Escalonamento de processos utilizando técnicas de ACO

Bruno Rodrigues Nery[1], **Rodrigo Fernandes de Mello**[1], **André C. P. L. F. de Carvalho**[1]

[1]ICMC - Departamento de Ciência de Computação – Universidade de São Paulo (USP)
São Carlos – SP – Brasil

solo@grad.icmc.usp.br, {mello, andre}@icmc.usp.br

Resumo. *A disponibilidade crescente de microprocessadores de baixo custo e a evolução das redes de computadores possibilitaram a construção de sofisticados sistemas distribuídos. A capacidade de computação desses sistemas motivou a adoção de clusters para construir soluções de alto desempenho. O melhor escalonamento de processos em clusters originou muitas propostas de algoritmos de escalonamento e balanceamento de carga. Essas propostas motivaram este trabalho, que define, avalia e implementa um algoritmo de balanceamento de carga para clusters heterogêneos. Esse algoritmo, chamado Ant Scheduler, utiliza conceitos de colônias de formigas como fonte de inspiração para o desenvolvimento de soluções de otimização. Resultados experimentais obtidos na comparação do Ant Scheduler com outras abordagens investigadas na literatura mostram sua capacidade de minimizar o tempo médio de resposta dos processos, obtendo um maior desempenho.*

1. Introdução

A disponibilidade crescente de microprocessadores de baixo custo aliada a evolução das redes de computadores possibilitaram a construção de sofisticados sistemas distribuídos. Nesses sistemas, os processos executados em uma rede de computadores se comunicam de forma a realizar uma tarefa colaborativamente. Um algoritmo de balanceamento de carga é frequentemente adotado para distribuir os processos entre os computadores.

Um algoritmo de balancemento de carga é responsável por distribuir a carga dos processos entre as máquinas de um ambiente distribuído [Shivaratri et al. 1992]. Krueger and Livny [Krueger and Livny 1987] demonstram que esses algoritmos podem reduzir a média e o desvio padrão do tempo de resposta dos processos. Tempos de resposta menores resultam em um melhor desempenho na execução de processos.

Os algoritmos de balanceamento de carga envolvem quatro políticas: transferência, seleção, localização e informação [Shivaratri et al. 1992, Sinha 1997]. A política de transferência determina se uma máquina pode participar numa transferência de tarefas, como servidor ou receptor de processos. A política de seleção define os processos que devem ser transferidos da máquina mais ocupada para a menos ocupada. A política de localização é responsável pela procura de um parceiro de transferência adequado (servidor ou receptor) para uma máquina, uma vez que a política de transferência decidiu sobre seu estado. Um servidor oferece processos, quando está sobrecarregado; um receptor requisita processos, quando não está ocupado. A política de informação define quando e como a informação sobre o estado dos computadores é atualizada no sistema. Vários trabalhos foram feitos na área de balanceamento de carga [Zhou and Ferrari 1987, Theimer and Lantz 1989, Shivaratri et al. 1992, Amir 2000, Hsu 2000, Mitzenmacher 2001, Mello et al. 2004a, Senger et al. 2005].

Zhou e Ferrari [Zhou and Ferrari 1987] avaliaram cinco algoritmos de balanceamento de carga iniciados pelo servidor, i.e. iniciados pela máquina sobrecarregada. Esses algoritmos são: Disted, Global, Central, Random e Lowest. No Disted, quando uma máquina sofre qualquer modificação em sua carga, esta gera mensagens para as outras de forma a informar sua carga atual. No Global, uma máquina centraliza toda a informação de carga do ambiente e envia mensagens para as outras de forma a mantê-las atualizadas. No Central, como no Global, uma máquina central recebe toda a informação de carga do sistema; entretanto, esta não atualiza as outras máquinas. Esta máquina centralizada decide a alocação dos recursos no ambiente. No Random, nenhuma informação sobre a carga do ambiente é utilizada. Nesse algoritmo, uma máquina é escolhida aleatoriamente para receber um processo a ser iniciado. No Lowest, a informação de carga é enviada somente quando é pedida. Quando uma máquina inicia um processo, esta pede informação e analisa a carga de um conjunto pequeno de máquinas e manda o processo para a menos ocupada.

Mello *et al.* [Mello et al. 2004a] propuseram um algoritmo de balanceamento de carga para clusters heterogêneos de computadores. Esse algoritmo, chamado TLBA (em inglês, Tree Load Balancing Algorithm), organiza os computadores numa topologia de árvore virtual e coloca os processos da raiz para as folhas. Nos experimentos realizados, o algoritmo apresentou um desempenho muito bom, com um baixo tempo de resposta.

Senger *et al.* [Senger et al. 2005] propuseram o *GAS*, um algoritmo de escalonamento genético que usa informações da capacidade das máquinas, comunicação entre as aplicações e a carga de processamento de forma a alocar recursos em ambientes heterogêneos distribuídos. O *GAS* usa técnicas de algoritmos genéticos para encontrar o conjunto de recursos mais adequado para as aplicações.

Esses trabalhos e o desenvolvimento de novas técnicas de computação baseadas em biologia motivaram a proposta de um novo algoritmo de balanceamento de carga, chamado Ant Scheduler, que tenta aplicar técnicas de ACO [Dorigo et al. 1996] para escalonar processos em ambientes distribuídos heterogêneos. Experimentos foram conduzidos para avaliar o algoritmo proposto e compará-lo com outros descritos na literatura. Os resultados confirmam sua aplicação em ambientes distribuídos heterogêneos e seu potencial como uma abordagem alternativa para balanceamento de carga.

