Acta Scientiarum. Technology
ISSN: 1806-2563
eduem@uem.br
Universidade Estadual de Maringá
Brasil

Valle Rego Gorino, Fabio; Martini, João Angelo; de Lara Gonçalves, Ronaldo Augusto
Estendendo o escalonamento de tarefas da biblioteca LAM/MPI
Acta Scientiarum. Technology, vol. 30, núm. 2, 2008, pp. 125-133
Universidade Estadual de Maringá
Maringá, Brasil

DOI: 10.4025/actascitechnol.v30i2.5461

Estendendo o escalonamento de tarefas da biblioteca LAM/MPI

Fabio Valle Rego Gorino, João Angelo Martini e Ronaldo Augusto de Lara Gonçalves*

*Programa de Pós-graduação em Ciência da Computação, Departamento de Informática, Universidade Estadual de Maringá, Avenida Colombo, 5790, 87020-900, Maringá, Paraná, Brasil. *Autor para correspondência. E-mail: ronaldo@din.uem.br*

RESUMO. O desempenho dos *clusters* de computadores está diretamente relacionado à forma como a carga de trabalho é distribuída entre os nós. Neste sentido, técnicas de escalonamento de tarefas são usadas para otimizar o uso dos recursos do sistema, aumentando o desempenho das aplicações paralelas. Este trabalho apresenta uma extensão da biblioteca LAM/MPI (*Local Area Multicomputer MPI*), que permite escalonar processos de forma mais eficiente do que o método original. Testes realizados em três aplicações MPI, em um *cluster* de oito nós, mostraram que nossos mecanismos podem reduzir o tempo de execução em mais de 50% em várias situações.

Palavras-chave: escalonamento de tarefas, aplicações distribuídas, LAM/MPI.

ABSTRACT. Extending the task scheduling of the LAM/MPI library. Computer cluster performance is directly related to how the workload is distributed among its nodes. Task scheduling techniques are employed to optimize the usage of system resources, thus improving the performance of parallel applications. This work presents an extension to the LAM/MPI library that allows process scheduling to be done more efficiently than in its original implementation. Tests performed with three MPI applications, running on an 8-node cluster, show that our solutions can reduce execution time by over 50% in most cases.

Key words: task scheduling, distributed applications, LAM/MPI.

Introdução

Clusters de computadores são amplamente usados na execução de aplicações que requerem grande esforço computacional para tratar e analisar grandes volumes de dados provenientes da geografia, climatologia, meteorologia, física quântica, mecânica de fluidos, biologia molecular e diversas outras áreas. O barateamento dos equipamentos computacionais, aliado ao potencial de processamento que estes equipamentos podem prover em conjunto, permitiu a rápida expansão no uso destes sistemas.

Equipamentos simples, que já não atendem mais às necessidades das aplicações atuais, podem ser interconectados em um sistema de processamento paralelo e distribuído, a um custo bem menor do que aquele necessário para a aquisição de um computador paralelo de grande porte. Quanto maior o número de processadores conectados maior o potencial de processamento.

Entretanto, quando os nós do *cluster* não são utilizados com cargas de trabalho condizentes com a disponibilidade de recursos e a capacidade de processamento, o desempenho pode ser prejudicado. Fatores como o número de usuários que utilizam o sistema e os tipos de aplicações que são executadas pode complicar a dinâmica de funcionamento de um *cluster*. Nesse sentido, técnicas de escalonamento de tarefas são extremamente recomendadas.

Atualmente, a LAM/MPI (*Local Area Multicomputer MPI*) é uma das plataformas de softwares mais usadas como infra-estrutura básica de programação e execução paralela e distribuída em *clusters*. Todavia, o escalonamento de tarefas-padrão da plataforma LAM/MPI é feito por um algoritmo simples em estilo *round-robin* e não tira proveito das oscilações de carga e processamento.

O presente trabalho propõe uma extensão da biblioteca LAM/MPI, permitindo que ela realize o escalonamento de tarefas de forma mais eficiente que a original. Três soluções algorítmicas são apresentadas, as quais são baseadas no uso da CPU e no poder computacional. Experimentos, em um *cluster* de oito nós na execução de diferentes aplicações, incluindo de reconhecimento de seqüências genéticas, mostram que os resultados são significativos na redução do tempo de execução destas aplicações.

Nas seções seguintes, este artigo conceitua o escalonamento de tarefas, contextualiza o estado da arte, cita algumas das principais pesquisas relacionadas ao tema, descreve os materiais e métodos, apresenta os resultados e discussão e finaliza com as conclusões e referências.

