Servidor Genético: Uma abordagem de balanceamento de carga baseada em algoritmo de aprendizado de máquina genético para agregados de computadores

A.R. Pinto e M.A.R. Dantas
Departamento de Informática e Estatística(INE)
Universidade Federal de Santa Catarina(UFSC)
{arpinto,mario}@inf.ufsc.br

Resumo: *A utilização de agregados de computadores está cada vez mais presente no contexto computacional atual. Um dos grandes problemas de tais ambientes, muitas vezes, é a má utilização dos recursos computacionais. O módulo de escalonamento de processos é um importante componente para a melhoria de distribuição das cargas do sistema. Neste artigo, apresentamos uma abordagem de escalonamento dinâmico de processos baseada em sistemas classificadores. O Servidor Genético realiza a integração entre os clientes e o ambiente de agregado de computadores, utilizando sistemas classificadores para o balanceamento de carga. Sistemas classificadores são algoritmos de aprendizado de máquina, baseados em algoritmos genéticos, altamente adaptáveis. Em adição, apresentamos a implementação do pacote de software necessário para a execução dos testes, o qual é testado sob o paradigma de uma arquitetura mestre-escravo de agregados de computador. Nossos resultados experimentais demonstram um diferencial na capacidade de adaptação do sistema classificador mediante o ambiente sob qual está inserido.*
Palavras-chave: Algoritmo de Aprendizado de Máquina Genético, Agregados de Computadores, Balanceamento de Carga

Genetic Server: A Genetic Machine Learning Approach for Load Balancing in Cluster Computing Configurations

Abstract: *Cluster configurations are a cost effective scenarios which are becoming common options to enhance several classes of applications in many organizations. In this article, we present a research work to enhance the load balancing, on dedicated and non-dedicated cluster configurations, based on a genetic machine learning algorithm. Our approach is characterized by an on time assignment scheme using a classifier system. Classifier systems are learning machine algorithms, based on high adaptable genetic algorithms. We developed a software package which was designed to test the proposed scheme in a master-slave Cow (Cluster of Workstation) and Now (Network of Workstation) environment. Experimental results, from two different operating systems, indicate the enhanced capability of our load balancing approach to adapt in cluster configurations.*
Keyworks: Genetic Machine Learning Algorithm, Cluster Computing, Load Balancing

(Received June 7, 2005 / Accepted August 15, 2005)

1. Introdução

O crescente avanço das tecnologias de hardware e software, juntamente com a necessidade de desempenho computacional cada vez maior 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[13,14]. 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 [17], o OpenMosix[22] e o Linux Virtual Server[23].

Um dos grandes problemas em tais sistemas é o desenvolvimento de técnicas efetivas de distribuição de processos entre os nodos do agregado[6]. Segundo [4], em um agregado de computadores existe uma grande probabilidade de um nodo ficar sobrecarregado, enquanto outros ficam ociosos[4,14,18]. Tal problema, conhecido como desbalanceamento de carga, degrada a 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[15]. 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 [1,2,3,16,20,21,24]. Neste artigo apresentamos uma abordagem de escalonamento de processos através de sistemas classificadores [10,11,25]. 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.

O desempenho do sistema classificador proposto foi comparado com o método *threshold* e *random*. Os testes computacionais além de demonstrarem o bom desempenho do método proposto, indicam a capacidade de aprendizado do sistema classificador. Tais testes foram executados em um agregado de computadores OSCAR e em uma rede Windows, cujo sistema foi implementado na linguagem Java.

Este artigo é dividido da seguinte forma: a seção 2 apresenta uma revisão sobre as técnicas de escalonamento de processo e balanceamento de carga. Na seção 3 são introduzidos os conceitos de algoritmos genéticos e sistemas classificadores. Nosso ambiente de teste, juntamente com os detalhes da implementação do Servidor Genético são apresentados na seção 4. A seção 5 apresenta os testes computacionais executados. Finalizamos o artigo, na seção 6, com nossas conclusões e trabalhos futuros .

