciada até que um nodo seja encontrado. O sistema classificador utiliza a mesma política *threshold*. O diferencial é que nossa abordagem tenta adaptar o *threshold* de acordo com a variação do tempo médio de resposta do sistema. Desta forma, nossa abordagem pode ser considerada adaptativa. O serviço **ThClassifier** é responsável por consultar o tempo médio de resposta e submeter o mesmo para o sistema classificador. De posse de tal informação o sistema classificador devolve o melhor classificador

possível (de acordo com a variação do tempo médio de resposta). A resposta do sistema classificador é utilizado por **ThClassifier** para modificar o *threshold*.

4.4. O Sistema Classificador

O sucesso na implementação de um sistema classificador está intimamente ligado com a escolha de uma estrutura para os classificadores que seja capaz de representar todos os possíveis estados do sistema. Nossa proposta é adaptar o método de *threshold* (WANG & MORRIS, 1985). Tal algoritmo estabelece um limite de processos (*threshold*), que cada nodo pode executar, quando este limite é excedido o nodo não recebe novos processos. De acordo com (EAGER et al, 1986), a escolha do melhor *threshold* depende da carga do sistema e do custo de transferência dos processos. Uma vez que tais parâmetros mudam de acordo com a taxa de utilização do ambiente, a melhor abordagem seria a utilização de um algoritmo adaptativo (KREMIEN & KRAMER, 1992, EAGER et al, 1986, DANTAS et al, 2000).

Segundo (EAGER et al, 1986), políticas de balanceamento de carga que utilizam informações simples do sistema possuem desempenho semelhante a aquelas que utilizam parâmetros complexos e não sobrecarregam o sistema. De acordo com (DANTAS et al, 2000), o desempenho de um algoritmo de balanceamento de carga é dependente do índice de carga utilizado e que os índices baseados no tamanho da fila de processos apresentam resultados melhores que outros índices.

De acordo com (FERRARI & ZHOU, 1988), uma grande variedade de índices de carga foram implícita ou explicitamente utilizados na literatura. Nossa proposta é utilizar a variação do tempo médio de resposta dos processos como condição de entrada para o classificador. Desta forma o sistema classificador receberá como condição a variação de incremento ou decremento do tempo médio de respostas e devolverá uma ação que ajustará o *threshold*. O sistema classificador será responsável tanto pela melhora no desempenho do agregado de computadores quanto pelo seu correto funcionamento. Isto deve-se ao fato de que um aumento drástico no valor de *threshold* poderá causar o travamento dos nodos por sobrecarga. O sistema

classificador é responsável por tentar diminuir o tempo de espera dos processos e por garantir a robustez do sistema.

A configuração dos classificadores utilizados é a seguinte:

<Se houve aumento ou decremento>+<porcentagem de variação do tempo médio de resposta>: <Se o *threshold* deve ser incrementado ou decrementado>+<porcentagem de variação do *threshold*>

A tabela 4.2, demonstra as configurações da condição dos classificadores:

Tabela 4.2: Configuração das condições dos classificadores

Primeiro bit "0"	tempo médio de resposta dos processos diminuiu
Primeiro bit "1"	tempo médio de resposta dos processos aumentou
3 próximos bits indicam a variação do tempo médio de respostas	
"000"	de 0 a 15%
"001"	de 15 a 25%
"010"	de 25 a 35%
"011"	de 35 a 45%
"100"	de 45 a 55%
"101"	de 55 a 65%
"110"	de 65 a 85%
"111"	maior que 85%

A tabela 4.3, demonstra a configuração das ações dos classificadores:

Tabela 4.3: Configuração das ações dos classificadores

Primeiro bit "0"	*threshold* será decrementado
Primeiro bit "1"	*threshold* será incrementado
3 próxmos bits indicam a variação que o *threshold* irá sofrer	
"000"	0%
"001"	10%
"010"	20%
"011"	30%
"100"	40%
"101"	60%
"110"	80%
"111"	100%

Quando o sistema classificador é criado, um número determinado de classificadores são gerados aleatoriamente.

A única informação que o sistema classificador possui para ajustar o *threshold* é o ultimo tempo médio de resposta. Aonde Tempo Médio de Resposta = somatório dos tempos de resposta/número de tarefas.

De posse desta informação, a função de consulta codifica a condição de acordo com a tabela 2. O melhor classificador (aquele que possuir maior aptidão), paga a taxa para o sistema de créditos e submete a sua condição para o ThClassifier que irá decodificar a condição de acordo com a tabela 3 e ajustar o *threshold* do sistema. O sistema de divisão de créditos paga a sua quantia total de créditos ao último classificador que enviou uma ação positiva ao sistema, ou seja uma ação que conseguiu diminuir o tempo médio de resposta do sistema.

De acordo com um número determinado de consultas a população de classificadores é evoluída e reposta no conjunto de classificadores de acordo com a taxa de reposição. Existe um número mínimo de classificadores de cada condição que deve estar presente na população de classificadores. Se o descendente gerado possuir o número mínimo de classificadores na população ele é trocado pelo classificador de idêntica condição que possuir a menor aptidão. Caso contrário ele simplesmente é acrescentado a população. Optamos por aplicar o operador de mutação somente na condição dos classificadores. Foi utilizado no algoritmo genético cruzamento de 1 ponto e mutação por troca.

Se possuirmos os classificadores apresentados na tabela 4.4 em nosso sistema classificador:

Tabela 4.4: Exemplo de classificadores

Classificador	Aptidão
#000:0001	230
#0#1:1000	240
1000:0001	300
0000:1100	110
##01:1111	80

A condição 1000 recebida do ambiente, irá retornar os classificadores #000 e 1000 uma vez que o segundo classificador possui maior aptidão este irá pagar a taxa ao sistema de divisão de créditos e submeter a sua ação para o ThClassifier. O

ThClassifier de posse da ação decodifica a mesma e ajusta o *threshold* do sistema, que no caso será decrementado em 10%. Caso a condição não fosse satisfeita por nenhum classificador da população, um novo classificador com a condição recebida e uma ação aleatória seria gerado e adicionado a população.