Este artigo é dividido nas seguintes seções: seção 2. introduz brevemente os conceitos básicos de ACO; seção 3. descreve o algoritmo de escalonamento proposto baseado em colônias de formigas, chamado Ant Scheduler; As simulações realizadas e os resultados obtidos são apresentados na seção 4.; a seção 5. descreve o processo de implementação do algoritmo; por fim, a seção 6. possui as principais conclusões desse trabalho.

2. Otimização baseada em colônia de formigas

Nos últimos anos, novas técnicas computacionais têm sido criadas inspiradas em mecanismos encontrados na natureza. Esta área, chamada Computação Bioinspirada, provê soluções motivadas biologicamente para muitos problemas reais. Entre as técnicas de Computação Bioinspirada, podemos mencionar Redes Neurais Artificiais (ANN), Algoritmos Evolucionários (EA), Sistemas Imunológicos Artificiais (SIA) e Otimização Baseada em Colônia de Formigas (ACO).

Uma das técnicas mais promissoras, ACO [Dorigo et al. 1996], é uma meta-

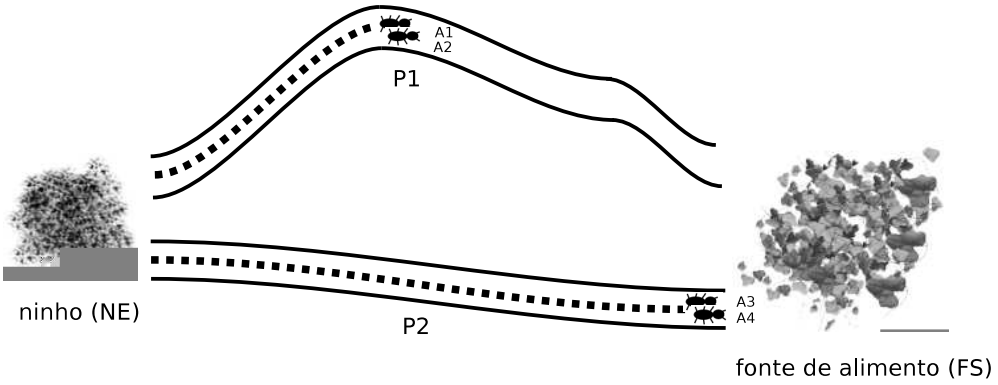

Figura 1. Exemplo de formigas buscando alimento

heurística baseada na estrutura e no comportamento de colônias de formigas, e foi aplicada com sucesso a vários problemas de otimização [Shmygelska and Hoos 2005, Daniel Merkle, Martin Middendorf 2000, Acan 2004].

Organismos aparentemente simples, formigas podem lidar com tarefas complexas agindo coletivamente. Esse comportamento coletivo é realizado através da emissão de uma substância química chamada feromônio. Durante seu movimento, as formigas depositam feromônio nos caminhos que percorrem. A presença de feromônio num caminho atrai outras formigas. Assim, o feromônio é peça chave na troca de informação entre as formigas, possibilitando a realização de muitas tarefas importantes. Um exemplo clássico é a escolha do caminho mais curto entre seu ninho e uma fonte de comida.

Formalmente, a ACO pode ser definida como segue: Suponha que existam quatro formigas e dois caminhos possíveis (Figura 1) de um ninho N_E até uma fonte de alimentos F_S: P_1 e P_2, tal que $P_1 > P_2$. Inicialmente, todas as formigas (A_1, A_2, A_3 e A_4) estão em N_E e devem escolher entre os caminhos P_1 e P_2 para chegar até F_S.

1. Em N_E, as formigas (A_1, A_2, A_3 e A_4) não sabem a localização da fonte de alimentos (F_S). Assim, estas escolhem aleatoriamente entre P_1 e P_2, com a mesma probabilidade. Suponha que A_1 e A_2 escolhem P_1, e A_3 e A_4 escolhem P_2.
2. Enquanto as formigas passam por P_1 e P_2, estas deixam uma certa quantidade de feromônio no caminho, τ_{C1} e τ_{C2}, respectivamente.
3. Como $P_2 < P_1$, A_3 e A_4 chegam a F_S antes de A_1 e A_2. Nesse momento, $\tau_{C2} = 2$, mas, como A_1 e A_2 ainda não chegaram a F_S, $\tau_{C1} = 0$. Para voltar a N_E, A_3 e A_4 devem novamente escolher entre P_1 e P_2. Como, em F_S, $\tau_{C2} > \tau_{C1}$, a probabilidade destas escolherem P_2 é maior. Suponha que A_3 e A_4 escolhem P_2.
4. Quando A_3 e A_4 chegam novamente a N_E, τ_{C2} chega a 4. Esse aumento em τ_{C2} consolida o posto de P_2 como o caminho mais curto. Quando A_1 e A_2 chegam a F_S, $\tau_{C2} = 4$ e $\tau_{C1} = 2$. Assim, a probabilidade de A_1 e A_2 voltarem a N_E por P_2 é maior.

Neste exemplo, qualquer formiga em F_S ou N_E consegue determinar o melhor caminho uma vez que A_3 e A_4 tenham chegado em F_S. Se uma formiga tiver que escolher entre dois caminhos quando nenhum dos dois apresenta feromônio, essa escolhe aleatoriamente entre P_1 e P_2 com uma probabilidade de 0.5. Entretanto, quando feromônio está presente, existe uma *maior probabilidade* da formiga escolher o caminho com

uma concentração maior de feromônio.

É bom notar que a maior parte das abordagens de ACO, apesar de serem inspiradas pelos paradigmas encontrados em formigas, não são uma réplica dos mesmos. Características das formigas podem estar ausentes e outras técnicas adicionais podem ser usadas para complementar o uso de feromônios.

