ALOCAÇÃO DE TAREFAS EM UMA REDE HETEROGÊNEA DE COMPUTADORES UTILIZANDO TÉCNICAS EVOLUTIVAS

C. L. Castro e Silva, E. J. S. Oliveira, J. C. Mendes, O. R. Saavedra

Grupo de Sistemas de Energia Elétrica
Departamento de Engenharia de Eletricidade
Universidade Federal do Maranhão
São Luís - MA - 65085-580

Resumo— Muitas aplicações computacionais podem ser paralelizadas com objetivo de serem executadas mais rapidamente. Idealmente, o tempo de execução pode se reduzir proporcionalmente à quantidade de máquinas utilizadas, mas há o problema da alocação de tarefas nas máquinas da rede. Em um conjunto com máquinas de arquiteturas iguais, basta um simples algoritmo de balanceamento de carga para alocar as tarefas. Na prática é freqüente encontrar redes compostas por hardwares heterogêneos, onde o balanceamento de carga pode não ser a opção mais eficiente. Neste artigo é apresentado um estudo de alocação de tarefas em uma rede heterogênea, considerando três funções de mérito, a que são otimizadas através de uma estratégia evolutiva. Resultados comparativos das simulações são apresentados e confrontados com critérios gulosos clássicos.

Abstract— Several computer applications can be solved more quickly by using parallel processing. A virtual heterogeneous machine can be used to execute tasks more quickly than a set of homogeneous computers, because jobs can be allocated on machines particularly adequate to its requirements. This paper presents a study of the optimal task scheduling on the heterogeneous distributed environment. Three figures of merit have been implemented by using evolution strategies and compared with the classical greedy algorithms. Result from tests and relevant conclusions are also included.

Key Words— Alocação de Tarefas, Computação Distribuída, Estratégias Evolutivas

1 Introdução

Em (Pfister, 1998) o autor escreve que existem três formas de obter melhoras no desempenho: *trabalhar muito*, *trabalhar com "esperteza"*, ou *pedir ajuda*. Levando estas idéias ao contexto das tecnologias computacionais, elas podem ser interpretadas através das seguintes analogias (Jaen-Martinez, J., 2000): trabalhar muito equivale a utilizar hardware de alta velocidade, enquanto que trabalhar com "esperteza" está relacionado com a utilização de técnicas e algoritmos mais eficientes. Por ' último, "solicitar ajuda", se refere à utilização de processadores múltiplos para resolver uma tarefa. Isto é, a ajuda para resolver um problema mais rapidamente vem através da cooperação entre processadores. Durante os últimos anos, grandes esforcos têm sido dispendidos para aumentar o desempenho computacional, estando estas iniciativas distribuídas nessas três categorias.

A opção "solicitar ajuda" para resolver problemas em menores tempos tem sido ocupada por vários anos pelas chamados supercomputadores paralelos, que por razões de custo e outras restrições -algumas de ordem estratégica - ficaram acessíveis apenas para círculos restritos de usuários.

Desde os inícios dos anos 90, o rápido desenvolvimento de componentes de alto desempenho para computadores e de comunicações têm levado que as redes de computadores se tornem uma real alternativa de multiprocessamento frente aos supercomputadores, que possuem precos elevados, são especializados e proprietários. Neste contexto, os "clusters" são fácil de integrar-se dentro de redes existentes e podem ser expandidos facilmente aumentando memória e/ou processadores adicionais.

A disponibilidade de hardware e software não proprietário tem motivado que o processamento paralelo seja demitificado e que ele se torne acesível para um compêndio irrestrito de potenciais aplicações. Por outro lado, a alta taxa de renovação do hardware a custos altamente competitivos tem causado que as redes de computadores e *clusters* em geral não sejam formados por máquinas de características iguais. Em particular, no contexto das máquinas virtuais, aquelas formadas por hardware de arquitetura variada são chamadas de Máquinas Virtuais Heterogêneas - MVH. Uma aplicação paralelizada num conjunto de máquinas homogêneas precisaria somente de um simples algoritmo de balanceamento de carga para a alocação de suas tarefas de forma eficiente. Porém, em uma MVH, o mesmo algoritmo pode não ter um desempenho tão satisfatório, isto é, levar em contas apenas o balanceamento de carga pode significar que seja ignorada a eficiência relativa da máquina para resolver uma determinada tarefa (Fogel, D.B., Fogel, L.J., 1996). De fato, para um ambiente de máquinas heterogêneas, a alocação de tarefas em forma eficiente se torna um problema complexo e combinatorial. Folgel (Fogel, D.B., Fogel, L.J., 1996) propõe uma função

