Metaheurísticas na Alocação de Tarefas em Sistemas Distribuídos de Tempo Real

A.M. Barroso, J.R.A. Torreão, J.C.B. Leite e O.G. Loques
CAA - P.G. em Computação Aplicada e Automação
e Deptº de Engenharia Elétrica
Universidade Federal Fluminense
{melon, jrat, julius, loques}@caa.uff.br

J.S. Fraga
Lab. de Controle e Microinformática
LCMI-EEL
Universidade Federal de Santa Catarina
fraga@lcmi.ufsc.br

Resumo

A alocação de tarefas a processadores em um sistema distribuído é conhecida como um problema NP-árduo. Nesse sentido, na busca de soluções que executem em tempo polinomial, heurísticas têm sido propostas. Esse trabalho compara o uso de duas dessas heurísticas na alocação de tarefas com requisitos temporais críticos em sistemas distribuídos de tempo real: o recozimento simulado (*simulated annealing*) e a otimização microcanônica.

Abstract

Task allocation to processors in a distributed system is known to be an NP-hard problem. In search of solutions that execute in polynomial time, heuristics have been proposed. This paper compares the use of two heuristics for scheduling hard real-time tasks in a distributed system: simulated annealing and microcanonical optimization.

1. Introdução

Sistemas distribuídos podem ser compreendidos como uma coleção de processos, possivelmente cooperativos, executando em diversos espaços de endereçamento. Em particular, sistemas distribuídos têm sido utilizados em várias aplicações de tempo real, em diversas áreas de interesse econômico ou científico. Podemos encontrá-los nos centros de supervisão e gerenciamento de energia, linhas de montagem robotizadas, sistemas embarcados, entre outros. Cada um dos processos em um sistema distribuído pode gerar uma ou mais tarefas, cada uma delas com graus de criticalidade distintos e diferentes demandas de processamento temporal. Essas tarefas podem ser classificadas como periódicas ou aperiódicas, havendo diferentes formas de tratar cada uma delas para efeito de escalonamento [Auds 94].

O tratamento de tarefas periódicas é feito, normalmente, antes da operação (*off-line*), já que todos os seus parâmetros são conhecidos. Assim, para essas tarefas, dois problemas podem ser visualizados: (i) para o conjunto de todas as tarefas a serem executadas e para o conjunto de todos os processadores, qual a melhor alocação tarefa-processador; (ii) dado um grupo de tarefas a serem executadas em um determinado processador (ou nó) de um sistema distribuído, como garantir um escalonamento que atenda a todas as restrições (tempo real , ou não) desse grupo. De fato, esses dois problemas são interrelacionados: uma tarefa que não tenha a garantia de ter suas restrições respeitadas em um processador pode tê-las em outro. As restrições a serem atendidas podem ser de vários tipos. Por exemplo, se o sistema deve ser tolerante a falhas, réplicas de tarefas críticas devem existir e, necessariamente, devem ser executadas em processadores distintos. Além disso, tarefas podem exigir recursos especiais só disponíveis em determinados processadores. Várias outras restrições poderiam ser citadas. Em suma, o problema reside em, dado um conjunto de tarefas, $T = \{t_1, t_2, \ldots, t_n\}$, e um grupo de processadores, $U = \{u_1, u_2, \ldots, u_m\}$, obter uma alocação, $A = \{(t_a, u_1), (t_b, u_2), \ldots, (t_x, u_m)\}$, que atenda a um conjunto de restrições $R = \{r_1, r_2, \ldots, r_p\}$.