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

O problema de escalonamento de processos é uma das questões mais críticas na construção de um sistema distribuído[7,14]. Segundo [6], 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 resposta. 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.

A classificação de escalonamento apresentada em [6], divide o escalonamento global em estático e dinâmico. A utilização de escalonamento estático necessita de um prévio conhecimento do comportamento e das dependências dos módulos de um programa paralelo. Tal abordagem não leva em conta o estado atual do sistema. 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. Por outro lado, 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 [6], no escalonamento dinâmico nenhuma decisão é tomada antes que o processo seja criado no ambiente.

Sistemas distribuídos que implementam escalonamento dinâmico geralmente implementam a migração de processos. 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) [9]. Entretanto, é possível implementar uma abordagem menos rígida de escalonamento dinâmico chamada *on-time assignment(OTS)*[6]. A técnica *OTS* é considerada uma abordagem dinâmica, aonde a decisão do nodo aonde o processo vai ser executado é feita logo após a criação do processo e não considera migração preemptiva. Neste trabalho foi utilizada a técnica *OTS*.

3. Algoritmos Genéticos e Sistemas Classificadores

Algoritmos genéticos são algoritmos de busca, baseados nos mecanismos de seleção natural [10,12]. 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 pelo seu cromossomo (genótipo), que usualmente são representados por um conjunto de caracteres. A cada nova geração um novo conjunto de criaturas artificiais (caracteres) são gerados e 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.

O processo de evolução de um algoritmo genético começa com a definição de uma população inicial de possíveis respostas para o problema. 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 aptidão 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 ancestrais e podem ou não ter seu cromossomo mutado (através do operador de mutação). A população de respostas é então renovada, de acordo com um critério de reposição previamente escolhido e uma nova iteração é iniciada. O critério de parada da evolução pode ser por convergência (quando a aptidão média da população de soluções não aumenta a um certo número de gerações) ou por um número de gerações previamente definido.

3.1. Sistemas Classificadores

Sistemas classificadores são sistemas de aprendizado de máquina baseados em algoritmos genéticos, capazes de aprender regras sintaticamente simples chamadas classificadores, com o propósito de guiar seu desempenho em um ambiente arbitrário[10,11]. Um sistema classificador consiste de três componentes principais:

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

O sistema de regras e 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 correspondente a condição imposta pelo ambiente deverá ser efetuada. Os classificadores geralmente possuem um alfabeto ternário{0,1,#}, aonde # é considerado o símbolo *don't care*, ou seja o símbolo # pode representar 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 1:

Tabela 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.

Na criação do sistema classificador todos os classificadores possuem a mesma aptidão, quando um classificador é escolhido 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 o classificador que tiver o maior valor de aptidão e que conseqüentemente terá maior quantia a pagar ao sistema de divisão de créditos (parte de sua aptidão). Esta quantia será paga ao sistema classificador que gerou tal mensagem, se a ação que este executou no sistema tiver sido positiva. De tempos em tempos o algoritmo genético é ativado, e os classificadores são renovados.

4. Sistema Proposto: Servidor Genético

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[1,16,20,24]. 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 todos 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[13,14], aonde o nodo mestre recebe os processos e repassa estes para serem executados nos nodos escravos. Um esquema mais explicativo do sistema pode ser visto na figura 1.