5. Resultados Experimentais

5.1. Introdução

O desempenho do sistema classificador proposto foi comparado com as políticas *threshold* e *random*. Na primeira fase do trabalho, o sistema foi testado em máquinas com sistema operacional Windows XP®. Na segunda fase, testamos o comportamento do sistema classificador em um agregado de computadores baseado no gerenciador OSCAR (PINTO et al, 2004a). Desta forma, foi possível testar o desempenho do sistema classificador quando inserido em diferentes configurações de agregados de computadores.

Como métrica para comparar o desempenho dos métodos de balanceamento de carga foi utilizado o tempo médio de resposta dos processos. Desta forma, quanto menor o tempo médio de resposta melhor será o desempenho do sistema. Foram utilizados dois tipos de processos. O primeiro é considerado um processo de granularidade fina. Tal processo possui um parâmetro *tamanhoProblema*, e executa um laço for de 1 até *tamanhoProblema*. A cada iteração do laço é executada uma operação de soma. O segundo processo executa o crivo de Eratóstenes, conhecido método de cálculo de números primos. A tabela 5.1 apresenta os parâmetros utilizados nos testes. Os testes realizados tiveram como objetivo principal verificar a adaptabilidade do método proposto mediante processos diferentes e ambientes distintos.

Tabela 5.1: Parâmetros utilizados nos testes computacionais

Parâmetro	Valor
Intervalo de envio de status dos nodos	0,5 segundos
Intervalo de inserção de novos processos na fila do sistema pela *thread* usuário	0,01 segundos
Intervalo de extração do status total do sistema	10 segundos
Threshold	10 processos
Intervalo de atualização do *Threshold*	5 segundos
População inicial de classificadores	200 classificadores
Numero de consultas necessárias para evolução da população de classificadores	50 consultas
Taxa de pagamento ao Sistema Divisor de Créditos	10% da aptidão
Aptidão Inicial dos Classificadores	200
Probabilidade de mutação	1%
Probabilidade de crossover	100%
Taxa de reposição da população de classificadores	10%

5.2. Testes Executados em ambiente Windows XP®

Utilizamos nos testes 1 nodo mestre e 3 nodos escravos, sendo que o nodo mestre foi utilizado para operar como mestre e escravo ao mesmo tempo. Todas as máquinas utilizadas possuem 256 Megabytes de memória principal e processador AMD Duron 1,2 Ghz. Vale ressaltar, que tais máquinas são utilizadas como máquinas de usuário e executaram somente o sistema proposto durante os testes. Desta forma, os testes não sofreram influência de outros aplicativos.

Comparamos o desempenho dos métodos *random*, *threshold* e do sistema classificador proposto. Utilizamos os dois tipos de processos apresentados na seção 5.1. Foram executadas 4 baterias de teste, as quais serão descritas nas próximas seções.

5.2.1 Primeira Bateria de Testes

Na primeira bateria de testes foram utilizadas 4 **ThUser**. Foram executados 4 testes de 10 minutos e o gráfico 5.1 apresenta a média de cada ponto. O tempo médio de resposta do sistema foi extraído a cada 10 segundos. Os processos submetidos eram de tamanho variável, sendo o processo utilizado o primeiro processo descrito na seção 5.1. O gráfico da figura 5.1 apresenta o desempenho dos métodos *random*, *threshold* e sistema classificador.

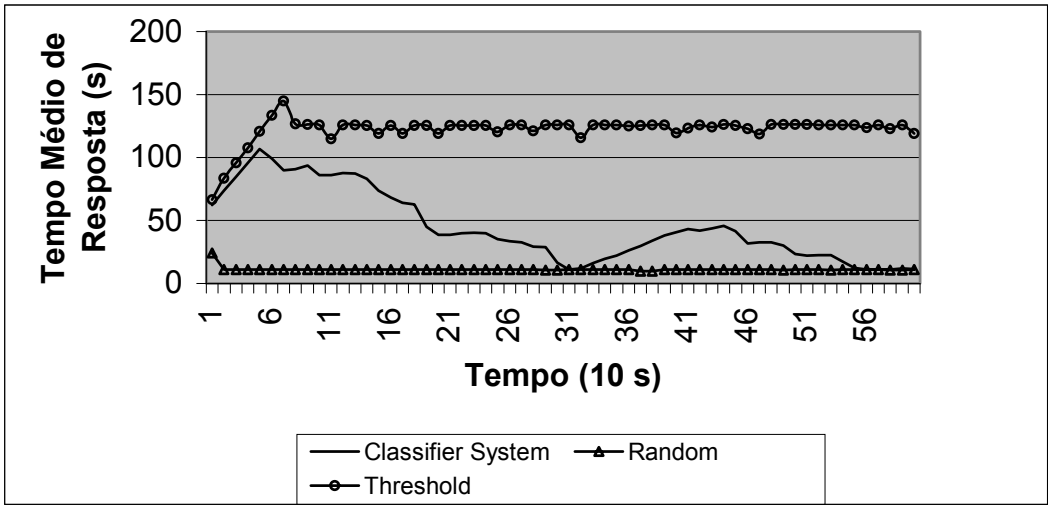

Figura 5.1: Resultados Obtidos na primeira bateria de testes

O gráfico da figura 5.1 demonstra pontos de baixo desempenho do sistema classificador seguidos de melhoras significativas. Estas discrepâncias no desempenho do método proposto podem ser explicadas pela natureza dos sistemas classificadores. Um sistema classificador deve "aprender" com o ambiente em que está inserido (GOLDBERG, 1989). Desta forma, nos intervalos de baixo desempenho são testados novos classificadores. Isto explica os intervalos de baixo desempenho seguidos de melhoras significativas. Devido à política de localização utilizada do método *random* e da natureza dos processos submetidos (granularidade fina) este apresentou o melhor desempenho dos três métodos. Apesar de ter obtido um desempenho inferior que o método *random* (o método random utilizado não respeita número máximo de processos que um nodo pode executar), é possível notar que o sistema classificador consegue o mesmo desempenho que o método random no final dos testes. Vale ainda ressaltar que o tempo de execução destes testes é pequeno. Logo, o sistema

classificador não consegue definir um conjunto de classificadores mais apropriado ao ambiente. Isto deve-se ao fato que poucas evoluções acontecem na população de classificadores.

O gráfico da figura 5.2 apresenta a média dos tempos médios de resposta obtidos na primeira bateria de testes.