No caso das tarefas aperiódicas, o escalonamento local pode ser feito através do recurso a uma tarefa servidora periódica (com período menor que o menor intervalo de tempo de ativação da tarefa aperiódica) ou ser feito utilizando-se a capacidade não utilizada pelas tarefas periódicas. Nesse caso, alguns dos nós podem temporariamente ficar sobrecarregados enquanto outros podem estar subutilizados ou mesmo ociosos. Faz-se então

necessária alguma política de divisão de tarefas (migração) entre os nós. Contudo, uma tal política em sistemas de tempo real tem objetivos distintos daqueles de uma política de escalonamento para um sistema de uso geral. Enquanto nestes o objetivo é garantir um bom balanceamento de carga de forma a minimizar o tempo médio de resposta para cada tarefa, nos sistemas de tempo real o objetivo é garantir a execução de cada tarefa dentro de seus limites temporais. Assim, outras formas de alocação que não o balanceamento devem ser perseguidas. Claramente, a existência de diferentes capacidades de reserva temporal em cada um dos nós do sistema, ou seja, a existência de um perfil de cargas, é desejável [Best 96]. Esse perfil pode ser também perseguido como uma das restrições para a alocação das tarefas periódicas. Nesse trabalho, admitiremos somente a existência de tarefas periódicas. A extensão para o caso das tarefas aperiódicas está sendo realizada em um trabalho complementar [Barr 97].

É sabido que o problema da alocação ótima de um número T de tarefas em um número P de processadores é do tipo NP-árduo. Dessa forma, heurísticas têm sido usadas para tratá-lo de forma prática. Uma dessas heurísticas, o recozimento simulado, tem sido exaustivamente estudada, pela sua característica de ser apropriada a uma variedade de aplicações, além de poder obter soluções arbitrariamente próximas a um ótimo. Contudo, para obter soluções de qualidade, ela exige um grande esforço computacional. Assim, vários métodos computacionais, como a paralelização, têm sido empregados para torná-la mais eficiente. Além disso, variações baseadas em busca local (ou em busca na vizinhança, ou ainda em melhorias iterativas) têm sido utilizadas no intuito de obter melhores (mais rápidas) soluções.

Nesse trabalho, apresentaremos o uso de uma nova heurística, a otimização microcanônica, na solução do problema de alocação ótima de tarefas em sistemas distribuídos. De forma a poder visualizar os seus benefícios, uma comparação é feita com uma solução utilizando o recozimento simulado [Tind 92]. Na seqüência, apresentaremos, na seção 2, uma introdução às técnicas de recozimento simulado e otimização microcanônica e sua aplicação ao problema de alocação de tarefas em sistemas distribuídos. Na seção 3, o modelo usado para comparação entre as técnicas é apresentado. Na seção 4, alguns resultados numéricos das simulações são fornecidos e discutidos. Finalmente, na seção 5, são apresentadas as conclusões.

2. Metaheurísticas e a Alocação de Tarefas em Sistemas Distribuídos

Nessa seção iremos apresentar as principais características de duas heurísticas aplicáveis ao escalonamento de tarefas em sistemas distribuídos. A primeira delas, o recozimento simulado, é conhecida e tem sido utilizada na solução de diversos problemas. Seu maior problema é o tempo de processamento gasto na obtenção de uma solução. A segunda heurística, a otimização microcanônica, foi recentemente proposta [Torr 95].

2.1. Recozimento Simulado

O método de recozimento simulado é originário da área de processos de fabricação metalúrgicos. Neste tipo de fabricação, um material é aquecido até altas temperaturas, um estado de alta energia, sendo após esfriado lentamente (ou seja, é levado a um estado de baixa energia). Nessa sucessão de estados, o material produzido fica livre de tensões. A mecânica estatística descreve esse processo através da distribuição de Boltzmann. No caso de problemas de otimização, o método consiste em obter um mínimo (ou máximo) dessa distribuição [Aart 89].

Seja uma configuração descrita por uma distribuição Ω, que inclui toda a informação de estado. No caso do problema de escalonamento de tarefas, esta configuração pode descrever as possíveis alocações de tarefas entre os diversos processadores (ou entre membros de um grupo de processadores). Seja a função $E(\Omega)$, a energia ou custo desta configuração. No problema, a energia é uma função do atendimento às restrições de otimização estabelecidas. Por exemplo, para efeito de tolerância a falhas, uma configuração que tenha duas réplicas de uma tarefa alocadas ao mesmo nó terá mais alta energia que uma configuração similar mas onde essas réplicas estejam em nós distintos. A probabilidade de ocorrência da configuração Ω, pode ser expressa pela distribuição de Boltzmann