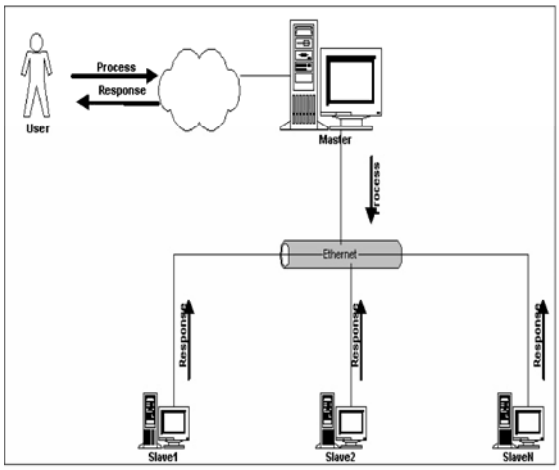

Figura 1. Arquitetura do sistema

Tal arquitetura permite que os processos sejam recebidos de clientes, sejam eles móveis ou fixos. Além disso, depois que o sistema for devidamente testado o escalonador poderá ser implementado em um ambiente de agregado que utilize arquitetura semelhante tais como OSCAR[17] e Linux Virtual Server[21]. Na tabela 2 descrevemos os ambientes computacionais utilizados nos testes.

Tabela 2: Descrição dos Ambientes Computacionais Utilizados nos Testes

Ambiente	Sistema Operacional	Processador	Memoria Principal
Now	Windows XP	AMD Duron 1.2 GHz	256 Mbytes
Cow	Red Hat Linux 9	Intel Pentium V 1.8 MHz	256 Mbytes

O nodo mestre é responsável por gerenciar todo o escalonamento do sistema, e ainda recebe o estado dos nodos escravos. Um esquema com as classes principais do módulo mestre é demonstrada na figura 2. 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, garantindo desta forma a escalabilidade do sistema. 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.

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 do nodo e enviar para os nodos. ThMasterSonStatus recebe o status dos nodos 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.

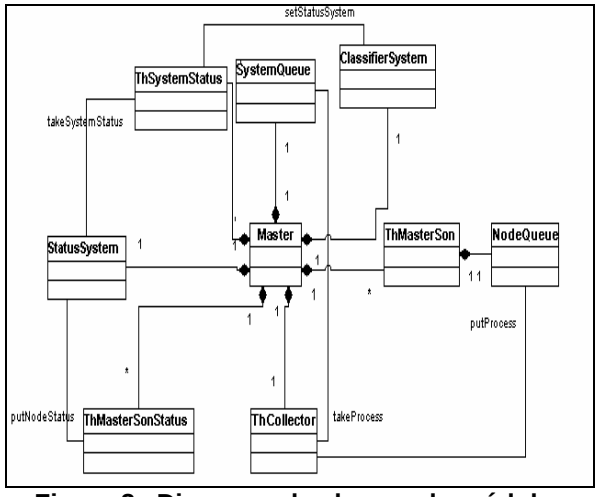

Figura 2. Diagrama de classes do módulo mestre

Cada nodo possui um NodeStatus, aonde é armazenado o status 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 de cada processo submetido o tempo médio de resposta é atualizado pela ThProc. A figura 3 demonstra a o diagrama de classes do módulo escravo.

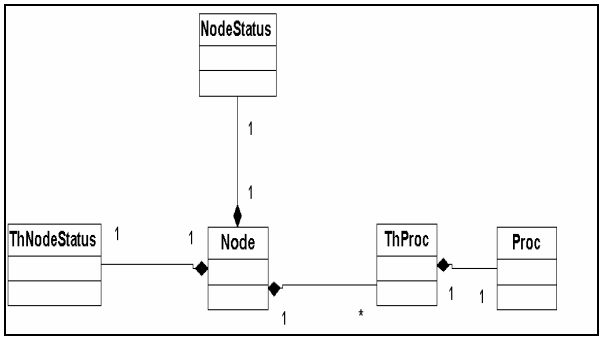

Figura 3. Diagrama de classes do módulo escravo

O sistema foi implementado na linguagem Java, devido a portabilidade oferecida pela mesma. Na figura 4 é apresentada a arquitetura formada pelo sistema apresentado com os ambientes NOW e COW.