Escalonamento de tarefas

O escalonamento de tarefas (Casavant e Kuhl, 1988) é uma alternativa para o balanceamento de carga, podendo ser classificado como local ou global, estático ou dinâmico e centralizado ou distribuído, dependendo do ponto de vista considerado. Em sistemas de vários processadores e tarefas de diferentes localizações, tal como *clusters*, o escalonamento global pode ser utilizado para minimizar o tempo de conclusão da aplicação (Chau e Fu, 2003).

O escalonamento deve primar para que as necessidades individuais de cada tarefa e das comunicações entre elas sejam satisfeitas com o máximo aproveitamento da capacidade total de recursos do sistema. Os principais custos associados são os de processamento e comunicação. As tarefas *CPU-bound* tendem a ser associadas a processadores com maior capacidade de processamento disponível para finalizarem primeiro, enquanto que as tarefas *I/O-bound* tendem a ser alocadas em um mesmo processador para reduzir *overhead* de comunicação. De uma forma geral, o escalonamento de tarefas tende a encontrar o melhor compromisso entre estes dois custos.

A classificação de escalonamento estático ou dinâmico reflete a flexibilidade das regras de escalonamento em função do estado atual do sistema, podendo ser aplicado local ou globalmente. O escalonamento estático não considera as alterações no estado do sistema, enquanto o escalonamento dinâmico detecta alterações no estado do sistema e gerencia os processos com base no estado atual (Baumgartner e Wah, 1998). No escalonamento global estático, as tarefas são alocadas antes da execução. Já no escalonamento global dinâmico, as tarefas são alocadas em tempo de execução em função das condições do sistema, podendo migrar entre processadores para melhorar o desempenho final da execução.

A classificação de escalonamento centralizado ou distribuído é típica do escalonamento global. O escalonamento centralizado estabelece um único nó do sistema para decidir como, quando e onde distribuir as tarefas. Já no escalonamento distribuído, qualquer nó do sistema pode tomar estas decisões. O escalonamento centralizado é mais simples de implementar, embora seja mais susceptível a falhas. Por outro lado, no escalonamento distribuído, a sobrecarga de comunicação pode reduzir o benefício do escalonamento de tarefas.

Outra forma de classificação do escalonamento de tarefas considera a origem da iniciativa para a distribuição da carga, conforme referenciado em Gonçalves *et al.* (1996). Na abordagem "emissor inicia", um nó sobrecarregado procura por outro nó menos carregado para enviar a ele parte da sua carga. Na abordagem "receptor inicia", um nó menos carregado procura por outro nó sobrecarregado para solicitar parte da carga dele. Alguns sistemas podem trabalhar com as duas abordagens simultaneamente.

O escalonamento de tarefas também pode ser classificado quanto à sua adaptabilidade às alterações de certos parâmetros do sistema (Casavant e Kuhl, 1988). Em resposta às mudanças no sistema, o escalonador pode ignorar ou reduzir a importância de algum parâmetro e em função disto adaptar ou não o seu algoritmo.

O escalonamento de tarefas também pode ser classificado (Lling *et al.*, 1991) de acordo com duas características em adição àquelas já discutidas anteriormente. A primeira leva em consideração de onde obter informações para decidir sobre manter ou migrar um processo. A decisão pode ser baseada em informações locais ou globais. Uma decisão baseada em informações locais considera apenas o status de carga do próprio nó. Uma decisão baseada em informações globais considera o status de carga de um subconjunto ou de todos nós do sistema. A outra característica se preocupa com a migração da carga para outro processador com o intuito de balancear a carga. Se a migração ocorre apenas para os vizinhos diretos, então o espaço de migração é local, caso contrário, é considerado global.

Trabalhos relacionados

Muitos trabalhos têm sido desenvolvidos para o escalonamento de tarefas em *clusters*. Em 1997, Decker (1997) apresentou uma ferramenta para distribuir e balancear automaticamente as aplicações paralelas. Logo depois, George (1999) focou na distribuição de carga baseada no tempo de resposta dos nós quando da submissão de programas de testes pequenos e rápidos. Ainda neste mesmo ano, Bohn e Lamont (1999) desenvolveram técnicas de escalonamento considerando assimetria e prioridade.