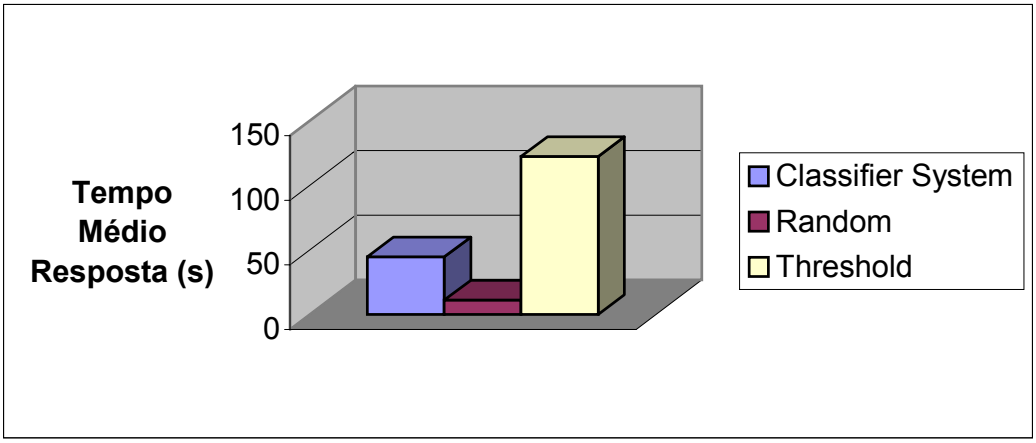

Figura 5.2: Média dos tempos médios de resposta obtidos no primeiro conjunto de testes

5.2.2 Segunda Bateria de Testes

Na segunda bateria de testes foram utilizadas seis **ThUser**. Foram executados 4 testes de 10 minutos e o gráfico 5.2 apresenta a média de cada ponto. O tempo médio de resposta do sistema foi extraído a cada 10 segundos. Os processos submetidos eram de tamanho variável, sendo o processo utilizado o primeiro descrito na seção 5.1. O gráfico da figura 5.3 apresenta o desempenho dos métodos *random*, *threshold* e sistema classificador.

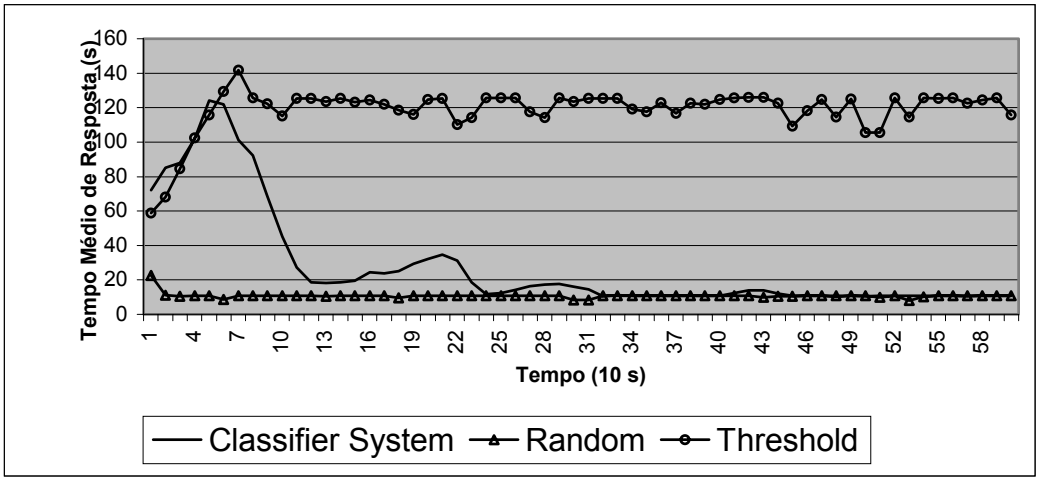

Figura 5.3: Resultados Obtidos na segunda bateria de testes

O objetivo deste conjunto de testes foi verificar se uma quantidade maior de ThUser e conseqüentemente um maior número de processos submetidos ao sistema iriam alterar os resultados obtidos na primeira bateria de testes. Não foi possível identificar uma mudança significativa nos resultados. O sistema classificador obteve melhores resultados que o método *threshold* e pior desempenho que o método *random*. Foram verificados ainda os mesmos intervalos de aprendizagem do sistema classificador que foram observados na primeira bateria de testes. O comportamento do sistema classificador foi semelhante, uma vez que depois de determinado ponto o mesmo equiparou-se ao método *random*. O gráfico da figura 5.4 apresenta a média dos tempos médios de resposta obtidos na segunda bateria de testes.

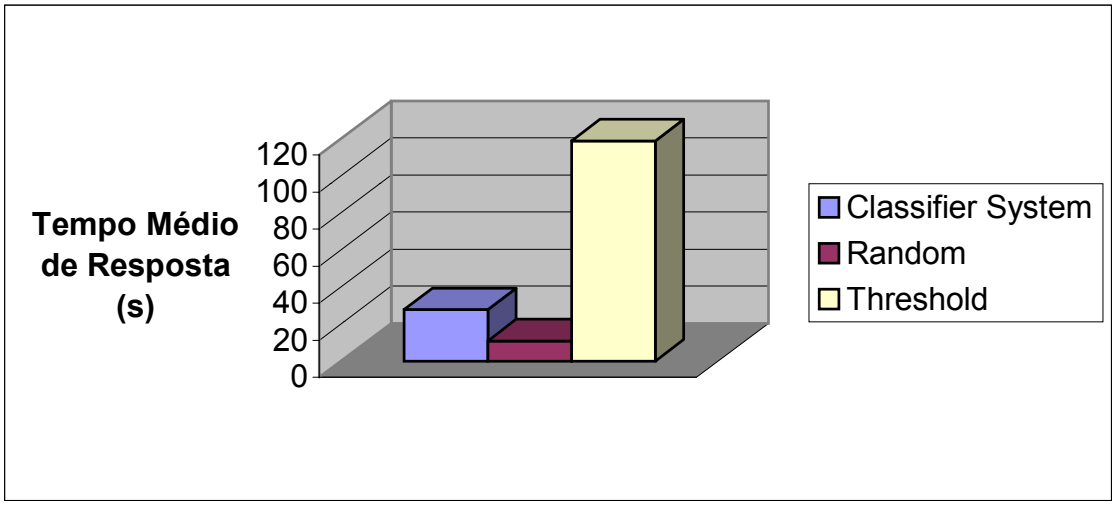

Figura 5.4: Média dos tempos médios de resposta obtidos no segundo conjunto de testes

5.2.3 Terceira Bateria de Testes

A terceira bateria de testes utilizou 4 **ThUser**. Foram executados 4 testes de 10 minutos e o gráfico 5.5 apresenta a média de cada ponto. O tempo médio de resposta do sistema foi extraído a cada 10 segundos. Os processos submetidos foram do cálculo dos números primos de 1 a 8000, 900 vezes pelo método crivo de Eratostenes. O gráfico da figura 5.5 apresenta o desempenho dos métodos *threshold* e sistema classificador. O método *random* teve que ser desconsiderado na terceira e quarta bateria de testes, pois onerou as máquinas de modo a causar falha na comunicação dos nodos escravos com o nodo mestre e a queda do sistema. Tal comportamento pode ser explicado pela política de localização utilizada por tal método que desconsidera o estado dos nodos escravos.