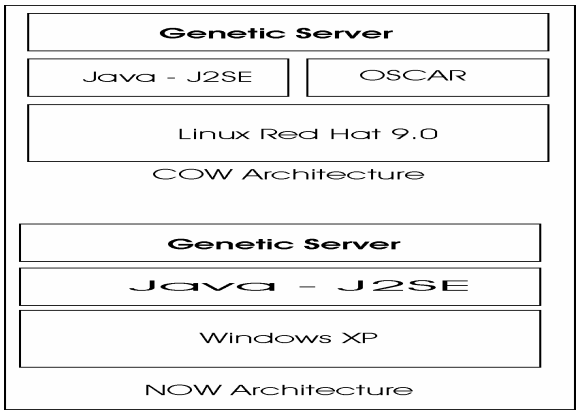

Figura 4. Arquitetura dos sistemas COW e NOW

4.1. 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* [7]. Uma vez que algoritmos genéticos geralmente geram um alto custo computacional, resolvemos utilizar algoritmo de aprendizado de máquina genético. Embora tais sistemas utilizem para o aprendizado do ambiente algoritmos genéticos, somente de tempos em tempos (numero de consultas ao sistema classificador) é que a população de classificadores é atualizada.

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 [8], 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[5, 8,18].

Segundo [8], 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 [18], 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.

Segundo [19], 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*. A configuração dos classificadores será a seguinte:

<Se houve aumento ou decremento>+<porcentagem de variação do tempo médio de resposta>

então

<Se o *threshold* deve ser incrementado ou decrementado>+<porcentagem de variação do *threshold*>

A tabela 3, demonstra as configurações da condição dos classificadores:

Tabela 3: 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, demonstra a configuração das ações dos classificadores:

Tabela 4: 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. De posse desta informação, a função de consulta codifica a condição de acordo com a tabela 3. 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 4 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 renovada 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 igual idêntica 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.

Se possuirmos os classificadores apresentados na tabela 5 em nosso sistema classificador:

Tabela 5: Exemplo de classificadores

Classficador	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

O desempenho do sistema classificador proposto foi comparado com o desempenho dos métodos *random* e *threshold* fixo. No método randômico um nodo destino é escolhido ao acaso e o processo é então enviado para o mesmo. No método de *threshold* fixo, um *threshold* é escolhido e os nodos recebem processos enquanto este número pré-determinado de processos não for atingido. A tabela 6 indica os parâmetros utilizados nos testes computacionais apresentados.

Tabela 6: Parâmetros dos Testes

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	10% da aptidão

Créditos	
Aptidão Inicial dos Classificadores	200
Probabilidade de mutação	1%
Probabilidade de crossover	100%
Taxa de reposição da população de classificadores	10%

Utilizamos nos testes das figuras 5 – 10, 1 nodo mestre e 3 nodos escravos. Sendo que o nodo mestre foi utilizado para operar como mestre e escravo. Todas as máquinas possuem 256 Megabytes de memória principal e processador AMD Duron 1,2 Ghz. Foram realizadas três baterias de testes, em todas as bateria foram utilizadas 4 ThUser. Os processos gerados pelas ThUser no primeiro e terceiro conjuntos de teste são de tamanho variável, desta forma acreditamos que nossa simulação ficará mais próxima de um sistema real. Os resultados apresentados na primeira e segunda bateria de testes representam a média de 4 testes de 10 minutos. Os resultados da terceira bateria de teste representam a media de 2 testes de 20 minutos. Utilizamos o tempo médio de resposta(*mean response time*) dos processos como métrica, desta forma quanto menor o tempo médio de resposta maior o desempenho do sistema.

Na segunda e terceira bateria de testes os processos submetidos executavam o cálculo dos números primos pelo método conhecido como Crivo de Eratostenes. Na segunda bateria de testes todos os processos submetidos calculavam os número primos até oito mil, novecentas vezes. Na terceira bateria de testes os processos gerados recebiam como entrada um número aleatório entre 1000 e 8000 e então calculavam todos os primos até este número recebido novecentas vezes.