Um dos problemas que pode surgir com o uso de feromônios é o da estagnação. Suponha, por exemplo, que as formigas fiquem viciadas por um deteminado caminho. No futuro, aquele caminho pode ficar congestionado, tornando-se não ótimo. Outro problema surge quando um caminho preferido é obstruído e não pode mais ser usado pelas formigas. Para reduzir estes problemas as seguintes abordagens foram utilizadas:

Evaporação: Reduza os valores de feromônio τ_i de um fator ρ para evitar a alta concentração de feromônios em caminhos ótimos, que evita a explocação de outras (novas ou melhores) alternativas.

Heurística: Esta abordagem combina a concentração de feromônios τ_i e uma função heurística η_i. A importância relativa de cada informação é definida por dois parâmetros, α and β.

3. Ant Scheduler

O problema da alocação de processos em ambientes multi-computadores heterogêneos pode ser modelado utilizando gráficos, como ilustrado na Figura 2 [Amir et al. 2000]. Neste caso, cada processo tem um nó S como origem e um nó T como destino. S e T são conectados por N caminhos diferentes, cada um correspondendo a uma das máquinas do cluster. Este grafo é empregado de modo a aumentar a performance do sistema diminuindo o congestionamento dos caminhos.

Os bons resultados obtidos pela ACO em problemas baseados em grafos favoreceu o uso de ACO para a otimização da alocação de processos em ambientes heterogêneos de cluster. Para tanto, cada processo iniciado pode ser visto como uma formiga procurando pelo melhor caminho para chegar de forma mais rápida do ninho até a fonte de alimento. Sendo assim, cada máquina pode ser vista como um caminho e a conclusão da execução do programa como a fonte de alimento.

O algoritmo Ant Scheduler é baseado no *ant-cycle* proposto por Dorigo *et al.* [Dorigo et al. 1996]. Quando o computador responsável pela distribuição de processos (mestre) no cluster é iniciado, cada aresta no grafo tem sua intensidade de feromônio iniciada com um valor $\tau_i = c$. Quando um processo é iniciado, este é visto como uma formiga e pode migrar. Assim, este processo deve escolher um dos caminhos (as máquinas do cluster) para seu destino (terminar sua execução). A probabilidade de uma formiga escolher um caminho i é dada pela equação 1, onde τ_i é o nível de feromônio no caminho i, η_i é um valor associado com a máquina i por uma função heurística e os parâmetros α e β controlam a relevância de τ_i e η_i.

$$p_i = \frac{\tau_i^\alpha \cdot \eta_i^\beta}{\sum_{j=1}^{N} \tau_j^\alpha \cdot \eta_j^\beta} \tag{1}$$

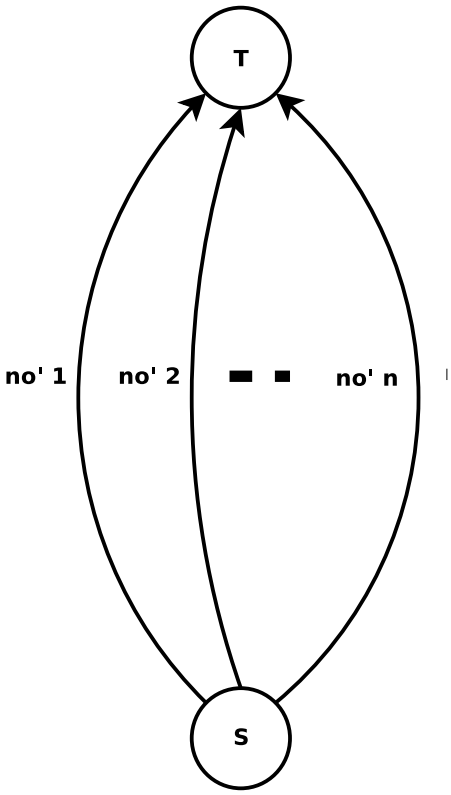

Figura 2. Cluster modelado como um grafo de recursos

Neste artigo, a função heurística é proporcional à carga da i-ésima máquina. O denominador é a soma dos níveis de feromônio ponderados pela função heurística e controlado pelos parâmetros α e β. Quando uma formiga chega a seu destino (quando um processo termina), ela deposita uma quantia delta de feromônio no caminho (equação 2, onde Q é uma constante e T é o tempo gasto por uma formiga para chegar em seu destino (a duração da execução do processo)).

$$\Delta_i = \Delta_i + \frac{Q}{T} \tag{2}$$

Para evitar a escolha viciada de uma máquina particular, uma evaporação contínua de feromônio acontece nos caminhos. Assim, em intervalos de tempo regulares, a quantidade de feromônio muda de acordo com a regra da equação 3, onde ρ é um coeficiente tal que $(1 - \rho)$ representa a evaporação de feromônio entre t e $t + 1$. Além disso, Δ_i é zerado em intervalos de tempo regulares.

$$\tau_i(t + 1) = \rho \cdot \tau_i(t) + \Delta_i \tag{3}$$

Um problema com essa abordagem é a possibilidade de uma performance ruim devido a diferente faixa de valores para τ_i e η_i. Para resolver esse problema, decidimos normalizar os valores utilizando uma escala logarítimica, modificando a equação 1 e originando a equação 4.

$$p_i = \frac{(\log \tau_i)^\alpha \cdot (\log \eta_i)^\beta}{\sum_{j=1}^{N} (\log \tau_j)^\alpha \cdot (\log \eta_j)^\beta} \quad (4)$$

Outro problema encontrado foi a alocação frequente de um valor baixo, entre 0 e 1, para τ_i, fazendo $\log \tau_i < 0$, levando a valores irrealísticos para a função de probabilidade. Esse problema foi resolvido utilizando $\log \epsilon + \tau_i$ ao invés de $\log \tau_i$, onde $\epsilon = 1$, resultando na equação 5.

$$p_i = \frac{(\log \epsilon + \tau_i)^\alpha \cdot (\log \epsilon + \eta_i)^\beta}{\sum_{j=1}^{N} (\log \epsilon + \tau_j)^\alpha \cdot (\epsilon + \log \eta_j)^\beta} \quad (5)$$

O Ant Scheduler é composto dos Algoritmos 1, 2 e 3.