$$P(\Omega) \equiv Ke^{\frac{-E(\Omega)}{T}}$$

onde T é a temperatura do sistema (um parâmetro de controle no problema de otimização) e K é o fator de normalização, obtido como abaixo indicado

$$K = \left[\sum_{\Omega} e^{\frac{-E(\Omega)}{T}} \right]^{-1}$$

O método inicia-se partindo de uma configuração arbitrária e novas configurações são propostas, para as quais avalia-se a função energia (ou o custo). Dadas duas configurações Ω e Ω^*, vale a relação

$$P(\Omega^*) = P(\Omega) e^{-\frac{E(\Omega^*)-E(\Omega)}{T}}$$

que pode ser reescrita como

$$P_s = e^{\frac{-\Delta E}{T}}$$

Na implementação do recozimento simulado, se $E(\Omega^*) < E(\Omega)$, Ω^* é aceita como nova configuração. Em caso contrário, um número aleatório é gerado no intervalo [0, 1] e comparado com P_s. Se P_s for menor, Ω^* é aceita como nova configuração, caso contrário, a configuração Ω é mantida e uma nova tentativa Ω^* é gerada. Quando o equilíbrio é alcançado, o processo é repetido com T sendo gradualmente reduzida até um mínimo determinado no plano de recozimento (*annealing schedule*) [Aart 89]. Para T constante, pode-se mostrar que esse processo embute uma cadeia de Markov que converge para uma distribuição estacionária de probabilidades associada às configurações possíveis, quando o número de iterações tende a infinito. Esse resultado continua válido mesmo no caso de T decrescente (cadeia não homogênea), desde que certas condições (normalmente atendidas) para a série de valores de T sejam seguidas. No limite, em um número de iterações infinito, o processo converge com probabilidade um para o mínimo global [Aart 89, DiNa 95]. Claramente, são as implementações aproximadas, que fornecem solução em tempo polinomial, que tornam o método atrativo. A escolha adequada do plano de recozimento é fundamental para o sistema atingir o mínimo global (ou próximo a ele) em um número finito de iterações. Normalmente, planos baseados no número de iterações são utilizados.

2.2. Otimização Microcanônica

A metaheurística de otimização microcanônica é composta de duas etapas [Linh 96, Torr 95]. A primeira delas, chamada fase de iniciação, vasculha o espaço de soluções no intuito de obter uma solução de baixa energia (próxima a um mínimo local). A segunda, a fase de amostragem, caracteriza-se pela busca de movimentos que levem a solução de baixa energia, obtida na fase anterior, a posições de energia "equivalente" no espaço de soluções. A equivalência é aqui definida como uma faixa de valores de energia, para mais e para menos, da energia obtida no término da fase de iniciação. Assim, se imaginarmos um espaço de soluções tridimensional, o que a otimização microcanônica pretende é, ao atingir uma posição próxima a um mínimo local na fase de iniciação, ao invés de tentar "escalar os morros" que o espaço de soluções oferece, tenta contorná-los. Uma vez finda a fase de amostragem, uma nova fase de iniciação é iniciada, tentando atingir uma outra posição de baixa energia. O processo se repete até que uma condição de parada seja atingida. Na seqüência, essas duas fases são detalhadas.

i) Fase de iniciação:

A configuração de baixa energia é atingida segundo uma busca aleatória, na qual, a partir de uma alocação inicial, é aceito cada movimento proposto que diminua a energia do sistema considerado. Caso o movimento proposto leve a um aumento de energia, a alocação não é aceita. Entretanto, mesmo não aceitando o movimento, o algoritmo cuida de armazenar, em um vetor *vet_saltos_ruins*, a variação de energia positiva ΔE que a aplicação desse movimento causaria caso fosse aceito. A razão desse vetor é manter um mapeamento da topologia do espaço de soluções na região próxima à solução atual. Isso será útil na fase de amostragem. A fase de iniciação termina quando um determinado número de movimentos consecutivos (*max_mov_rej_consec*) for proposto sem que nenhum deles leve a uma diminuição da energia do sistema.