No ano de 2002, pelo menos três trabalhos se destacaram. Ibrahim e Lu Xinda (2002) avaliaram o desempenho de sistemas paralelos que usam o escalonamento de tarefas dinâmico, em termos de tempo de execução, velocidade e eficiência, na execução de uma versão paralela do algoritmo de pesquisa *depth-first* sobre a plataforma MPI. Kacer e Tvrdík (2002) investigou a adequabilidade do escalonamento de tarefas pequenas que usam intensivamente o processador, tais como compiladores e utilitários de compressão, e propôs uma técnica eficiente de execução remota. Já o

trabalho de Shen *et al.* (2002) focou as políticas de escalonamento de tarefas para serviços de rede de granularidade fina.

Nos anos seguintes, Choi *et al.* (2003) investigaram o escalonamento focado nas tarefas que efetivamente afetam o desempenho do sistema e propôs uma nova métrica de carga. Drozdowski e Wolniewicz (2003) apresentaram um modelo de escalonamento para aplicações que podem ser quebradas em partes de tamanho arbitrário. Attiya e Hamam (2004) propuseram um algoritmo ótimo, executado em duas fases baseadas nas heurísticas *Simulated Annealing* e *Branch-and-Bound*. Savvas e Kechadi (2004) propuseram um algoritmo de escalonamento que divide recursivamente o *cluster* em subespaços a fim de encontrar a dimensão que melhor provenha desempenho.

Escalonamento de tarefas na LAM/MPI

A implementação LAM/MPI (*Local Area Multicomputer MPI*) (Lusk, 2001; Nguyen e Pierre, 2001) é um software de código aberto que se emprega bem em ambientes heterogêneos. Além disso, ela possui uma ativa comunidade de usuários e o seu grupo de desenvolvedores está disponível para auxiliar na solução de problemas.

A LAM/MPI utiliza um pequeno *daemon* (*lamd*) de nível de usuário para o controle de processos, redirecionamento de mensagens e comunicação entre processos. Este *daemon* é carregado no início de uma sessão, através do aplicativo *lamboot* que pode utilizar diversas tecnologias de execução de comandos remotos como *rsh/ssh*, TM (OpenPBS/PBS Pro), SLURM ou BProc, para realizar a operação de carga. O *daemon* deve ser executado em cada nó do *cluster* (Martins Jr. e Gonçalves, 2005). A aplicação é executada com o comando *mpirun*.

Uma aplicação na LAM/MPI é composta por diversos processos, sendo comum cada processo atuar sobre uma parcela diferente dos dados do problema a ser resolvido. Algumas aplicações, como aquelas usadas no presente trabalho, são organizadas no modelo mestre/escravos. Neste modelo, um dos computadores do *cluster*, conhecido como mestre, divide e distribui os dados do problema aos demais computadores, chamados de escravos, que executam suas parcelas da computação e enviam os resultados parciais ao mestre, que contabiliza o resultado global da aplicação. Neste sistema, o escalonamento de tarefas visa distribuir os processos de forma mais coerente com a disponibilidade de recursos e capacidade de processamento de cada nó do *cluster*. Sem o escalonamento de tarefas, o tempo que o usuário deve aguardar para finalizar a aplicação será diretamente proporcional ao tempo do nó mais lento.

A implementação padrão da biblioteca LAM/MPI escalona os processos usando o algoritmo *round-robin* sobre uma estrutura de dados vetorial que contém informações sobre os nós (lista de nós), onde serão encaminhados os processos da aplicação. O algoritmo visita a lista de nós em *round-robin* e distribui um processo por vez para o nó correntemente visitado. Nenhum tipo de reordenação especial sobre a lista de nós é utilizado, e nenhuma restrição quanto à decisão de alocação de um processo para determinado nó é imposta.

Material e métodos

Antes de os algoritmos propostos neste trabalho serem apresentados, alguns conceitos e esclarecimentos se fazem necessários. A carga de CPU indica a média de uso de CPU, a qual é representada por um valor real maior ou igual a zero. Aplicando este conceito, um valor próximo de 1, em qualquer intervalo escolhido, indica que em torno de um processo esteve em execução usando 100% de CPU durante o referido intervalo.

O poder de processamento indica a capacidade de processamento livre das interferências das aplicações e usuários, obtido durante a carga do sistema operacional. Trata-se de um valor constante que independe de *benchmark* e pode ser obtido pela execução isolada de um laço de repetição com nenhum código em seu interior. O número de processos ativos indica o número de processos que efetivamente está sendo executado, em média, durante determinado tempo. Processos que fazem frequentemente entrada/saída pouco contribuem para este índice. Este valor é um número natural muito próximo ao valor da carga de CPU.