de mérito que quantifica o "tempo total para finalização dos processos", como critério a minimizar para obtenção de uma alocação de carga ótima. Em (Fogel, D.B., Fogel, L.J., 1996) o problema de alocação é definido como um problema de otimização que é resolvido através de um algoritmo de programação evolutiva (Fogel, D.B. 1995) (Fogel, D.B., 1994). Comtudo, a forma de como se deseja alocar essas tarefas nos recursos computacionais disponíveis pode variar de acordo a aplicação específica e também a qual aspecto se deseja priorizar. Mais ainda; o conceito de alocar cargas (tarefas) em uma rede heterogênea é válido para problemas que são decompostos para serem resolvidos através de processamento paralelo distribuído, bem como para processos que devem ser alocados em um conjunto de máquinas sem que isto implique necessariamente processamento distribuído. Assume-se que os tempos de processamento de cada tarefa em cada máquina disponível são conhecidos (i.e., alguma ferramenta - estática ou dinâmica- gerou esta informação).

Com o intuito de observar o impacto na distribuição de processos quando são levados em conta critérios diferentes, neste artigo é apresentado um estudo da alocação de tarefas em uma rede heterogênea. Foram implementados os algoritmos gulosos clássicos (*minmin* e *minmax*), bem como outras três funções de mérito. A primeira, corresponde à sugerida em (Fogel, D.B., Fogel, L.J., 1996), isto é, minimizar o tempo de conclusão do conjunto de tarefas, sendo formulado através da minimização do tempo do processo mais demorado. O segundo critério corresponde a minimizar a *soma dos tempos de execução das tarefas* e por fim, a terceira função objetiva o *balanceamento de carga*, através da minimização da diferença dos tempos de processamento de cada máquina. Neste trabalho, o problema de alocação é formulado como um problema de otimização e resolvido utilizando uma técnica evolutiva, conhecida como Estratégias Evolutivas. Neste contexto, as funções de mérito referidas acima assumem o papel de *medidas de desempenho* ou *fitness*, que definem a direção de busca em prol da configuração ótima. Este artigo está organizado como segue: primeiramente são revisados os algoritmos gulosos mencionados acima; em seguida é realizada uma rápida revisão das estratégias evolutivas e da formulação das funções de mérito propostas neste trabalho. Finalmente, são apresentados resultados comparativos com esses critérios incluindo a discussão dos mesmos e conclusões do trabalho.

2 Algoritmos Gulosos

Duas estratégias gulosas de alocação de tarefas muito populares são: o *minmax* (MMx) e o *minmin* (MM). Estes algoritmos são chamados assim, devido que em cada passo, a decisão é tomada visando máximo benefício imediato. O algoritmo *minmax* consiste numa técnica de alocação de 2 passos ou iterações. No primeiro passo, identifica-se, para cada tarefa, a máquina que leva o menor tempo para executá-la. Devem ser considerados vários fatores que possam influenciar o tempo de execução, por exemplo, o tempo gasto na comunicação entre as máquinas. Por exemplo, se uma máquina está com outra tarefa alocada, o tempo final de execução para uma nova tarefa deve ser igual a soma dos dois tempos.

O resultado do primeiro passo do algoritmo é uma lista da *melhor máquina*, que contém as máquinas com os tempos mínimos para executar cada tarefa. O segundo passo utiliza esta lista para alocar a tarefa com tempo *máximo* (tarefa mais demorada da lista) na melhor máquina, para ser executada primeiro. Em seguida, são repetidos os passos 1 e 2, atualizando a lista e alocando tarefas, até que todas tenham sido alocadas.