A última tarefa da fase de iniciação é verificar se a energia da solução atingida (Ω) é menor que qualquer outra encontrada por fases de iniciação (Ω_{min}) anteriores. Caso a resposta seja afirmativa, a melhor solução é atualizada. Caso a resposta seja negativa, cuida-se de atualizar a variável *n_iter_sem_melhora_consec*, que guarda o número de iterações consecutivas do algoritmo de otimização microcanônica sem que uma melhoria da solução tenha sido atingida. Ao ser atingido esse número limite, o algoritmo é interrompido. O pseudo-código da figura 1 apresenta o esqueleto dessa fase.

ii) Fase da amostragem:

No início dessa etapa deve-se escolher os valores dos parâmetros que controlam a faixa de energia dentro da qual as soluções dessa fase poderão se mover. Tomando como base a energia da solução obtida na fase de iniciação, essa faixa deve estar dentro dos limites indicados na figura 2.

Na figura 2, temos:

E_S : Energia da solução obtida na fase de iniciação
E_{DMAX} : Energia máxima do demônio
E_{DI} : Energia inicial do demônio

Observar que aqui é utilizado o conceito de "demônio", introduzido por Creutz [Creu 83]. O demônio representa um grau de liberdade extra que gera perturbações controladas no estado do sistema, permitindo que o mesmo evolua a partir do mínimo local. Ele corresponde a uma variável que contém valores de energia (E_D) variando na faixa de zero até um valor máximo (E_{DMAX}). Assim, no início da fase de amostragem os valores dos parâmetros E_{DI} e E_{DMAX} devem ser obtidos. Tais valores serão oriundos do vetor de saltos de energia para os movimentos rejeitados, construído na fase de iniciação (*vet_saltos_ruins*). O procedimento é colocar em ordem crescente os elementos deste vetor, atribuir o x-ésimo elemento ao parâmetro E_{DI} e o y-ésimo elemento ao parâmetro E_{DMAX} (onde $y \geq x$) . Dessa forma, o demônio estará totalmente definido para a fase de amostragem. Na implementação descrita nesse trabalho foi utilizado $E_{DMAX} = E_{DI}$, ou seja, x = y.

```
fase_de_iniciação {
    n_mov_rej_consec ← 0
    Enquanto ( n_mov_rej_consec < max_mov_rej_consec ) faça {
        Propor movimento aleatório (tarefa k para o processador i)
        Obter nova alocação, Ω*, a partir da alocação atual, Ω
        Obter o custo desse movimento: ΔE = E(Ω*) - E(Ω)
        Se ( ΔE < 0 ) então faça {
            n_mov_rej_consec ← 0
            Ω ← Ω*
        }
        Se não faça {
            Armazenar ΔE no vetor vet_saltos_ruins
            n_mov_rej_consec ← n_mov_rej_consec + 1
        }
    }
    Se ( E(Ω) < E(Ω_min) ) então faça {
        Ω_min ← Ω
        n_iter_sem_melhora_consec ← 0
    }
    Se não faça {
        n_iter_sem_melhora_consec ← n_iter_sem_melhora_consec + 1
    }
}
```

Fig. 1 - Otimização microcanônica: fase de iniciação

Fig. 2 - Faixa de energia na fase de mostragem

A cada iteração da fase de amostragem, um movimento é proposto. Caso este movimento leve a um acréscimo de energia, ele será aceito se o demônio contiver energia suficiente para suprir tal acréscimo. Isto porque cada acréscimo de energia ao sistema deve ser debitado do demônio, e isto só será possível caso a sua energia após esse débito seja maior do que zero. Da mesma forma, caso o movimento proposto leve a um decréscimo de energia, ele só será aceito se a capacidade do demônio não for ultrapassada. Isto porque cada decréscimo de energia do sistema deve ser creditado ao demônio, e isso só será possível caso a sua energia após esse crédito seja menor do que E_{DMAX}. Este mecanismo garante que somente movimentos dentro da faixa de energia anteriormente citada serão aceitos. A fase de amostragem termina após um determinado número de iterações (*max_iter_amost*). O pseudo-código da figura 3 indica a estrutura do procedimento da fase de amostragem.