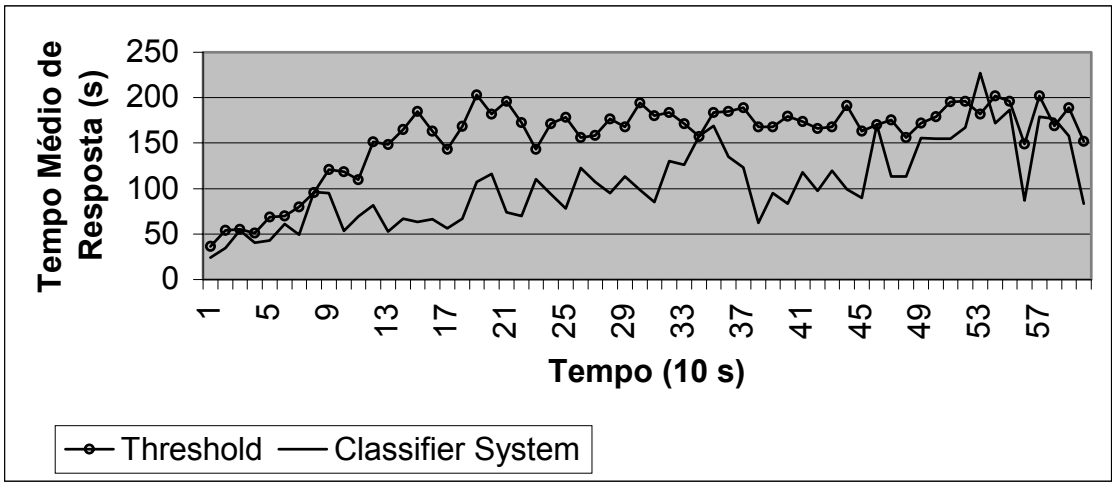

Figura 5.5: Resultados Obtidos na terceira bateria de testes

Apesar do tempo de execução do sistema ter sido insuficiente para uma correta evolução dos classificadores, o sistema classificador obteve melhores resultados que o método *threshold*. Assim como nas baterias de teste anteriores é possível identificar intervalos de "aprendizagem" do sistema classificador. Comparando os resultados desta bateria de testes com as anteriores nota-se uma maior dificuldade de adaptação do sistema classificador. Tal fato pode ser atribuído aos processos submetidos ao sistema. O tempo de execução é também insuficiente para uma melhor evolução dos classificadores. O gráfico da figura 5.6 apresenta a média dos tempos médios de resposta obtidos na terceira bateria de testes.

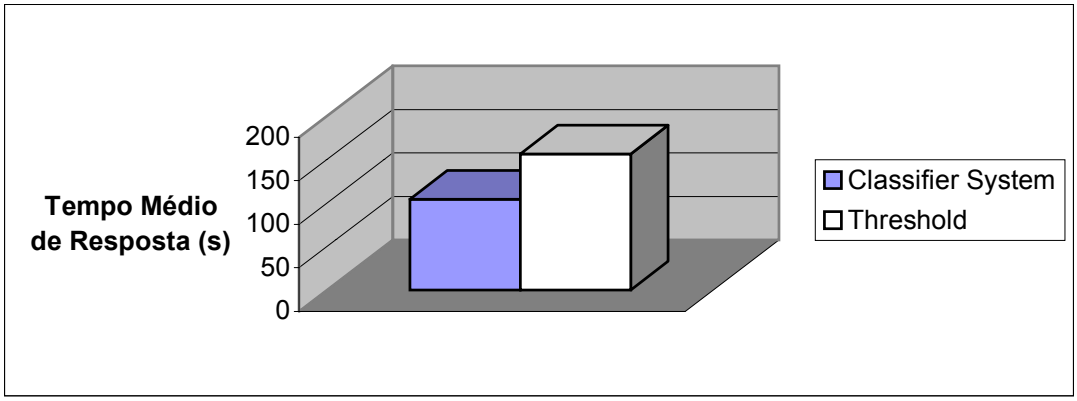

Figura 5.6: Média dos tempos médios de resposta obtidos no terceiro conjunto de testes

5.2.4 Quarta Bateria de Testes

A quarta bateria de testes utilizou 4 **ThUser**. Foram executados 2 testes de 20 minutos e o gráfico 5.7 apresenta a média de cada ponto. O tempo médio de resposta do sistema foi extraído a cada 10 segundos. Os processos submetidos foram do cálculo dos números primos sendo que o intervalo superior era variável de 1000 a 8000, executado 900 vezes pelo método crivo de Eratóstenes. Desta forma o método calculava os primos de 1 até um intervalo superior. O gráfico da figura 5.7 apresenta o desempenho dos métodos *threshold* e sistema classificador. O método *random* teve que ser desconsiderado pelo mesmo motivo explicitado na seção 5.2.3. Esta bateria de teste teve como objetivo principal observar o comportamento do sistema classificador mediante um tempo de execução maior.