O algoritmo *minmin* é igual ao *minmax*, em termos do passo 1. A diferença se encontra no segundo passo, onde, é alocada primeiramente as tarefa com tempo mínimo da lista.

O algoritmos gulosos têm sido amplamente utilizados para alocação de tarefas em sistemas homogêneos. Porém, a sua aplicação para alocação de tarefas em redes heterogeneas pode levar a configurações de cargas sub-ótimas, especialmente quando a complexidade do problema aumenta.

A formulação de funções de mérito que representem critérios de alocação que enfatizam algum aspecto de interesse para o usuário se torna uma alternativa atraente. Isto significa que o problema deve ser formulado como um problema de otimização combinatorial discreto, onde a complexidade do mesmo cresce com o tamanho do sistema e número de tarefas a serem alocadas.

As técnicas evolutivas se tornam atraentes para resolver este tipo de problemas por não apresentar restrições quanto a complexidade, tipo de variáveis e dimensão do problema a tratar. A seguir é apresentada um breve revisão das Estratégias Evolutivas utilizadas neste trabalho.

3 Estratégias Evolutivas

Os primeiros trabalhos envolvendo Estratégias Evolutivas - EE's - surgiram na Alemanha na década de 60, com o objetivo de resolver problemas contínuos de otimização paramétrica sendo estendidas recentemente para o tratamento de problemas discretos. Nas EE's, um indivíduo é representado por um par de vetores reais da forma $v = (x, \sigma)$, onde x representa o ponto de busca no espaço e σ o vetor de desvio padrão associado. Nas versões atuais, a descendência é obtida

submetendo-se os indivíduos da geração a dois operadores: cruzamento e mutação. Observa-se que o parâmetro σ - que determina a mutação de x - também está sujeito ao processo de evolução. Esta é uma característica fundamental das EE's, que permite o auto-ajuste de seus parâmetros. Assumindo algumas hipóteses, é possível provar que as EE's convergem ao 'ótimo global com probabilidade 1, considerando um tempo de busca suficientemente longo.

As EE's multi-membros foram aperfeiçoadas tendo-se atualmente dois principais tipos: $(\mu + \lambda) - EE$ e $(\mu, \lambda) - EE$. Na primeira, μ indivíduos produzem λ descendentes, gerando-se uma população temporária de $(\mu + \lambda)$ indivíduos, de onde são escolhidos μ indivíduos para a próxima geração. Na versão $(\mu, \lambda) - EE$, μ indivíduos produzem λ descendentes, com $\mu < \lambda$, sendo que a nova população de μ indivíduos é formada por apenas indivíduos selecionados do conjunto de λ descendentes. Assim, o período de vida de cada indivíduo é limitado a apenas uma geração. Este tipo de estratégia tem bom desempenho em problemas onde o ponto ótimo é função do tempo, ou onde a função é afetada por ruído (Bäck, T., Hammel, U., Schwefel, H. P., 1996). Neste trabalho foi implementada esta última estratégia, motivada pelo fato de existirem poucos estudos envolvendo sua implementação.

4 Algoritmo Evolutivo

O algoritmo genérico para EE multi-indivíduo segue a seguinte estrutura básica:

t:=0;
inicializa P(t);
avalia P(t);
enquanto *(critério de parada não satisfeito)*
  P'(t):= recombinação [P(t)]
  P''(t):=mutação [P'(t)]
  avalia P''(t)
  se (μ, λ)*-EE então*
    P(t + 1):=seleção [P''(t)];
  senão
    P(t + 1):=seleção [P''(t) U P(t)];
  t:=t+1
fim enquanto

Neste algoritmo, $P(t)$ refere-se à população de μ indivíduos na geração t. Uma população de descendente $P''(t)$ de tamanho λ é gerada através de operadores de variação, no caso recombinação e mutação (porém outros operadores, tais como inversão são possíveis). Os descendentes são então avaliados e selecionados calculando-se os valores das suas respectivas fitness e levando o processo para melhores soluções.