Algoritmo 1 Ant Scheduler: processo iniciado

Escolher uma máquina com probabilidade p_i, calculada utilizando a equação 5
Alocar processo no nó escolhido

Algoritmo 2 Ant Scheduler: processo finalizado

Atualizar a quantidade de feromônio Δ_i usando a Equação 2

Algoritmo 3 Ant Scheduler: evaporação de feromônio

loop
  para cada i tal que i é um nó do cluster **faça**
    Atualizar a quantidade de feromônio τ_i usando a Equação 3
    Zerar a quantidade de feromônio Δ_i ($\Delta_i = 0$)
  fim para
fim loop

O primeiro algoritmo deve ser executado sempre que um novo processo com possibilidade de migração for iniciado. Já o segundo algoritmo deve ser executado sempre que um processo terminar sua execução. O terceiro algoritmo deve ser executado continuamente, de modo a efetuar a evaporação do feromônio.

4. Resultados da simulação

Experimentos foram realizados em ambientes com 32 computadores, o que permitiu uma avaliação do comportamento do algoritmo Ant Scheduler. Os parâmetros utilizados para o Ant Scheduler foram $\alpha = 1$, $\beta = 1$, $\rho = 0.8$ e $Q = 0.1$. Aplicações paralelas de 8, 64 e 128 tarefas foram avaliadas.

A carga gerada por estas aplicações segue o modelo de carga de Feitelson [1] [Feitelson and Jette 1997]. Este modelo é baseado na análise dos registros do comportamento de execução dos seguintes ambientes: iPSC/860 de 128 nós na NASA Ames; IBM SP1 de 128 nós em Argonne; Paragon de 400 nós na SDSC; Butterfly de 126 nós na LLNL; IBM SP2 de 512 nós no CTC; Paragon de 96 nós na ETH, Zurich.

[1] http://www.cs.huji.ac.il/labs/parallel/workload/models.html

De acordo com este modelo, a chegada de processos deriva de uma função de distribuição de probabilidade exponencial (`fdp`) de média igual a 1500 segundos. Este modelo foi adotado para simular e fazer uma avaliação comparada do Ant Scheduler com outros algoritmos encontrados na literatura.

Para realizar os experimentos e avaliar os algoritmos de escalonamento utilizados neste estudo, o aluno usou o modelo para a criação de ambientes heterogêneos distribuídos e avaliação do tempo de resposta de aplicações paralelas - UniMPP (em inglês, *Modelo Unificado para Predição de Performance*) [de Mello and Senger 2006]. Tal modelo combina o consumo de CPU dos modelos de [Amir 2000] e [et. al 2002], o tempo gasto para transmissão de mensagens descrito em [Culler et al. 1993] e [Sivasubramaniam et al. 1994] e o volume de mensagens e a distribuição probabilística de geração de mensagens de [Choe and Park 2002] e [Vetter and Mueller 2003]. Este modelo gera um tempo de execução médio para as aplicações submetidas ao sistema. O tempo médio de resposta é gerado após ser alcançado um intervalo de confiança de 95%.

Para avaliar o Ant Scheduler, uma classe foi incluída no simulador orientado a objetos [2] [de Mello and Senger 2006]. Essa classe implementa as funcionalidades do Ant Scheduler e foi agregada ao simulador do modelo UniMPP para gerar os tempos de resposta médios da execução de aplicação. Esses resultados foram utilizados para avaliar o desempenho do Ant Scheduler e permitir a comparação com algoritmos alternativos.

4.1. Parametrização do ambiente

Ambientes com 32 computadores foram simulados. Os PE_i para o modelo UniMPP foram definidos probabilisticamente: a capacidade de computação de cada computador, definida por uma distribuição probabilística uniforme com uma média de 1500 Mips (em inglês, Milhões de instruções por segundo); a capacidade total de memória principal de cada computador, parametrizada por uma distribuição probabilística uniforme de média 1024 MBytes; a capacidade total de memória virtual de cada computador, definida por uma distribuição probabilística uniforme com uma média de 1024 MBytes; o throughput de leitura de disco, parametrizado por uma distribuição probabilística uniforme de média 40 MBytes; e o throughput de escrita do disco, definido por uma distribuição probabilística uniforme de média 30 MBytes. Tais médias foram baseadas nos valores médios obtidos utilizando um grupo de máquinas de um laboratório de pesquisa. Estas medidas seguiram o benchmark proposto por Mello e Senger [de Mello and Senger 2006][3]. Tais parâmetros e o uso de distribuições probabilísticas permitem a criação de ambientes heterogêneos para avaliar o Ant Scheduler sob essas circunstâncias.

O modelo de carga de Feitelson foi usado para definir o padrão de ocupação (em Mips) dos processos (ou tarefas) que fazem parte da aplicação paralela. Os parâmetros restantes do modelo UniMPP para representar um processo foram definidos como: a quantidade de memória estática utilizada pelo processo, baseada em uma distribuição exponencial com uma média de 300 KBytes; a quantidade de memória alocada dimanicamente, definida por uma distribuição exponencial com uma média de 1000 KBytes; a probabilidade de leitura de arquivos, definida por uma distribuição exponencial com uma

[2] SchedSim - disponível em http://www.icmc.usp.br/~mello/outr.html.