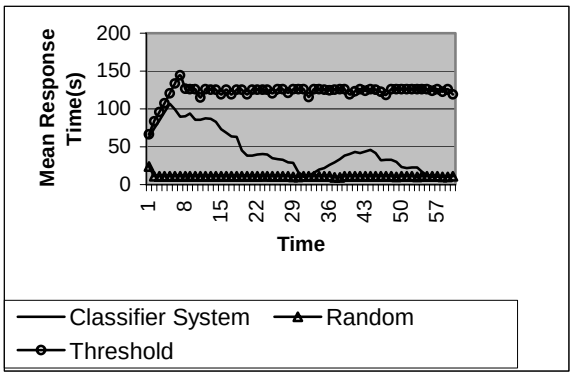

Figura 5. Resultados obtidos no primeiro conjunto de testes

Analisando o gráfico da figura 5, podemos observar um melhor desempenho do método *random*. Tal fato é decorrente principalmente da pequena quantidade de nodos e da granularidade fina dos processos, deste modo a escolha aleatória de um nodo é mais eficiente do que a utilização de uma política mais complexa. O método *threshold* obteve o pior resultado, isto deve-se principalmente a grande quantidade de processos que são gerados pelas *threads* de usuário. Uma vez que o *threshold* é fixo, a medida que o número de processos aumenta na fila do sistema o tempo médio de resposta aumenta. O gráfico da figura 5 demonstra ainda que o desempenho do sistema classificador proposto é inferior ao método randômico e superior ao *threshold*. No início da execução o sistema classificador obteve um aumento de tempo médio de resposta. Tal comportamento é decorrente da exploração de diferentes classificadores ocorrida no sistema classificador, até que o mesmo consiga escolher os mais adaptados ao ambiente. A medida que mais classificadores são testados, as escolhas do sistema classificador melhoram, aumentando assim o desempenho do sistema.

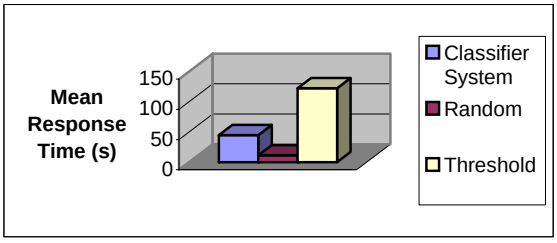

Figura 6: Médias dos tempos médios de resposta obtidos no primeiro conjunto de testes

O gráfico da figura 6 demonstra a média dos tempos médios de resposta. Os intervalos de aprendizado do sistema classificador contribuem para aumentar a média dos tempos médios de resposta. Apesar disto, o sistema classificador obteve uma média bem menor do que o método de *threshold* fixo.

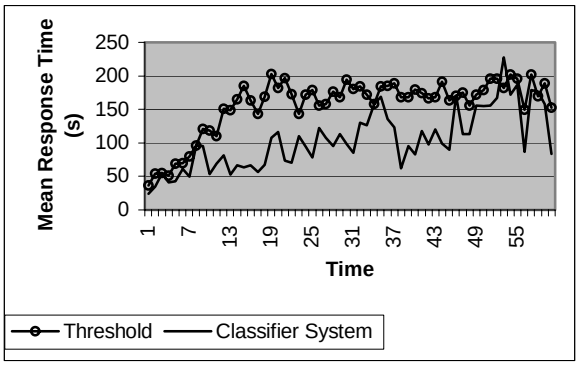

Figura 7: Resultados obtidos no segundo conjunto de testes

No segundo e terceiro conjuntos de teste, quando os processos submetidos possuiam uma granularidade grossa o método randômico onerou as máquinas de forma que seus resultados tiveram de ser desconsiderados. As máquinas ficaram sobrecarregadas a ponto de pararem de funcionar. Os gráficos das figuras 7 e 8 demonstram a superioridade do nosso método em comparação ao método *threshold* na terceira bateria de testes. Apesar disto é notável um grande número de intervalos de aprendizado, ou seja o sistema classificador teve mais dificuldades em se adaptar ao ambiente.