Para a obtenção dos valores de carga de CPU e do poder de processamento dos nós do *cluster* foi utilizada a biblioteca *LibGTop*. Essa biblioteca permite coletar dados do sistema operacional, tais como as taxas de uso de CPU, da memória e informações sobre os processos em execução. Praticamente, todos sistemas baseados em Unix suportam a biblioteca *LibGTop*, inclusive o Linux, o qual é o ambiente de estudo deste trabalho. A função *glibtop_get_loadavg* retorna a taxa média da carga do processador, contabilizada dentro de certo intervalo de tempo decorrido. Além da taxa média de carga do processador, o valor em *bogomips* (unidade de poder de processamento) também é coletado e é utilizado como métrica de poder computacional. Portanto, o módulo de obtenção de carga retorna uma tupla na

forma (nó, carga, *bogomips*) de cada nó componente do *cluster*.

Algoritmos propostos

O presente trabalho propõe estender a biblioteca de programação paralela LAM/MPI, permitindo a ela realizar um escalonamento de tarefas mais eficiente do que o estilo *round-robin* padrão. Na prática, apesar de ainda ser aplicado o algoritmo *round-robin*, este trabalho propõe reordenar a lista de nós e impor restrições na alocação de processos resultando em três algoritmos de escalonamento diferentes.

Diversos módulos da biblioteca LAM/MPI foram modificados e outros inseridos, de forma a permitir modificar o escalonamento. Mais precisamente, o escalonamento foi modificado dentro do aplicativo *mpirun* (utilizado pelo usuário para iniciar a execução paralela), o qual é o responsável pela alocação de processos aos nós do *cluster* durante o disparo da execução da aplicação. Os algoritmos experimentados são descritos a seguir.

Algoritmo BAL1

O primeiro algoritmo, identificado por BAL1, usa a carga de CPU de cada nó para ordenar a lista de nós de forma crescente (do menos para o mais carregado). Com isso, durante o escalonamento *round-robin*, os nós menos sobrecarregados são atendidos em primeiro lugar (privilegiados). Ademais, o algoritmo BAL1 utiliza um limiar de carga para restringir as alocações. O nó cuja carga é superior ao limiar de carga não recebe processos enquanto houver outra possibilidade de escolha de um nó com carga inferior.

Estando todos os nós com carga superior ao fator militante, os processos restantes serão alocados para qualquer nó seguindo a ordem de visitação na lista de nós. O limiar de carga fixado empiricamente, nos experimentos com este algoritmo, foi de 0.8, mas pode ser reajustado dependendo da taxa média de carga de CPU do *cluster*.

Algoritmo BAL2

O segundo algoritmo, identificado por BAL2, trabalha com dois níveis de escalonamento. Em um primeiro nível, ele usa o poder de processamento de cada nó para ordenar de forma decrescente a lista de nós (do mais forte para o mais fraco). Durante o escalonamento *round-robin*, os nós com maior poder de processamento são atendidos em primeiro lugar.

Em um segundo nível, BAL2 também faz uso da carga de CPU e do limiar de carga para restringir a alocação de processos aos nós. Para uma mesma faixa de poder de processamento, os nós são atendidos tal como no algoritmo BAL1. Desta forma, os processos são distribuídos em ordem para aqueles que tiverem maior disponibilidade de carga dentre aqueles que têm maior poder de processamento. Estando todos os nós com carga superior ao limiar de carga, os processos restantes serão alocados para qualquer nó seguindo a ordem de visitação na lista de nós. O limiar de carga fixado empiricamente nos experimentos com este algoritmo também foi de 0.8. De uma forma geral, BAL2 é uma especialização sobre BAL1.

Algoritmo BAL3

O terceiro algoritmo, identificado por BAL3, também ordena a lista de nós de acordo com o poder de processamento em primeiro nível e de acordo com a carga de CPU em segundo nível; entretanto, usa um limiar de carga baseado no número de processos ativos ao invés da carga de CPU. Além disso, quando todos os nós estão acima do limiar de carga, o algoritmo distribui os processos de acordo com o poder de processamento relativo (poder de processamento dividido pelo número de processos).

De uma forma resumida, BAL3 funciona da seguinte forma. Primeiramente, somente os nós que não têm processos ativos (carga de CPU próxima de zero) recebem processos da aplicação. Quando todos os nós tiverem pelo menos um processo ativo, uma segunda fase do algoritmo é executada. Então, enquanto o número de processos ativos em determinado nó for menor que o limiar de carga, o algoritmo continua distribuindo os processos restantes, com a seguinte restrição: para nós lentos (com baixo poder de processamento), distribui processo apenas se já houver processos da aplicação sendo executados. Esta decisão foi tomada após análise empírica a qual mostra que a concorrência de processos de uma mesma aplicação é bem suportada por nós de baixa capacidade de processamento.