[3] disponível em http://www.icmc.usp.br/~mello/

média de uma leitura a cada 1000 ticks de relógio, sendo o mesmo valor utilizado para parametrizar a escrita em arquivos; e o envio e o recebimento de mensagens pela rede, parametrizado por uma distribuição exponencial com uma média de uma mensagem a cada 1000 ticks de relógio.

Nesses experimentos, todos os computadores estavam localizados na mesma rede e organizados como um cluster padrão. Dentro da mesma rede, os computadores apresentam um atraso de comunicação (RTT - Round-Trip Time, de acordo com o modelo de Hockney [Hockney 1996]) de 0.0001 (valor médio extraído pelo benchmark de rede de Mello e Senger [de Mello and Senger 2006] para uma rede Fast Ethernet).

4.2. Algoritmos simulados

O desempenho do Ant Scheduler é comparado com 5 algoritmos de escalonamento e balanceamento de carga propostos na literatura: `DPWP` [Araújo et al. 1999a], `Random`, `Central`, `Lowest` [Zhou and Ferrari 1987] e `TLBA` [Mello et al. 2004b].

O algoritmo `DPWP` (*Dynamic Policy Without Preemption*) escalona aplicações paralelas levando em conta um cenário distribuído e heterogêneo de execução [Araújo et al. 1999a, Araújo et al. 1999b]. Este algoritmo permite o escalonamento de aplicações desenvolvidas em PVM, MPI e CORBA. Os detalhes envolvidos na atribuição de tarefas são supervisionados pelo software de escalonamento, AMIGO [Souza 2000].

O índice de carga usado neste algoritmo é o tamanho da fila de cada elemento de processamento (EP). Nesse índice, a carga de um EP é baseada na relação entre o número de tarefas sendo executadas e a capacidade de processamento do EP. A capacidade de processamento é medida através de *benchmarks* específicos [Souza 2000, Santos 2001]. O `DPWP` algoritmo então usa os índices de carga para criar uma lista ordenada de EPs. As tarefas das aplicações paralelas são então atribuídas para as EPs desta lista, numa estrutura circular.

Os algoritmos `Lowest`, `Central` e `Random` foram estudados em [Zhou and Ferrari 1987]. Neste estudo, estes algoritmos foram empregados para balanceamento de carga. Eles são definidos por dois componentes principais: LIM (em inglês, *Gerente de informação de carga*) e LBM (em inglês, *Gerente de balanceamento de carga*). O primeiro componente é responsável pela política de informação e para monitoramento da carga dos computadores de modo a calcular os índices de carga. O último define a política de informação e tenta encontrar o computador mais apropriado para atribuir os processos. A maneira que esses componentes realizam suas tarefas permite a definição de algoritmos distintos. Tais algoritmos diferem dos algoritmos de escalonamento por serem desenvolvidos para realizar balanceamento de carga; assim, não existe um software global de escalonamento para o qual as aplicações devem ser submetidas. De fato, cada EP deve gerenciar localmente as tarefas de aplicações que são executadas nele, iniciando-as localmente ou definindo a maneira pela qual outra EP será encontrada para executar as tarefas.

O algoritmo `Lowest` realiza o balanceamento de carga minimizando o número de mensagens trocadas entre os componentes do algoritmo. Quando uma tarefa é submetida ao ambiente, a EP LIM que recebeu a requiição define um conjunto limitado de LIMs

remotos. A carga das EPs deste conjunto é recebida e a EP menos ativa é escolhida para receber a tarefa.

O algoritmo `Central` utiliza um LBM mestre e um LIM mestre. Ambos centralizam as decisões de balanceamento de carga. O LIM mestre recebe os índices de carga enviados pelos LIMs escravos. O LBM mestre recebe requisições para alocar processos para o sistema e utiliza a informação fornecida pelo LIM mestre para tomar decisões.

O algoritmo `Random` não utiliza informações sobre a carga do sistema para tomar decisões. Quando uma tarefa é submetida para o ambiente de execução, o algoritmo seleciona uma EP aleatoriamente. O índice de carga utilizado pelos algoritmos `Lowest` e `Central` é calculado baseado no número de processos na fila de execução. Zhou and Ferrari [Zhou and Ferrari 1987] observaram que os algoritmos `Lowest` e `Central` apresentam um desempenho similar e que o algoritmo `Random` apresenta os piores resultados. Eles também indicaram o algoritmo `Lowest` para cenários distribuídos.

O algoritmo `TLBA` (em inglês,*Algoritmo de balanceamento de carga em árvore*) realiza o balanceamento de carga em sistemas distribuídos heterogêneos com uma factibilidade crescente e escalável [Mello et al. 2004b]. Este algoritmo cria uma interconexão topológica lógica entre as EPs, num formato de árvore, e realiza a migração de tarefas de modo a melhorar o balanceamento de carga do sistema.

O algoritmo `GAS` (em inglês, *Genetic Algorithm for Scheduling*) utiliza algoritmos genéticos para propor soluções otimizadas de escalonamento [Senger et al. 2005]. O algoritmo considera conhecimento sobre o tempo de execução e o comportamento das aplicações para definir o conjunto mais adequado de recursos computacionais para suportar uma aplicação paralela num ambiente distribuído heterogêneo. `GAS` utiliza operadores de crossover e mutação para otimizar a busca probabilística pela melhor solução de um problema. Baseado em Genética e Evolução, AGs são adequados para buscas globais e podem ser implementados eficientemente em máquias paralelas.

5. Implementação

Para permitir experimentos reais, Ant Scheduler foi implementado utilizando o kernel do Linux 2.4.24. Essa implementação utiliza o serviço the migração de processos do patch do openMosix [4]. openMosix é um software desenvolvido para realizar o balanceamento de carga em clusters distribuindo processos.