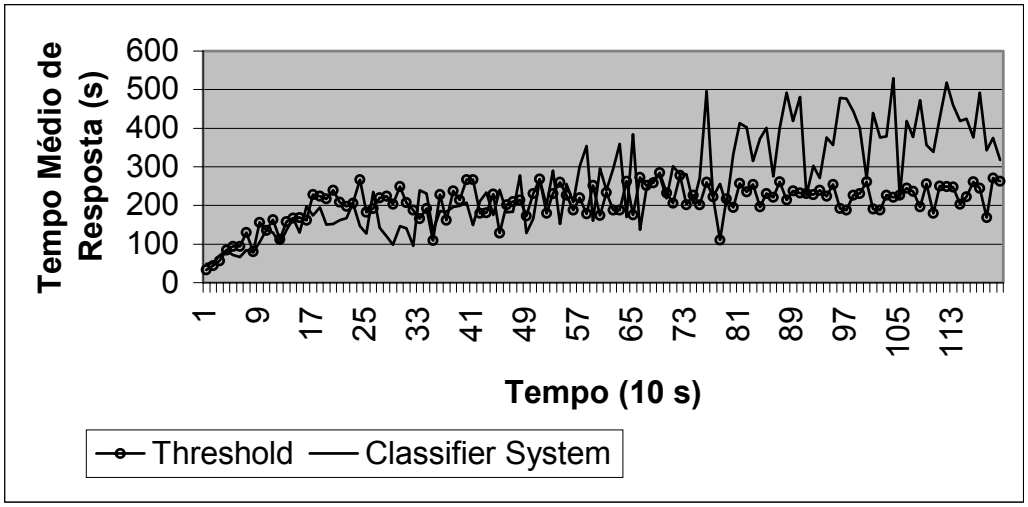

Figura 5.7: Resultados obtidos na quarta bateria de testes

O gráfico da figura 5.7 demonstra uma maior dificuldade do sistema classificador em adaptar-se ao ambiente. É possível notar um maior número de intervalos de "aprendizagem". Apesar disto, o sistema classificador obteve um melhor desempenho que o método *threshold* em alguns pontos da execução. O gráfico da figura 5.8 apresenta a média dos tempos médios de resposta obtidos na quarta bateria de testes.

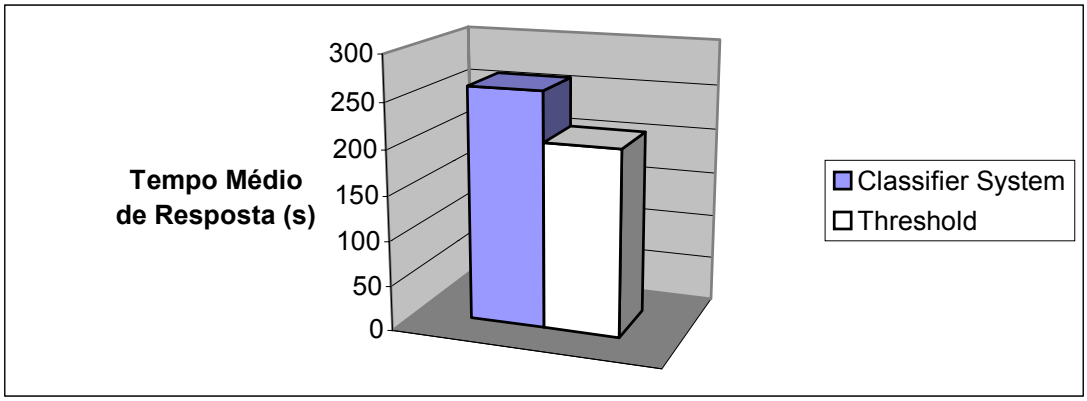

Figura 5.8: Média dos tempos médios de resposta obtidos no quarto conjunto de testes

5.3. Testes executados em Agregado de Computadores OSCAR

Os resultados experimentais apresentados nesta seção foram obtidos mediante execução do sistema proposto em agregado de Computadores OSCAR (PINTO et al,

2004a). O agregado é composto de 4 máquinas com processador Pentium IV 1,8Ghz com 256 Mb de memória principal, sendo que o nodo mestre possui 512 Mb de memória principal. O sistema operacional utilizado é Linux distribuição Red Hat 9.0. Foram executados testes de 20 horas. O intervalo de extração do tempo médio de resposta foi de 20 minutos. Comparamos os desempenhos do sistema classificador e do método threshold com valores 10 e 20.

O método *random* não foi testado pelas mesmas razões demonstradas nas seções anteriores. O objetivo principal desta bateria de testes foi verificar a adaptabilidade do sistema classificador em um agregado de computadores OSCAR. O tempo de execução foi aumentado drasticamente para verificar o desempenho do método e também a robustez do sistema como um todo. O processo utilizado foi o cálculo dos números primos de 1 a 8000, 900 vezes pelo crivo de Eratóstenes. O gráfico da figura 5.9 demonstra os resultados obtidos nesta bateria de testes. Vale ressaltar que os parâmetros utilizados neste conjunto de testes foram os mesmos especificados na tabela 5.1.