Os critérios de parada das duas fases do algoritmo microcanônico podem ser inequivocamente estabelecidos. A fase de iniciação cessa quando um estado próximo a um mínimo local foi atingido. A fase de amostragem pode ser interrompida quando o equilíbrio termodinâmico for estabelecido, o que pode ser rigorosamente determinado a partir de uma análise da distribuição dos estados do demônio [Torr 95]. Na implementação aqui apresentada, heurísticas mais simples para o término da segunda fase foram seguidas [Linh 96].

```
fase_de_amostragem {
        Ordenar crescentemente os elementos do vetor vet_saltos_ruins
        Fazer E_DI igual ao x-ésimo elemento do vetor vet_saltos_ruins
        Fazer E_DMAX igual ao y-ésimo elemento do vetor vet_saltos_ruins
        E_D ← E_DI
        n_iter_amost ← 0
        Enquanto ( n_iter_amost < max_iter_amost ) faça {
                Propor movimento aleatório (tarefa k para o processador i)
                Obter nova alocação, Ω* , a partir da alocação atual, Ω
                Obter o custo desse movimento: ΔE = E(Ω*) - E(Ω)
                Se (ΔE < 0 ) então faça {
                        Se ( E_D - ΔE ≤ E_DMAX ) então faça {
                                Ω ← Ω*
                                E_D ← E_D - ΔE
                        }
                }
                Se não faça {
                        Se ( 0 ≤ E_D - ΔE) {
                                Ω ← Ω*
                                E_D ← E_D - ΔE
                        }
                }
                n_iter_amost ← n_iter_amost +1
        }
}
```

Fig. 3 - Otimização microcanônica: fase de amostragem

3. O Modelo Empregado

Para efeito de ilustração da aplicação do método vamos utilizar o mesmo modelo empregado em [Tind 92]. Assim, seja um sistema distribuído cuja topologia do sistema de comunicação é um barramento, para o qual, para efeito de controle de acesso, é utilizada uma variação do conhecido protocolo de passagem de ficha. Cada um dos processadores conectados ao barramento tem velocidade de processamento e capacidade de memória conhecidas e limitadas. Em cada um dos processadores, um determinado conjunto de tarefas periódicas, com restrições de tempo severas (*hard real-time*), é executado. Dentro de um período, uma determinada tarefa executa por um número limitado de ciclos de UCP (tempo de execução máximo conhecido). Cada tarefa tem exigências fixas de memória e processamento, e pode enviar um número limitado de mensagens, de tamanho fixo, para tarefas que executam no mesmo processador ou em nós remotos. É assumido que o tempo de transmissão de mensagens entre tarefas no mesmo processador é nulo. Dessa forma, o tempo limite para enfileirar uma mensagem para transmissão pode ser representado como na figura 4.