A implementação foi feita através de um conjunto de traps dentro do kernel do Linux. A primeira trap foi implementada na chamada de sistema *do_fork*. Quando um processo novo é iniciado, *do_fork* é chamada. Essa chamada de sistema executa a primeira trap do Ant Scheduler, que escolhe o nó onde o novo processo será rodado. Essa fase é baseada no nível de feromônio e na capacidade de cada nó. Quando um processo termina, a chamada de sistema *do_exit* é feita. Essa chamada de sistema executa a segunda trap do Ant Scheduler, que atualiza a quantidade de feromônio no nó onde o processo estava rodando.

Essas traps foram implementadas num módulo de kernel utilizando ponteiros para funções, permitindo a mudanças simples para utilizar outro algoritmo de escalonamento

[4]openMosix é um patch do kernel do Linux desenvolvido por Moshe Bar que permite migração automática de processos num ambiente de cluster - disponível em http://openmosix.sourceforge.net

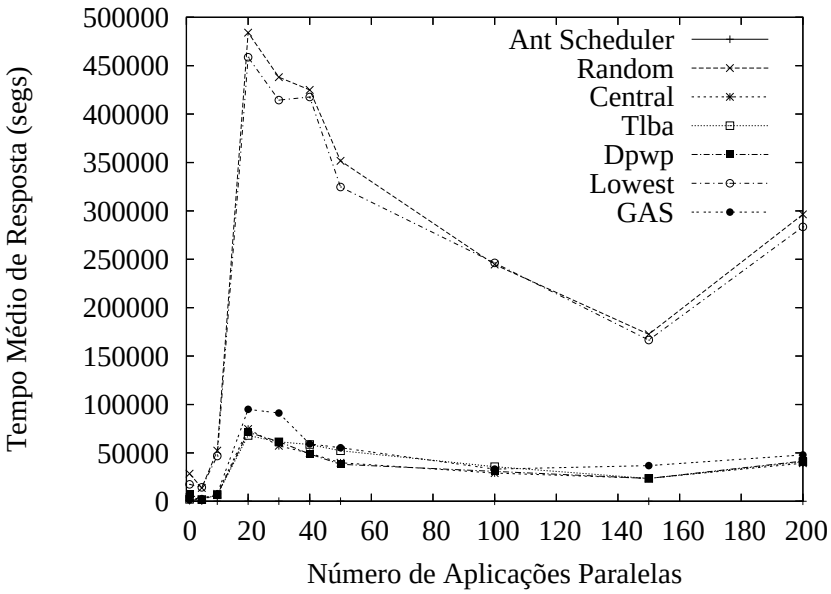

Figura 3. Simulação com 8 tarefas

de processos. Quando o módulo é carregado, ele registra suas funções (traps). O módulo também inicia uma thread que atualiza periodicamente o nível de feromônio de cada nó aplicando a equação 3.

5.1. Resultados

Para a validação do algoritmo Ant Scheduler, seu desempenho foi comparado com os resultados obtidos pelos cinco algoritmos descritos anteriormente. Para tanto, os autores realizaram simulações onde todos esses algoritmos foram avaliados rodando aplicações paralelas compostas de um número diferente de tarefas. As Figuras 3, 4 e 5 mostram o tempo médio de resposta para aplicações paralelas com 8, 64 e 128 tarefas, respectivamente.

O Random obteve os piores resultados, enquando o Ant Scheduler obteve os melhores. A performance ruim obtida pelo GAS pode ser explicada pelo fato de que seu tempo de execução aumenta de acordo com o número de computadores. Isso acontece porque este cria cromossomos maiores que devem ser avaliados pela função de fitness. Essa avaliação consome um longo tempo que é adicionado para o custo de escalonamento, aumentando muito o tempo de execução dos processos. É díficil observar a curva para o Ant Scheduler nas Figuras 3 e 4 devido ao baixo tempo de resposta médio em comparação com os outros algoritmos.

Esses resultados mostram que, em todos os cenários investigados, o Ant Scheduler apresentou o melhor desempenho.

Experimentos também foram feitos para avaliar o tempo de execução de processos para um ambiente utilizando Ant Scheduler e openMosix em um cluster de cinco máquinas Dual Xeon 2.4 Ghz. A tabela 1 mostra os resultados do tempo médio de execução de processos (em segundos) para uma carga de 10 aplicações de baixa, média e alta carga executadas simultaneamente.