Cada ponto nas EE's é um vetor $x \in R^n$, e o valor da fitness do indivíduo é o próprio valor da função aplicada neste ponto. Uma análise mais refinada permite explicar o algoritmo da maneira abaixo:

1. Gera-se a população inicial de μ indivíduos $P(t) = x_1(t), ..., x_\mu(t)$. Cada indivíduo v_ié representado por um par de vetores (x_i, σ_i), $\forall i \in 1, ..., \mu$.

2. Avalia-se o valor da fitness de cada indivíduo (x_i, σ_i), $\forall i \in 1, ..., \mu$ da população. Neste caso a fitness é dada pela própria função objetivo $f(x_i)$.

3. Escolhe-se aleatoriamente, mas de forma ponderada à fitness de cada indivíduo, dois elementos da população e efetua-se o cruzamento, que nesse estudo foi do tipo discreto, no qual dados os vetores v_1 e v_2, gera-se um vetor v:

$$v_1 = (x_1, \sigma_1) = ((x_1^1, ..., x_1^n), (\sigma_1^1, ..., \sigma_1^n))$$

$$v_2 = (x_2, \sigma_2) = ((x_2^1, ..., x_2^n), (\sigma_2^1, ..., \sigma_2^n))$$

   O vetor gerado pelo cruzamento será dado por

$$v = (x, \sigma) = ((x_q^1, ..., x_q^n), (\sigma_q^1, ..., \sigma_q^n))$$

   onde $q = 1$ ou 2, ou seja, cada componente provém de v_1 ou v_2.

4. Cada elemento gerado pelo cruzamento do passo anterior recebe uma mutação, gerando um indivíduo $v' = (x', \sigma')$, até que a quantidade λ de descendentes seja satisfeita. Esta mutação é dada por:

$$x_i'^j = x_i^j + \sigma_i^j N(0, 1) \qquad (1)$$

$$\sigma_i'^j = \sigma_i^j exp(\tau' N(0, 1) + \tau N_j(0, 1)) \qquad (2)$$

   onde $i = 1, ..., \lambda$ e $j = 1, ..., n$. $N(0, 1)$ representa um número Gaussiano com média zero e variância 1, que deve ser o mesmo para todas as posições do vetor. $N_j(0, 1)$ também representa um número gaussiano, entretanto este valor deve ser diferente para valor de j. Os fatores τ e τ' são sugeridos por Back como $\tau = (\sqrt[4]{4n})^{-1}$ e $\tau' = (\sqrt{2n})^{-1}$, respectivamente, onde n representa a ordem do problema (Bäck, T., Schwefel, H.P., 1993), (Bäck, T., Hammel, U., Schwefel, H. P., 1996), (Bäck T., Rudolph, G. and Schwefel, H.P., 1995).

5. Avalia-se a fitness de cada descendente (x_i', σ_i'), $\forall i \in 1, ..., \lambda$.

6. Os μ melhores indivíduos da população de descendentes são selecionados para formarem a nova população, $P(t+1)$. Volta-se para o passo 3, até que o critério de parada seja satisfeito.

No algoritmo descrito acima, os parâmetros σ também participam do processo evolutivo da mesma forma que as variáveis originais do problema. Este processo é referido como auto-adaptação de parâmetros (Bäck T., Rudolph, G. and Schwefel, H.P., 1995).

5 Implementação

Várias funções de mérito podem ser consideradas para a alocação das tarefas, dependendo do critério que se deseja otimizar. Com objetivo de estudar os impactos de decisões de alocação, neste trabalho são implementadas três funções de mérito ou fitness - para fins de comparação. Para todos os efeitos considera-se n_task tarefas para serem alocadas em n_maq máquinas disponíveis. O resultado obtido com estas funções é comparado com a alocação determinada pelos algoritmos gulosos *minmin* e *minmax*, designados MM e MMx respectivamente.

5.1 Critério M1 - Mínimo tempo de conclusão global

Este critério visa minimizar o tempo de execução do bloco de tarefas alocadas. Em outras palavras, isto equivale que o binômio processo - máquina mais demorado tenha seu tempo minimizado. Analiticamente, pode ser formulado assim:

$$Min \quad f = t_{max} \tag{3}$$

Onde t_{max} é o tempo do processo mais demorado para uma determinada distribuição de tarefas.