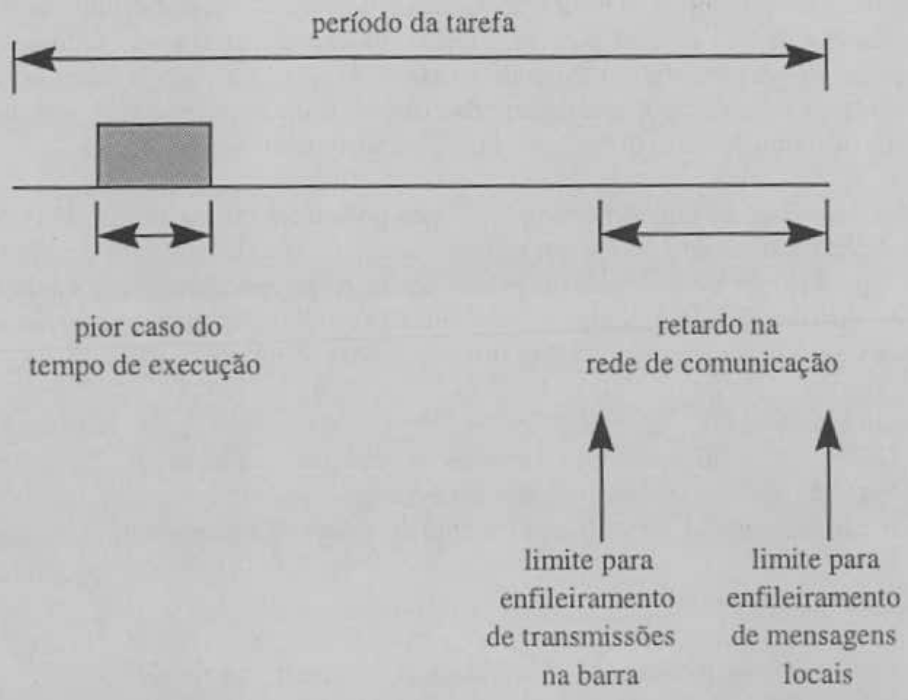

Fig. 4 - Tempo limite para enfileiramento de mensagens

No caso de mensagens enviadas para processadores remotos, o tempo limite de enfileiramento depende da carga do sistema e da velocidade do barramento. Esse tempo pode ser obtido pelo cálculo do tempo de rotação da ficha (TRT - *Token Rotation Time*) associado à distribuição de tarefas entre os processadores (configuração) escolhida.

Uma tarefa alocada a um processador tem de ter garantida a sua execução, a despeito das outras tarefas nele também alocadas. Assim, para uma dada alocação global de tarefas no sistema distribuído, um teste de escalonabilidade local (em cada processador) deve ser realizado para garantir a exeqüibilidade dessa alocação. No caso do modelo cíclico adotado, para a política utilizada, DMS - *Deadline Monotonic Scheduling*, um teste de garantia de escalonamento existe.

Um ponto crucial nesse método é a determinação da função energia (ou custo), que define as penalizações caso alguma(s) restrição (ou restrições) não seja(m) atendida(s). Para esse efeito, as seguintes condições são assumidas:

i) Determinadas tarefas somente podem ser alocadas a processadores específicos, seja por exigirem estrutura especial de E/S, potência computacional, ou outra condição;

ii) Sistemas de tempo real costumam exigir condições especiais de tolerância a falhas. Uma forma de implementar essa tolerância é através do uso de redundância. Assim, será assumido que determinadas tarefas são

implementadas de forma replicada. Claramente, réplicas de uma mesma tarefa não podem ser escaladas no mesmo nó;

iii) Cada nó tem uma capacidade limitada de memória. Dessa forma, o total de memória requerido pelas tarefas a ele alocadas não pode exceder a capacidade existente;

iv) Processadores são limitados em sua capacidade de processamento. Assim, as tarefas alocadas a um determinado processador devem ter garantia de serem escalonadas;

v) Quanto maior o número de mensagens no barramento, maior o TRT, e mais rigoroso fica o tempo limite para enfileiramento de mensagens a serem enviadas para nós remotos. Embora essa não seja uma condição que inviabilize a operação, claramente, um menor TRT é desejável.

A partir dessas restrições, uma função energia pode ser obtida como uma combinação linear das contribuições de cada uma das condições anteriores. Ou seja, a cada restrição, associa-se um custo (ou energia) ao seu não atendimento. Por exemplo, uma energia proporcional ao número de réplicas das mesmas tarefas alocadas aos mesmos processadores pode representar a restrição (ii) acima. A escolha dos coeficientes K_i utilizados na combinação linear é função da importância da restrição para o processo de otimização. Por exemplo, a condição (v), que é desejável ser atendida, não implica em uma alocação inviável. Dessa forma, o seu peso pode ser menor. A função energia final correspondente às restrições anteriormente indicadas pode ser implementada como:

$$E = K_0 \cdot E_{proc} + K_1 \cdot E_{réplica} + K_2 \cdot E_{mem} + K_3 \cdot E_{esc} + K_4 \cdot E_{barra}$$