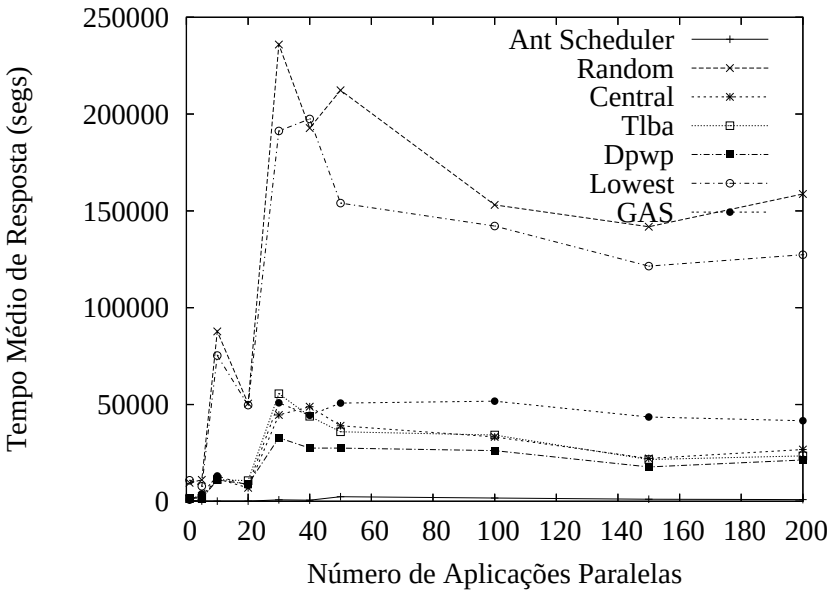

Figura 4. Simulação com 64 tarefas

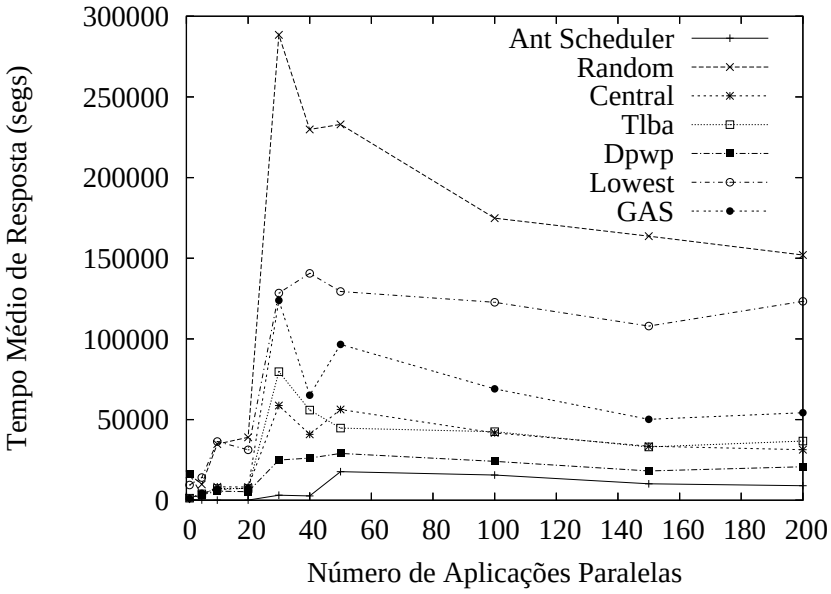

Figura 5. Simulação com 128 tarefas

Para avaliar os resultados aplica-se a equação 6 para definir o erro padrão s_x para pequenas amostras [W.C.Shefler 1988], onde: s é o desvio padrão e n o número de amostras, neste caso 10. Aplicando a equação obtém-se o erro padrão para o sem o Ant Scheduler igual a $0,51$ e $2,775$ com Ant Scheduler.

$$s_x = \frac{s}{\sqrt{n}} \tag{6}$$

A partir do erro padrão pode-se propor o teste de hipótese abaixo, para avaliar se os resultados são estatisticamente equivalentes:

Tabela 1. Resultados experimentais

Experimento	sem Ant Scheduler	com Ant Scheduler
1	351.00	327.00
2	351.00	336.00
3	351.00	318.00
4	354.00	321.00
5	351.00	318.00
6	351.00	333.00
7	351.00	321.00
8	351.00	336.00
9	348.00	309.00
10	348.00	315.00
Média	350.70	323.40
Desvio Padrão	1.615	8.777

Tabela 2. Teste de Hipótese

$$H_0 : \mu_{with} = \mu_{without}$$
$$H_A : \mu_{with} < \mu_{without}$$

É adotado o nível de significância para teste de uma cauda onde $\alpha = 0.0005$, onde: H_0 é a hipótese a ser avaliada; H_A é a hipótese alternativa, caso H_0 seja rejeitada; μ_{ca} é a média dos resultados com Ant Scheduler; μ_{sa} é a média dos resultados sem o Ant Scheduler. Para esse α pode-se encontrar em tabelas de valores para a distribuição *t-student* um limite de $4,781$ desvios padrão para que haja diferença estatística significante entre os dois conjuntos de dados.

Aplicando a equação 7 pode-se encontrar o número de desvios padrão igual a $9,83$ o que confirma que os resultados são estatisticamente significantes para $p < 0.005$, permitindo rejeitar a hipótese H_0. Portanto H_A é válida e o sistema com Ant Scheduler apresenta melhores resultados que um ambiente openMosix tradicional.

$$t = \frac{\mu_{without} - \mu_{with}}{s_x} \tag{7}$$

Aplicando ferramentas estatísticas[5] no conjunto de dados é possível encontrar o mais preciso $\alpha = 0.0000018$ para um teste de uma cauda. Nesse caso, H_A pode ser considerada verdadeira em $9.999.982$ de $10.000.000$ de execuções, o que confirma que a utilização do Ant Scheduler baixa o tempo de resposta. Somente em 18 dessas execuções os resultados do Ant Scheduler e do openMosix não apresentam diferenças estatísticas significativas.

6. Conclusões

Neste trabalho, os autores propoem uma nova abordagem para o balanceamento de carga em clusters de computadores utilizando um algoritmo baseado em ACO. A abordagem

[5]foi utilizado a ferramenta Gnumeric nessa análise – uma ferramenta livre licensiada na GNU/GPL.

proposta foi motivada pelos sucessos obtidos por esta técnica em vários problemas reais.

Este algoritmo, chamado Ant Scheduler, tem seu desempenho comparado com cinco outros algoritmos descritos na literatura. Os resultados da simulação, onde o algoritmo proposto apresentou um desempenho superior ao dos outros algoritmos investigados, sugere o potencial do Ant Scheduler para ambientes heterogêneos de clusters de computadores. Esses resultados motivaram a implementação para a validação dos conceitos teóricos. O algoritmo foi implementado num ambiente Linux real. A implementação foi avaliada através de experimentos e comparada com o algoritmo padrão de escalonamento do openMosix. Nesses experimentos, o Ant Scheduler apresentou bons resultados.