5.2 Critério M2 - Soma dos tempos

Este critério visa minimizar a soma dos tempos de execução de tarefas alocadas. Analiticamente, pode ser formulado assim:

$$Min \quad f = \sum_{i=1}^{n\_task} t_i \tag{4}$$

Onde t_i é o tempo que demora a tarefa i para ser executada em algum processador.

5.3 Critério M3 - Balanceamento de carga

Este critério visa o próprio balanceamento de carga. Este objetivo pode ser descrito assim:

$$Min \quad f = \sum_{i=1}^{n\_task} (t_{max} - t_i) \tag{5}$$

Onde t_{max} é o tempo de processamento mais demorado (ou processador mais carregado). Dessa forma o quanto menor sejam os desvios, mais balanceada estará a carga entre os processadores. No limite, (balanceamento total) o objetivo (5) vale zero.

6 Resultados de Testes

A seguir, são apresentados resultados desempenho comparativo dos algoritmos gulosos (MM e MMx) e dos critérios $M1$, $M2$ e $M3$, estes últimos otimizados através de estratégias evolutivas. O código do programa foi escrito em linguagem FORTRAN90 e executado em um computador Pentium III 650 MHz 128M de memória RAM. Foram considerados dois cenários ou estudo de caso denominados de $Caso$ 1 e $Caso$ 2, que são descritos a seguir:

$CASO$ 1: Corresponde a um estudo de caso ilustrativo, onde o número de tarefas a alocar foi menor do que as máquinas disponíveis, sendo $n\_task = 3$ e $n\_maq = 5$;

$CASO$ 2: Número de tarefas foi maior que o de máquinas alocadas, sendo $n\_task = 10$ e $n\_maq = 5$;

Assume-se que são conhecidas a priori os tempos de processamento de cada tarefa em cada máquina. Estas tarefas são consideradas independentes entre elas.

A Tabela (1) apresenta, para os casos 1 e 2, os tempos - em unidades de tempo de procesamento (UTP) - que cada tarefa consome em cada máquina. Por simplicidade, assume-se que nesse número já esta incluída a parcela de tempo devido a latências e outros custos de comunicação.

Casos	Tarefas	Máquina				
		1	2	3	4	5
1	1	7.0	3.0	11.0	8.0	5.0
	2	10.0	9.0	6.0	4.0	7.0
	3	8.0	7.0	4.0	10.0	3.0
2	1	7.0	3.0	11.0	8.0	5.0
	2	10.0	9.0	6.0	4.0	7.0
	3	8.0	7.0	4.0	10.0	3.0
	4	7.2	4.0	13.0	10.5	5.3
	5	1.0	9.4	6.0	3.0	8.0
	6	14.0	20.0	1.8	10.5	9.0
	7	3.0	3.0	2.5	8.1	8.0
	8	4.0	8.0	4.0	2.0	11.0
	9	12.0	7.9	4.8	11.5	3.5
	10	9.3	7.0	3.0	12.0	3.0

Tabela 1. Tempo de execução das tarefas para uma dada máquina.

6.1 *Parâmetros utilizados*

Devido que a simplicidade do $CASO$ 1, o tamanho da população utilizada foi $\mu = 20$, enquanto que a população de descendentes foi $\lambda = 60$. Os testes foram realizados executando um total de 100 gerações, para cada critério ($M1$, $M2$, $M3$).

O $CASO$ 2 avalia os critérios no algoritmo implementado em uma situação com características próximas as reais, com complexidade superior a do $CASO1$. O tamanho da população utilizada foi $\mu = 20$, enquanto que a população de descendentes foi $\lambda = 60$. Os testes foram realizados executando um total de 1500 gerações, para cada critério ($M1$, $M2$, $M3$). Em ambos casos, o critério de parada foi o número de gerações máximas consideradas.