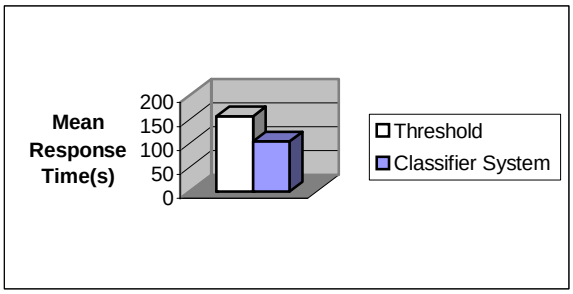

Figura 8: Médias dos tempos médios de resposta obtidos no segundo conjunto de testes

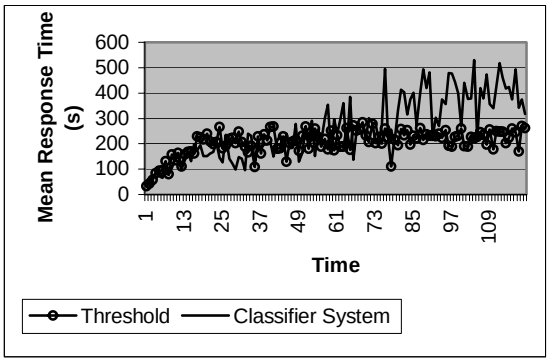

Figura 9: Resultados obtidos no terceiro conjunto de testes

No terceiro conjunto de testes o sistema classificador obteve um desempenho inferior ao método de *threshold*, como indicam os gráficos das figuras 9 e 10.

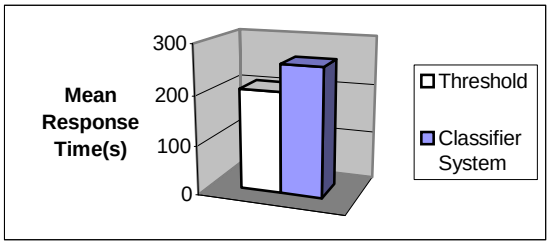

Figura 10: Médias dos tempos médios de resposta obtidos no terceiro conjunto de testes

Os testes apresentados nas figuras 11-13 foram realizados em um agregado de computadores OSCAR com 1 nodo mestre e 4 nodos escravo. O objetivo destes testes era comprovar primeiramente a portabilidade do sistema desenvolvido. Foi executado ainda um teste de robustez quando o sistema foi submetido a 60 horas de teste, sendo 20 horas para cada método. Estes resultados podem ser observados na figura 11 Os processos submetidos calculavam os números primos pelo método do crivo de eratostenes. Os resultados obtidos pelo sistema classificador foram comparados aos obtidos pelos método *threshold* com valores 10 e 20.

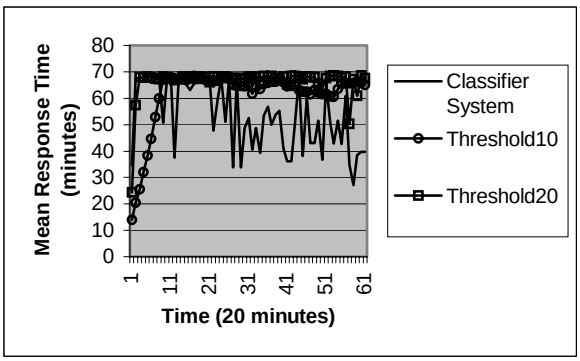

Figura 11: Resultados obtidos no teste de 20 horas

Nos resultados apresentados no gráfico da figura 11, o sistema classificador apresenta melhor desempenho e comportamento semelhante ao descrito anteriormente.

Nos próximos testes, apresentados nas figuras 12 e 13 utilizamos o método de eliminação de Gauss com matrizes de 256 posições (figura 12) e matrizes com tamanhos variáveis de 256 – 512 posições. O objetivo

era comprovar a eficiência do método proposto frente a um processo de natureza diferenciada das demais uma vez que este processo possui granularidade grossa[14].