7. Agradecimentos

Os autores agradecem à FAPESP (processo número 2004/02411-9).

Referências

Acan, A. (2004). An external memory implementation in ant colony optimization. In *Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence*, pages 73–82, Brussels, Belgium.

Amir, Y. (2000). An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Transactions on Parallel and Distributed Systems*, 11(7):760–768.

Amir, Y., Awerbuch, B., Borgstrom, R. S., Barak, A., and Keren, A. (2000). An opportunity cost approach for job assignment and reassignment in a scalable computing cluster. *IEEE Transactions on Parallel and Distributed Systems*, 11:760,768.

Araújo, A. P. F., Santana, M. J., Santana, R. H. C., and Souza, P. S. L. (1999a). DPWP: A new load balancing algorithm. In *ISAS'99*, Orlando, U.S.A.

Araújo, A. P. F., Santana, M. J., Santana, R. H. C., and Souza, P. S. L. (1999b). A new dynamical scheduling algorithm, international conference on parallel and distributed processing techniques and applications - pdpta'99. Las Vegas,Nevada, U.S.A.

Choe, T. and Park, C. (2002). A task duplication based scheduling algorithm with optimality condition in heterogeneous systems. In *International Conference on Parallel Processing Workshops*, pages 531–536. IEEE Computer Society.

Culler, D. E., Karp, R. M., Patterson, D. A., S., A., Schauser, K. E., Santos, E., Subramonian, R., and von Eicken, T. (1993). LogP: Towards a realistic model of parallel computation. In *Principles Practice of Parallel Programming*, pages 1–12.

Daniel Merkle, Martin Middendorf, H. S. (2000). Ant colony optimization for resource-constrained project scheduling. In Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., and Beyer, H.-G., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 893–900, Las Vegas, Nevada, USA. Morgan Kaufmann.

de Mello, R. F. and Senger, L. J. (2006). Model for simulation of heterogeneous high-performance computing environments. In *7th International Conference on High Performance Computing in Computational Sciences – VECPAR 2006*, page 11. Springer-Verlag.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:1,13.

et. al, R. F. M. (2002). Analysis on the significant information to update the tables on occupation of resources by using a peer-to-peer protocol. In *16th Annual International Symposium on High Performance Computing Systems and Applications*, Moncton, New-Brunswick, Canada.

Feitelson, D. G. and Jette, M. A. (1997). Improved utilization and responsiveness with gang scheduling. In Feitelson, D. G. and Rudolph, L., editors, *Job Scheduling Strategies for Parallel Processing*, pages 238–261. Springer. Lect. Notes Comput. Sci. vol. 1291.

Hockney, R. W. (1996). *The Science of Computer Benchmarking*. Soc for Industrial & Applied Math.

Hsu, T. (2000). Task allocation on a network of processors. *IEEE Transactions on Computers*, 49(12):1339–1353.

Krueger, P. and Livny, M. (1987). The diverse objectives of distributed scheduling policies. In *Seventh Int. Conf. Distributed Computing Systems*, pages 242–249, Los Alamitos, California. IEEE CS Press.

Mello, R. F., Trevelin, L. C., Paiva, M. S., and Yang, L. T. (2004a). Comparative analysis of the prototype and the simulator of a new load balancing algorithm for heterogeneous computing environments. In *International Journal of High Performance Computing and Networking*, volume 1, No.1/2/3, pages 64–74. Interscience.

Mello, R. F., Trevelin, L. C., Paiva, M. S., and Yang, L. T. (2004b). Comparative study of the server-initiated lowest algorithm using a load balancing index based on the process behavior for heterogeneous environment. In *Networks, Software Tools and Application, ISSN 1386-7857*. Kluwer.

Mitzenmacher, M. (2001). The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104.

Santos, R. R. (2001). Escalonamento de aplicações paralelas: Interface amigo-corba. Master's thesis, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo.

Senger, L. J., de Mello, R. F., Santana, M. J., Santana, R. H. C., and Yang, L. T. (2005). Improving scheduling decisions by using knowledge about parallel applications resource usage. In *Proceedings of the 2005 International Conference on High Performance Computing and Communications (HPCC-05)*, pages 487–498, Sorrento, Naples, Italy.

Shivaratri, N. G., Krueger, P., and Singhal, M. (1992). Load distributing for locally distributed systems. *IEEE Computer*, 25(12):33–44.

Shmygelska, A. and Hoos, H. H. (2005). An ant colony optimisation algorithm for the 2d and 3d hydrophobic polar protein folding problem. *BMC Bioinformatics*, pages 1–22.

Sinha, P. K. (1997). *Distributed Operating Systems: Concepts and Design*. John Wiley & Sons.

Sivasubramaniam, A., Singla, A., Ramachandran, U., and Venkateswaran, H. (1994). An approach to scalability study of shared memory parallel systems. In *Measurement and Modeling of Computer Systems*, pages 171–180.

Souza, P. S. L. (2000). *AMIGO: Uma Contribuição para a Convergência na Área de Escalonamento de Processos*. PhD thesis, IFSC-USP.

Theimer, M. M. and Lantz, K. A. (1989). Finding idle machines in a workstation-based distributed system. *IEEE Transactions on Software Engineering*, 15(11):1444–1458.

Vetter, J. S. and Mueller, F. (2003). Communication characteristics of large-scale scientific applications for contemporary cluster architectures. *J. Parallel Distrib. Comput.*, 63(9):853–865.

W.C.Shefler (1988). *Statistics: Concepts and Applications*. The Benjamin/Cummings.

Zhou, S. and Ferrari, D. (1987). An experimental study of load balancing performance. Technical Report UCB/CSD 87/336, PROGRES Report N.o 86.8, Computer Science Division (EECS), Universidade da California, Berkeley, California 94720.