4. Resultados Obtidos

De forma a comparar as heurísticas, um sistema distribuído com oito nós e quarenta e três tarefas foi utilizado, como descrito em [Tind 92]. Para cada tarefa são especificados o período (em ms), o pior tempo de execução (em ms), a necessidade de memória (em octetos), o tamanho (em octetos) e o destino das mensagens enviadas, além do grupo de nós onde pode ser executada (para as tarefas com restrições de recursos de E/S, por exemplo). Para efeito de tolerância a falhas, cinco tarefas críticas têm uma réplica cada. No caso dos nós, sua capacidade de memória em octetos é fornecida. A velocidade de transmissão no barramento é fixada em 90 octetos/ms.

Para esse conjunto de tarefas e processadores, foram executadas várias baterias de 100 simulações de alocação em um microcomputador PC-Pentium (166 Mhz). Para efeito de comparação, foi utilizado o TRT produzido pelas alocações válidas. Ambas as heurísticas forneceram o TRT mínimo, mas com média (M) e desvio padrão (σ) distintos. O tempo de execução e a percentagem de resultados escalonáveis fornecidos pelas duas heurísticas são marcadamente diferentes. Os resultados da simulação obtidos através do recozimento simulado são idênticos aos reportados por Tindell et al. [Tind 92]. A tabela 1 fornece os resultados numéricos.

Método	$TRT_{mín}$	M[TRT]	σ[TRT]	% escal.	$M[T_{exec}]$	$\sigma[T_{exec}]$
Recoz. simulado	8,56 ms	8,80	0,23	40	121,4 s	18,1
Ot. microcanônica	8,56 ms	8,65	0,15	59	46,5 s	18,0

Tab. 1 - Resultado das simulações

Essa tabela evidencia as vantagens da otimização microcanônica sobre o recozimento simulado. Notar que o conjunto tarefas/nós utilizado é de baixa cardinalidade; um número de tarefas da ordem de 400 é típico em sistemas de controle em tempo real. Fica, assim, clara a sua vantagem, embora o tempo de execução não necessariamente escale linearmente com o número de tarefas e de nós. A figura 5 fornece uma indicação da evolução no tempo para o processo de convergência das duas heurísticas.

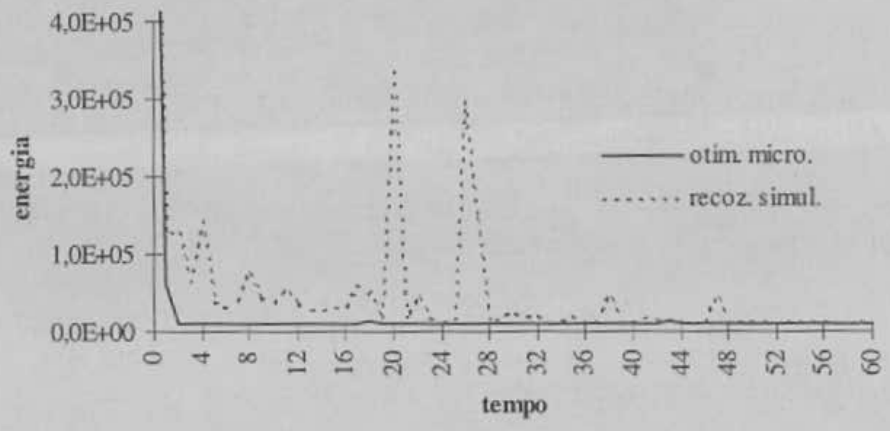

Fig. 5 - Evolução da convergência no tempo

Em um outro conjunto de experimentos realizados em nosso grupo [Linh 96], uma comparação entre a otimização microcanônica e várias outras heurísticas, inclusive busca tabu, foi realizada, tendo como base o problema do caixeiro viajante. Todos os resultados ali obtidos foram favoráveis à otimização microcanônica, independentemente da dimensão do espaço de soluções.