Na Tabela 2 são apresentados resultados da alocação para o $CASO$ 1 considerando os algoritmos gulosos e as três funções de mérito apresentadas anteriormente. Nesta tabela, mostra-se o resultado da otimização de 3 tarefas que devem ser alocadas em 5 máquinas em função das estratégias de alocação (MM,MMx,$M1$, $M2$ e $M3$). A avaliação das estratégias é feita em função de uma determinada distribuição de tarefas.

Para um número reduzido de tarefas, em relação ao de máquinas ($CASO1$), pretende-se maximizar a utilização das máquinas, sendo que algumas ficaram disponíveis (sem alocação) a espera de novas tarefas. A Estratégia Evolutiva, como ferramenta de otimização do binômio tarefa-máquina, associa como população as tarefas em função do número de máquinas disponíveis. Neste sentido, na Tabela 2 apresenta-se a melhor configuração de distribuição de tarefas para cada um dos critérios de alocação avaliados. Note-se que neste caso, não houveram mais de 1 tarefa alocada num mesmo processador.

Na Tabela 3, os valores ótimos obtidos com cada função objetivo são mostrados na diagonal. Estas configurações ótimas foram avaliadas, para fins de comparação, com as outras duas funções objetivo, sendo esses valores registrados por fila, fora da diagonal, na mesma tabela.

Tarefas	MM	MMx	$M1$	$M2$	$M3$
1	2	2	2	2	1
2	4	4	4	4	5
3	5	5	3	5	2

Tabela 2. Alocações finais obtidas os algoritmos gulosos e com as três funções de mérito consideradas - Caso 1.

Observe-se que com o critério $M3$ tem-se uma configuração ótima onde o valor da função de mérito é igual a 0.0; isto é, tem-se um balanceamento total. Note-se também que esta configuração (100% balanceada) apresenta um valor para o objetivo $M1$ de 7 UTP, enquanto que

Critério	$M1$	$M2$	$M3$
$M1$	4.00	11.00	1.00
$M2$	4.00	10.00	2.00
$M3$	7.00	21.00	0 .00

Tabela 3. Comparação das alocações ótimas obtidas avaliadas com as funções de mérito restantes - Caso 1.

há outra configuração que minimiza este objetivo com valor de 4 UTP, porém com um balanceamento de 1. Aparentemente, o critério $M1$ é atraente, pois assegura a liberação de todas as máquinas em 4 UTP, mantendo um balanceamento de carga razoável.

Na tabela (4) são apresentados resultados de simulações para o $CASO$ 2. Neste caso, mais de uma tarefa foi alocada a um mesmo processador.

Na Tabela (5) são apresentados os valores das funções objetivo. Os elementos diagonais correspondem aos valores ótimos das respectivas funções para as configurações da tabela (4). Aqui, o desbalanceamento mínimo foi de 6.70, sendo que esta configuração avaliada nos critérios $M1$ e $M2$ tem fitness de $4,00$ e 33.30, respectivamente. As três configurações obtidas liberam todas as máquinas após 4 UTP (coluna 2, tabela (5)) . A diferença está no nível de balanceamento ($M3$) v-s tempo total de processamento ($M2$). Observando este último critério, há maior disponibilidade parcial de máquinas (para, por exemplo, assumir outros processos que estejam na fila) em menos tempo, porém o desbalanceamento é mais elevado (12.20). Por outro lado, privilegiar o objetivo $M3$ implica numa alocação balanceada, mais não ótima em termos de tempo computacional.

Como se pode apreciar, a alocação mais adequada deverá ser escolhida levando em conta as pecuriaridades do tipo de tarefa, utilização do sistema computacional e interesse do usuário em geral. Para tomar essa decisão, a disponibilidade de configurações ótimas obtidas levando em conta vários critérios fornece subsídios na decisão da escolha da forma de alocação mais adequada. Uma maneira alternativa de abordar o problema é a formulação de índices de desempenho que combinem os critérios abordados neste trabalho e outros que possam ser formulados. Tipicamente, uma combinação linear ponderada poderia ser adequada. Desta forma, a melhor alocação seria aquela que tivesse o melhor índice.