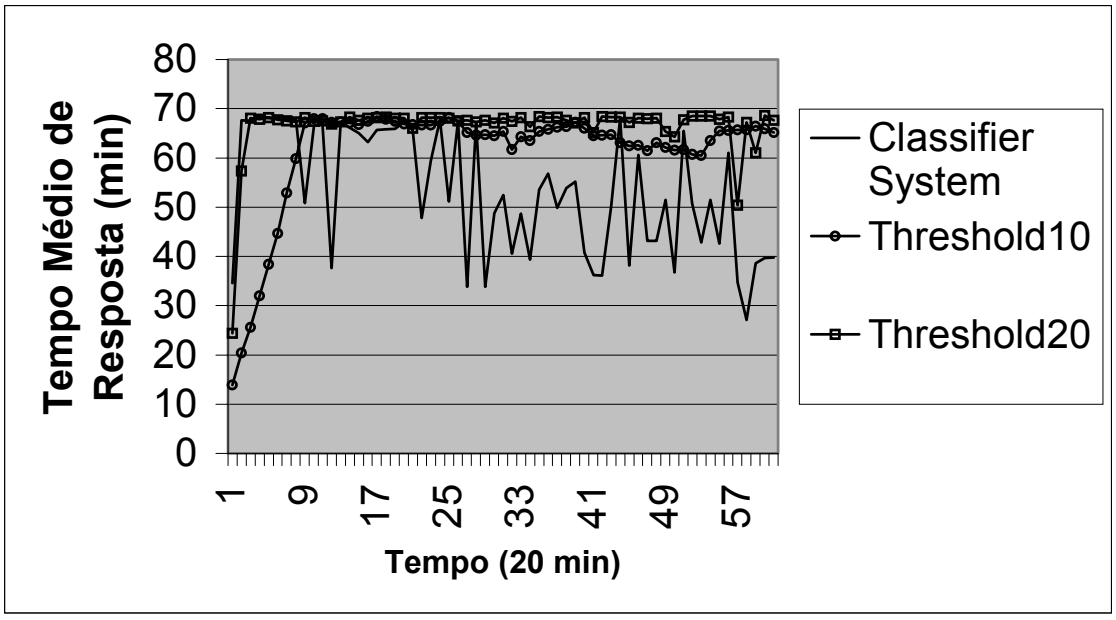

Figura 5.9: Resultados obtidos na quinta bateria de testes

O gráfico da figura 5.9 demonstra os mesmo intervalos de aprendizagem apresentados nos testes anteriores pelo sistema classificador. Uma vez que o tempo de execução foi maior que o dos outros testes, é possível fazer uma análise melhor do comportamento do método proposto. Nota-se também que o desempenho do sistema

classificador é melhor que o do método *threshold* em vários pontos do gráfico. Este experimento confirma ainda a dificuldade que o método *threshold* com valor fixo possui em aumentar o desempenho do sistema. O gráfico da figura 5.10 apresenta a média dos tempos médios de resposta obtidos na quinta bateria de testes.

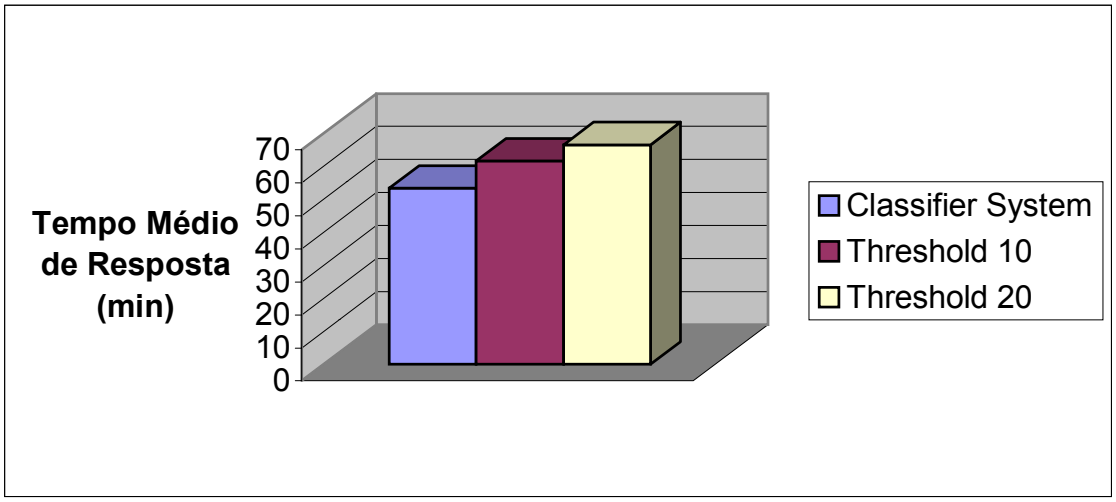

Figura 5.8: Média dos tempos médios de resposta obtidos no quinto conjunto de testes

6. Conclusões Finais e Trabalhos Futuros

A primeira etapa de desenvolvimento desta dissertação, foi a escolha de um método heurístico ou estocástico para resolver o problema do balanceamento de carga. Os estudos realizados apontavam para a utilização de algoritmos genéticos. A natureza adaptativa de tais métodos já justificariam sua utilização na resolução de um problema NP-Completo como o problema do balanceamento de carga. Apesar disto, a utilização de algoritmos genéticos acarreta em um tempo de evolução da população de resposta que inviabilizaria a sua utilização no balanceamento dinâmico de carga. Desta forma, foi escolhida a abordagem de sistemas classificadores pois combinam o método de algoritmos genéticos com a abordagem de algoritmos de aprendizagem de máquina. Além disso, a evolução da população de classificadores pode ser feita em paralelo às consultas aos classificadores, o que diminui o tempo de espera por uma resposta.

Logo após a escolha do método, devia-se optar por uma implementação que conseguisse proporcionar a extração de resultados experimentais. A primeira opção seria a utilização de um simulador de agregado de computadores. O uso de um simulador permitiria a execução de testes com várias configurações de agregados de computadores distintos. Apesar disto, com o uso de um simulador não seria possível ter o real sentimento das necessidades de um sistema real. Desta forma, optou-se por implementar um pacote de software para a execução dos testes. A implementação permitiu que fossem conhecidos todos os problemas que um módulo de balanceamento de carga baseado em sistemas classificadores possui.

O próximo passo foi a escolha de uma linguagem de programação. A primeira opção cogitada foi o uso da linguagem C. Tal linguagem é amplamente utilizada na área de alto desempenho. O problema da linguagem C é sua portabilidade, seria mais difícil testar um sistema implementado em C em uma grande quantidade de configurações de agregados de computadores. Desta forma, optamos pela linguagem Java. A linguagem Java possui como principal diferencial a portabilidade e as facilidades de *threads* e *sockets*. A implementação do pacote de software em Java permiti que se utilize computadores em rede como um agregado de computadores, sem a necessidade de alterar a configuração das máquinas. O único requisito é que

todas as máquinas utilizadas possuam a máquina virtual Java instalada. Desta forma, foi possível testar nosso método em sistemas operacionais Windows e Linux.

Os testes computacionais deveriam refletir os pontos fracos e fortes da abordagem utilizada. Optamos por dividir os testes em baterias, modificando alguns parâmetros a cada bateria. A idéia principal era testar a capacidade de aprendizagem e adaptabilidade do sistema classificador. O desempenho do método proposto deveria ser comparado com algoritmos clássicos de balanceamento de carga. Optamos por comparar o sistema classificador com os métodos *threshold* e *random*. Conseguimos desta forma, verificar a eficácia do sistema classificador na tomada de decisão em balanceamento de carga.