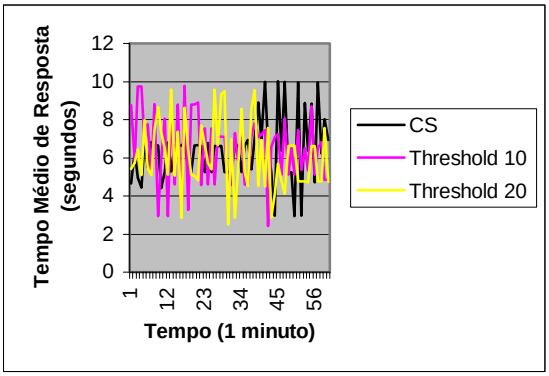

Figura 12: Resultados obtidos com método de Gauss, matriz 256

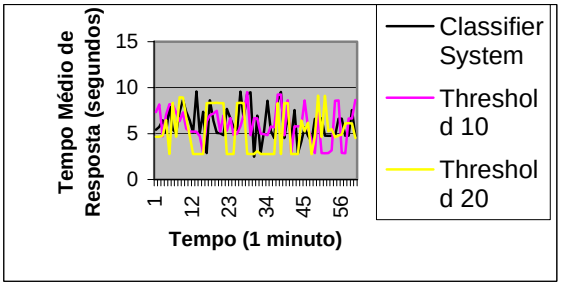

Figura 13: Resultados obtidos com método de Gauss, matriz 256-512

Apesar do sistema classificador não obter os melhores em todos os testes, é possível notar sua tendência em se adaptar aos mais variados tipos de ambiente. Logo, os resultados computacionais apresentados nesta seção demonstram a capacidade de aprendizado do sistema classificador e sua aptidão frente as mudanças do ambiente.

6. Conclusões e Trabalhos Futuros

Os resultados experimentais apresentados neste artigo demonstram a alta aptidão de um sistema classificador quando submetido a um sistema real. Em adição, a abordagem demonstra a capacidade de aprendizado do ambiente de agregado de computador, sob o qual o sistema classificador está inserido. Estes pontos podem ser observados através dos intervalos de baixo desempenho, seguidos por intervalos de alto desempenho apresentados nos gráficos. As degradações de desempenho apresentadas pela abordagem proposta são decorrentes do período de aprendizagem do sistema classificador, que testa novos classificadores com o intuito de otimizar os seus parâmetros. Os resultados demonstram ainda a alta capacidade de adaptação do sistema classificador proposto, robustez comprovada pelo teste de 20 horas ininterruptas. Comprovamos ainda a portabilidade do servidor genético uma vez que foram utilizados dois sistemas operacionais diferentes para os testes.

O principal ganho apresentado pelo algoritmo de aprendizado de máquina genético proposto foi a adaptabilidade do método threshold. Já que a escolha de um threshold apropriado é dependente de muitos fatores do ambiente, mesmo não obtendo o melhor desempenho, é possível obter uma maior adaptação do balanceamento de carga.

Como trabalhos futuros está sendo estudado um método de armazenar os classificadores. Desta forma, quando o sistema iniciar a execução já poderá contar com classificadores adaptados a sua realidade. Além disso a possibilidade de acesso através de computadores móveis.

7. Referências Bibliográficas

[1] A.Y. Zomaya, Y.H. Teh, "*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.

[2] E.S.H. Hou, N. Ansari, H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling", *IEEE Trans. On Parallel and Distributed Systems*, vol. .5, no. 2, Feb.1994, pp. 113-120.

[3] A.Y. Zomaya, C. Ward, B. Macey, "Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues", *IEEE Trans. On Parallel and Distributed Systems*, vol l0, no. 8, Aug.1999, pp. 795-812.

[4] S.Zhou, "A Trace-driven Simulation Study of Dynamic Load Balancing", *IEEE Trans. On Software Eng.*, vol. l4, no. 9, sep. 1988, pp. 1327-1341.