5. Conclusão

Este trabalho foi parcialmente finaciado pela Faperj e pelo ProTeM-CC/CNPq, e é parte do projeto ASAP (Ambiente para Suporte de Aplicações Distribuídas baseado em Objetos), que tem entre seus objetivos o suporte ao desenvolvimento de aplicações distribuídas com características de tempo real e segurança de funcionamento para ambientes abertos heterogêneos. Nesse sentido, nossa meta é o desenvolvimento de técnicas que permitam garantir a escalonabilidade de tarefas com restrição temporal em sistemas distribuídos. Além do trabalho descrito, estamos estudando técnicas que permitam aumentar a percentagem de tarefas aperiódicas que tem garantida sua execução antes de seu tempo limite (*deadline*). Assim, além do desenvolvimento de algoritmos, um simulador para tais políticas está sendo construído, bem como um ambiente de testes em um sistema distribuído real.

O uso de metaheurísticas no escalonamento de tarefas em sistemas distribuídos de tempo real é uma área ainda aberta em vários aspectos. Ao nosso conhecimento, o primeiro trabalho publicado explorando o problema da alocação através do recozimento simulado foi o de Tindell et al. [Tind 92]. Chang e Oldham [Chan 95] também usaram o recozimento simulado para a alocação dinâmica de tarefas em um sistema composto de múltiplas redes locais. Recentemente, Di Natale e Stankovic [DiNa 96] usaram essa heurística para minimização de *jitter* em sistemas distribuídos (com nós multiprocessadores), onde as tarefas têm restrições arbitrárias de tempo limite, precedência e exclusão. Embora o recozimento simulado possa fornecer resultados tão próximos a um mínimo global quanto desejado, isso acarreta um correspondente aumento do tempo de computação. Assim, o uso de outras metaheurísticas nessa classe de problemas deve ser perseguido. Em particular, no caso da otimização microcanônica, soluções paralelas também vêm sendo investigadas.

6. Referências

[Aart 89] E. Aarts e J. Korst, "Simulated Annealing and Boltzmann Machines", John Wiley & Sons, 1989.

[Auds 94] N. Audsley e A. Burns, "Real-Time Scheduling", Relatório Técnico YCS 134/94, Departamento de Ciência da Computação, Universidade de York, Inglaterra, 1994.

[Barr 97] A.M. Barroso, "Escalonamento e Alocação de Tarefas em Sistemas Distribuídos de Tempo Real", proposta de dissertação de mestrado, P.G. em Computação Aplicada e Automação, Universidade Federal Fluminense, Janeiro de 1997.

[Best 96] A. Bestravos, "Load Profiling in Distributed Real-Time Systems", Relatório Técnico 96-017, Departamento de Ciência da Computação, Universidade de Boston, EUA, Agosto de 1996.

[Chan 95] H.W.D. Chang e W.J.B. Oldham, "Dynamic Task Allocation Models for Large Distributed Computing Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 6, nº 12, pp. 1301-1315, Dezembro de 1995.

[Creu 83] M. Creutz, "Microcanonical Monte Carlo Simulation", Physics Review Letters, vol. 50, p.1411, Maio de 1983.

[DiNa 95] M. DiNatale e J.A. Stankovic, "Applicability of Simulated Annealing Methods to Real-Time Scheduling and Jitter Control", 16th IEEE Real-Time Systems Symposium, Pisa, Itália, Dezembro de 1995.

[Linh 96] A. Linhares, "Otimização Microcanônica Aplicada ao Problema do Caixeiro-Viajante", Dissertação de Mestrado, P.G. em Computação Aplicada e Automação, Universidade Federal Fluminense, Outubro de 1996.

[Tind 92] K.W. Tindell, A. Burns e A.J. Wellings, "Allocating Hard Real-Time Tasks: An NP-Hard Problem Made Easy", The Journal of Real-Time Systems, vol. 4, pp. 145-165, Junho de 1992.

[Torr 95] J.R.A. Torreão e E. Roe, "Microcanonical Optimization Applied to Visual Processing", Physics Letters A, vol. 205, pp.377-382, Setembro de 1995.