Na primeira e segunda bateria de testes utilizamos quatro e seis *threads* usuário respectivamente. Os usuários submetiam a cada 0,01 s um processo, o que gerou grande submissão de processos para o nodo mestre. Uma vez que o processo utilizado nestas baterias de teste eram de granularidade fina, quanto mais rápido fosse o envio dos processos para os nodos escravos, maior seria o desempenho do sistema como um todo. Devido às características do método *random*, este obteve melhores resultados que os outros métodos em ambas as baterias. O método *threshold* obteve o pior desempenho em ambas as baterias de teste. O sistema classificador obteve melhor desempenho que o método *threshold* e desempenho abaixo do obtido pelo método *random*. Apesar disto, é possível notar que o sistema classificador possui intervalos de aprendizagem (pontos de alto desempenho seguidos por pontos de baixo desempenho). O sistema classificador conseguiu "aprender" que o grande fluxo de processos de granularidade fina exigiam um grande intervalo (*threshold*). Esta afirmação pode ser confirmada, uma vez que o sistema classificador consegue equiparar-se ao método *random* no final dos testes. O tempo de execução é considerado pequeno para a correta evolução dos classificadores, o que contribuiu para que o método não conseguisse melhor desempenho.

Na terceira bateria de testes utilizamos um processo de granularidade média (cálculo dos números primos pelo crivo de Eratóstenes). Utilizamos quatro *thread* usuário e os mesmos parâmetros dos testes anteriores. O objetivo era verificar o comportamento do sistema classificador frente a um fluxo de processos diferente dos anteriores. Todos os processos eram iguais nesta bateria. O tempo de execução era o

mesmo dos testes anteriores. O método *random*, por não respeitar o estado dos nodos escravos, submetia grande quantidade de processos para estes. Tal comportamento causou a instabilidade do sistema, onerando as máquinas de forma a interromper a comunicação dos nodos escravos com o nodo mestre. Desta forma, o método *random* foi desconsiderado. O método *threshold* obteve pior resultado que o sistema classificador. O sistema classificador conseguiu adaptar o *threshold* e obter melhor desempenho. Os mesmos intervalos de "aprendizagem" foram observados.

O tempo de execução da quarta bateria de testes foi o dobro da terceira. O método *random* foi desconsiderado pelas mesmas razões antes explicitadas. Ao contrário da bateria anterior os processos eram variáveis. O sistema classificador teve mais dificuldades em adaptar-se a esta realidade obtendo desempenho inferior ao método *threshold*. Este fato pode ser justificado ainda pelo tempo insuficiente para a correta evolução dos classificadores.

Na quinta bateria de testes optamos por aumentar o tempo de execução drasticamente. Além disso, o pacote de software foi testado em um sistema operacional diferente (Linux). Desta forma, seria possível comprovar a portabilidade da implementação. Foi possível ainda testar a robustez do pacote de software devido ao tempo de execução elevado (vinte horas). Optamos por comparar o sistema classificador com o método *threshold* com valores 10 e 20. O sistema classificador obteve desempenho superior na maioria do tempo e na média geral. Apesar disto, foi possível notar o mesmo comportamento de aprendizagem já descrito anteriormente.

Os resultados experimentais apresentados nessa dissertação demonstram a grande capacidade de aprendizagem de sistemas classificadores quando submetidos ao problema do escalonamento dinâmico de processos. Tal fato pode ser comprovado pelos intervalos de baixo desempenho seguidos de intervalos de alto desempenho que o sistema classificador apresentou em todos os testes. A presente dissertação demonstra ainda a capacidade de utilização de diversos sistemas operacionais e diferentes configurações de agregados que o sistema implementado proporciona. Além disso, o sistema classificador proposto demonstrou ser capaz de adaptar-se a diversas realidades.

Como trabalhos futuros, indicamos a utilização de métricas diferentes da utilizada atualmente. Testes com os parâmetros utilizados pelo sistema classificador também

são necessários. O sistema deveria ainda ser testado mediante outros tipos de processo que não os apresentados neste trabalho. Além disso, configurações de agregados de computadores que utilizem diversos sistemas operacionais podem ser testadas a fim de comprovar a portabilidade do pacote de software desenvolvido. O sistema pode ainda ser facilmente estendido para receber processos de dispositivos móveis ou fixos. Desta forma, o sistema proposto poderia suprir problemas de baixo poder de processamento dos dispositivos móveis.

Referências Bibliográficas

(ARTSY & FINKEL 1989) Artsy, Y. , Finkel, R. "Designing a Process Migration Facility – The Charlotte Experience", IEEE Computer, vol. 22, 1989, pp. 725-737.

(BAUMGARTNER et al, 1995) Baumgartner, J., Cook, D.J., Shirazi, B. "Genetic Solutions to the Load Balancing Problem", in Proc. of the International Conference on Parallel Processing, 1995.

(CASAVANT & KHUL, 1988) Casavant,T.H. , Khul, J.G. "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems", IEEE Trans. On Software Eng., vol. l4, no. 2, Fevereiro 1988, pp. 141-154.

(CORRÊA & MELO, 2001) Corrêa, J.M. , Melo, A.C., "Using a Classifier System to Improve Dynamic Load Balancing", in Proc 30a-International Conference on Parallel Processing,, IEEE Press, Valencia, Espanha, 2001, pp. 411-416.

(CULLER & JASWINDER, 1999) Culler,D. , Jaswinder,P., Parallel Computer Architecture: A Hardware Software Approach, Morgan Kaufmann Publishers, 1999.

(DANTAS & ZALUSKA, 1998) Dantas,M.A.R. , Zaluska, E.J., "Efficient Scheduling of MPI Applications on Networks of Workstations" , Future Generation Computing Systems, 1998, pp 489-499.

(DANTAS et al, 2000)Dantas,M.A.R., Queiroz, W.J., Pfitscher, G.H., "An Efficient Threshold Approach on Distributed Workstation Clusters", in HPC in Simulation, Washington, USA, 2000, pp. 313-317.

(DANTAS, 2002)Dantas, M.A.R., Tecnologia de Redes de Comunicação e Computadores, Rio de Janeiro: Axcel Books, 2002.

(DEJONG, 1975) Dejong, K., "An Analysis of the Behaviour of a Class of Genetic Adaptative Systems", Dissertação de Doutorado, Universidade de Michigan, 1975.

(EAGER et al, 1986) Eager, D.L., Lazowska, E.D., Zahorjan, J., "Adaptative Load Sharing in Homogeneous Distributed Systems", IEEE Trans. On Software Eng., vol. 12, no. 5, Maio 1986, pp. 662-675.