[5] O. Kremien, J. Kramer, "Methodical Analysis of Adaptative load Sharing Algorithms", *IEEE Trans. On Parallel and Distributed Systems*, vol. 3, no. 6, Nov. 1992, pp. 747-760.

[6] T.H. Casavant, J.G. Khul, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems", *IEEE Trans. On Software Eng.*, vol. l4, no. 2, Feb. 1988, pp. 141-154.

[7] Y.T.Wang, R.J.T. Morris, "Load Sharing in Distributed Systems", *IEEE Trans. On Computers.*, vol. 34, no. 3, Mar. 1985, pp. 204-217.

[8] D.L. Eager, E.D. Lazowska, J. Zahorjan, "Adaptative Load Sharing in Homogeneous Distributed Systems", *IEEE Trans. On Software Eng.*, vol. 12, no. 5, May 1986, pp. 662-675.

[9] M.L. Powell, B.P. Miller, "Process migration in DEMOS/MP", in *Proc. 9th ACM Symp. Operat. Syst. Principles*, 1983, pp.110-119.

[10] D.E. Goldberg, *Genetic Algorithm in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[11] J.H. Holland, “Genetic Algorithms and adaptation”, in Proc. Of the NATO Advanced Research Institute on Adaptative Control of Ill-Defined Systems, 1984, pp. 317-333.
[12] M. Mitchell, *An Introduction to Genetic Algorithm*, MIT Press., 1996.
[13] D. Culler, P. Jaswinder, *Parallel Computer Architecture: A Hardware Software Approach*, Morgan Kaufmann Publishers, 1999.
[14] M.A.R. Dantas, E.J. Zaluska, *Efficient Scheduling of MPI Applications on Networks of Workstations*, pp 489-499, Future Generation Computing Systems, 1998.
[15] C. Papadimitriou, K. Steilglitz, *Combinatorial Optimization: Algoritms and Complexity*, Dover Publications, 1998.
[16] J. M. Corrêa, A. C. Melo., “Using a Classifier System to Improve Dynamic Load Balancing”, *in Proc 30a-International Conference on Parallel Processing,, IEEE Press*, Valencia, Spain, Sep., 2001, pp. 411-416.
[17] A.R. Pinto, C.Rista, M.A.R. Dantas, “OSCAR: Um Gerenciador de Agregado para Ambiente Operacional Linux”, in *ERAD 2004: 4ª Escola Regional de Alto Desempenho*, Pelotas, Brasil, 2004, pp.193-196.
[18] M.A.R. Dantas, W.J. Queiroz, G.H. Pfitscher, “An Efficient Threshold Approach on Distributed Workstation Clusters”, in *HPC in Simulation*, Washington, USA, 2000, pp. 313-317.
[19] D.Ferrari, S.Zhou, “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.
[20] J.Baumgartner, D.J. Cook, B.Shirazi, “Genetic Solutions to the Load Balancing Problem”, in *Proc. of the International Conference on Parallel Processing*,pp.72-78 1995.
[21] S.H. Woo, S.B. Yang, S.D. Kim, T.D.Han, “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,pp.301-305.
[22] Maya, Anu, Asmita, Snehal, Krushna (April 1st 2004) "MigShm: Shared memory over openMosix" A project report on MigShm [Online]. Disponible in http://openmosix.sourceforge.net/#Documentation
[23] W. Zhang (April 1st 2004) "Linux Virtual Server for Scalable Network Services" [Online]. Disponible in http://www.linuxvirtualserver.org/docs/scheduling.html
[24] M. Munetomo, Y. Takai, and Y. Sato”A Genetic Approach to Dynamic Load-Balancing in a Distributed Computing System”,Proc.First Int’l Conf. Evolutionary Computation, IEEE World Congress Computational Intelligence, vol. 1, 1994, pp. 418-421.
[25] A.R. Pinto, M.A.R. Dantas, Uma abordagem de balanceamento de carga baseada em algoritmo de aprendizado de máquina genético”, V WSCAD, Foz do Iguaçu, Brasil, 2004.