Na tabela (6) são reportados os tempos computacionais, em segundos, consumido pelo algoritmo evolutivo para obtenção das soluções ótimas.

7 Conclusões

Neste artigo foi apresentado um estudo da alocação de tarefas em uma rede heterogênea.

Tarefas	MM	MMx	$M1$	$M2$	$M3$
1	1	5	2	2	2
2	2	4	4	4	4
3	5	3	3	5	3
4	5	2	2	2	2
5	1	4	4	1	4
6	3	2	3	3	3
7	2	1	2	3	1
8	4	1	1	4	1
9	4	5	5	5	5
10	3	3	5	5	3

Tabela 4. Resultados da alocação ótima para cada critério considerado - Caso 2.

Critério	$M1$	$M2$	$M3$
$M1$	4.00	33.30	6.70
$M2$	4.00	27.80	12.20
$M3$	4.00	33.30	6.70

Tabela 5. Comparação das alocações ótimas obtidas avaliadas com as funções de mérito restantes - Caso 2.

Foram implementados os algoritmos gulosos (*min-min* e *minmax*), bem como outras funções de mérito tais como *soma dos tempos de execução das tarefas* e *Balanceamento de carga*. O problema de alocação é colocado como um problema de otimização e resolvido através de Estratégias Evolutivas. O efeito dos vários critérios considerados foi discutido no que se refere à alocação de tarefas obtida, verificando-se que dependendo do caso, um critério pode ser mais atraente do que outros, e em alguns casos, uma solução atende mais de um critério ótimo.

Para fins de testes foram assumidas várias hipóteses simplificadoras, tal como independência entre tarefas. Correntemente, trabalha-se com a modelagem de tarefas interdependentes, cujos resultados serão reportados em trabalhos futuros.

Agradecimentos

Os autores desejam expressar seus agradecimentos ao Conselho Nacional de Desenvolvimento Tecnológico, CNPq e ao Banco do Nordeste do Brasil - BNB, que apoiaram o desenvolvimento deste trabalho.

Casos	Critérios			
	MMx	$M1$	$M2$	MM
1	0,10	0,10	0,10	0,10
2	4,07	4,06	4,16	4,14

Tabela 6. Tempo de execução (em segundos) do algoritmo evolutivo para obtenção das soluções ótimas.

Referências Bibliográficas

Pfister, G.(1998), "In Search of Clusters", 2nd Ed. Prentice-Hall, Englewood Cliffs, NJ.

Jaen-Martinez, J. (2000), "The Java Management Extensions (JMX): Is Your Cluster Ready for Evolution?", Journal of Parallel and Distributed Computing, Vol 60, pp. 1341-1353, Academic Press.

Fogel, D.B., Fogel, L.J. (1996),"Using Evolutionary Programming to Schedule Tasks on a Suite of Heterogeneous Computers", Computer Ops. Res., Vol 23, No 6, pp. 527-534.

Fogel, D.B. (1995), "A Comparison of Evolutionary Programming and Genetic Algorithms on Selected Constrained Optimization Problems", Simulation, pp 397-404.

Fogel, D.B. (1992), "Evolutionary Optimization", Conference Record of the XXVI ASILOMAR Conference on Signal, System and Computers, IEEE Comput. Soc. Press, pp 409-414, Pacific Grove, CA, USA, 26-28.

Fogel, D.B. (1994), "An Introduction to Simulated Evolutionary Optimization", IEEE Transactions on Neural Networks, vol 5 No 1, pp 3-14.

Bäck, T., Schwefel, H.P. (1993), "An Overview of Evolutionary Algorithms for Parameter Optimization", Evolutionary Computation, pp. 1-27.

Bäck, T., Hammel, U., Schwefel, H. P. (1996) "Evolutionary Computation: An Overview", in Proc. 3rd IEEE Conf. on Evolutionary Computation, Piscataway, NJ: IEEE Press, pp. 20-29.

Bäck T., Rudolph, G. and Schwefel, H.P. (1995), "Evolutionary Programming and Evolution Strategies: Similarities and Differences", Proc. of European Conf. on Alife, Granada, Spain.