(EAGER et al, 1986b) Eager, D.L., Lazowska, E.D., Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptative Load Sharing", Performance Evaluation, Vol. 6, no. 1, Março 1986, pp. 53-68.

(FERRARI & ZHOU, 1988) Ferrari, D., Zhou, S., "An Empirical Investigation of Load Indices for Load Balancing Applications", in Proc. Performance ´87, the 12th Int´l Symp. on Computer Performance Modeling, Measurement and Evaluation, Amsterdam, The Netherlands, 1988, pp. 515-528.

(GOLDBERG, 1989) Goldberg,D.E., Genetic Algorithm in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

(GREENE, 2001) Greene,W.A. "Dynamic Load-Balancing via a Genetic Algorithm", in Proc. of 13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01), Dallas, Texas, Nov., 2001.

(HOLLAND, 1984) Holland, J.H., "Genetic Algorithms and adaptation", in Proc. Of the NATO Advanced Research Institute on Adaptative Control of Ill-Defined Systems, 1984, pp. 317-333.

(HOU et al, 1994)Hou, E.S.H., Ansari, N., Ren, H., "A Genetic Algorithm for Multiprocessor Scheduling", IEEE Trans. On Parallel and Distributed Systems, vol. .5, no. 2, Fevereiro1994, pp. 113-120.

(KREMIEN & KRAMER, 1992) Kremien, O., Kramer, J., "Methodical Analysis of Adaptative load Sharing Algorithms", IEEE Trans. On Parallel and Distributed Systems, vol. 3, no. 6, , Nov. 1992 pp. 747-760.

(LIVNY & MELMAN, 1982) Livny, M. and Melman, M., "Load Balancing in Homogeneous Broadcast Distributed Systems", Proc. ACM Computer Network Performance Symp.,1982, pp. 47-55.

(LITZKOW et al, 1988) Litzkow, M.J., Linvy, M. , Mutka, M.W.,"Condor – A Hunter of Idle Workstations", Proc. IEEE 8th.Int. Conf. on Distributed Computing Systems, IEEE Computer Soc. Press, California, L.A, 1988, pp. 104-111.

(MAYA et al, 2004)Maya, Anu, Asmita, Snehal, Krushna (01 de abril de 2004) "MigShm: Shared memory over openMosix" A project report on MigShm [Online]. Disponivel em http://openmosix.sourceforge.net/#Documentation

(MITCHELL, 1996) Mitchell, M., An Introduction to Genetic Algorithm, MIT Press., 1996.

(MPICH, 2004) MPICH – A Portable MPI Implementation [Online]. http://www.mcs.gov/mpi/mpich. Disponível em 02/06/2004.

(PAPADIMITRIOU & STEILGLITZ, 1998) Papadimitriou,C., Steilglitz, K., Combinatorial Optimization: Algorithms and Complexity, Dover Publications, 1998.

(PINTO et al, 2004a)Pinto,A.R., Rista, C., Dantas, M.A.R.,"OSCAR: Um Gerenciador de Agregado para Ambiente Operacional Linux", in ERAD 2004: 4ª Escola Regional de Alto Desempenho, Pelotas, Brasil, Fevereiro, 2004, pp.193-196.

(PINTO & BORGES, 2004b)Pinto,A.R., Borges, P.S.S.," Comparação de métodos de seleção de reprodutores alternativos com o método da roleta ", in I WorkComp-Sul, Florianópolis, Brasil, Maio, 2004.

(PINTO & DANTAS, 2004c)Pinto,A. R., Dantas,M.A.R.,"Uma Abordagem de Balanceamento de Carga Baseada em Algoritmo de Aprendizado de Máquina Genético", in V WSCAD - Workshop de Computação de Alto Desempenho, Brasil, Foz do Iguaçu, 2004.

(POWELL & MILLER, 1983) Powell,M.L., Miller, B.P., "Process migration in DEMOS/MP", in Proc. 9th ACM Symp. Operat. Syst. Principles, 1983, pp.110-119.

(PVM, 2003) PVM – Parallel Virtual Machine (PVM). [Online] http://www.csm.ornl.gov/pvm. Disponível em 24/11/2003.

(SHIVARATRI et al, 1992) Shivaratri, N., Krueger, P., Singhal, M. "Load Distributing for Locally Distributed Systems", IEEE Computer, vol. 25, 1992, pp. 33-44.

(SMITH, 1988) Smith, J.M., "A survey of process migration mechanisms", ACM Operating Systems Rev ., vol 22, no 3, julho,1988, pp. 28-40.

(SRINIVAS & PATNAIK, 1994) Srinivas, M. , Patnaik,L.M.,"Genetic Algorithms: A Survey", IEEE Computer, vol. 27, 1994, pp. 28-43.

(THOMAS et al, 1995) Thomas,E., Anderson, D.E.C. e Patterson, D.A.,"A Case for NOW (Network of Workstations)", IEEE Micro, vol. 15, no 1, pp.54-64,1995.

(WANG & MORRIS, 1985) Wang, Y.T., Morris, R.J.T., "Load Sharing in Distributed Systems", IEEE Trans. On Computers., vol. 34, no. 3, Mar. 1985, pp. 204-217.

(WOO et al, 1997) Woo, S.H., Yang, S.B., Kim, S.D., Han, T.D. “Task scheduling in Distributed computing Systems with a genetic algorithm”, in Proc. of the High-Performance Computing on the Information Superhighway, HPC-Asia’97, 1997.

(ZHANG, 2004) Zhang ,W., (01 de abril de 2004) "Linux Virtual Server for Scalable Network Services" [Online]. Disponível em http://www.linuxvirtualserver.org/docs/scheduling.html

(ZHOU, 1988) Zhou,S., “A Trace-driven Simulation Study of Dynamic Load Balancing”, IEEE Trans. On Software Eng., vol. l4, no. 9, setembro1988, pp. 1327-1341.

(ZOMAYA et al, 1999) Zomaya, A.Y., Ward, C., Macey, B., “Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues”, IEEE Trans. On Parallel and Distributed Systems, vol l0, no. 8, Agosto1999, pp. 795-812.

(ZOMAYA & TEH, 2001) Zomaya, A.Y., Teh, Y.H., “Observations on Using Genetic Algorithms for Dynamic Load-Balancing”, IEEE Trans. On Parallel and Distributed Systems, vol. 12, no. 2, Sep.2001, pp. 899-911.