O limiar de carga de CPU fixado empiricamente nos experimentos com este algoritmo também foi de 1. Este número foi escolhido porque as aplicações executadas, neste trabalho, sempre continham oito processos e o *cluster* contém apenas oito nós. Assim, o limite de um processo por nó foi suficiente nos experimentos realizados, mas pode ser reajustado dependendo do número de processos e da quantidade de nós. Quando todos os nós atingem o limite do número de processos, então o algoritmo faz uso apenas do poder computacional relativo para distribuir os processos restantes.

Resultados e discussão

Para avaliar o desempenho dos algoritmos propostos nesta pesquisa, experimentos foram

realizados, mensurados e comparados com a implementação original da LAM/MPI. Três aplicações foram utilizadas: cálculo integral, multiplicação matricial e reconhecimento de DNA, todas sobre um *cluster* de quatro nós de processadores Pentium 4 de 1.8 GHz e 4 nós de processadores Pentium 4 HT de 3 GHz, todos com 512 MB de RAM e 1 MB de *cache*. Estes processadores são conectados por um *switch* 3COM padrão *fast ethernet*. Os códigos foram escritos em linguagem C e compilados em gcc 3.2.2. O ambiente utilizado foi o Linux *Red Hat* e *kernel* 2.4.29. A versão da biblioteca de troca de mensagens utilizada foi a LAM/MPI 7.1.1, e a extensão é baseada nesta versão.

A aplicação cálculo integral divide a área sob a curva de uma função *f(x)* em um conjunto de trapézios e distribui estes trapézios entre os processos da aplicação. O processo-mestre finaliza o cálculo somando os resultados parciais recebidos dos demais nós. A aplicação multiplicação matricial atua sobre duas matrizes A e B. Cada processo da aplicação recebe uma quantidade de linhas da matriz A e a matriz B completa. O processo-mestre finaliza a multiplicação colocando em C os resultados parciais recebidos dos demais processos.

A aplicação reconhecimento de DNA recebe como entrada a seqüência genética (DNA) de dois indivíduos. A primeira seqüência é replicada, na íntegra, para todos os processos da aplicação. A segunda seqüência é dividida, em partes iguais, de acordo com o número de processos, e cada segmento é transmitido para um processo diferente. No fim, uma operação de redução é executada e o valor (*score*) encontrado nos dois indivíduos é retornado.

Uma versão mais pesada da aplicação cálculo integral, denominada aqui de carga artificial, foi disparada de forma controlada e de acordo com o propósito desejado, variando em número e local de ativação para então forçar o desbalanceamento. Nos gráficos apresentados neste trabalho, "BALX" representa os resultados obtidos na LAM/MPI estendida com o escalonamento proposto e "ORIG" representa os resultados obtidos na LAM/MPI original. O "X", da sigla "BALX", representa a versão da extensão que está sendo mencionada naquele momento. Em todos os gráficos, o eixo x marca o número de nós que foram sobrecarregados com cargas artificiais e o eixo y o tempo total de execução da aplicação em segundos.

Avaliando BAL1

O algoritmo BAL1 foi experimentado com a aplicação cálculo integral. A Figura 1 ilustra os resultados. Pode-se observar que o escalonamento BAL1 melhora o desempenho da aplicação na maioria das situações. Quando a quantidade de nós sobrecarregados é menor, o escalonamento provê maior benefício, que diminui na medida em que maior número de nós se torna sobrecarregado. No melhor caso, quando apenas um nó é sobrecarregado, o escalonamento reduz o tempo de execução em torno de 51%.

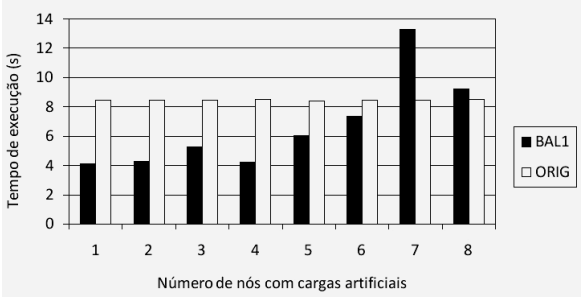

Figura 1. BAL1 no cálculo de integral.

A situação em que todos nós estão sobrecarregados é semelhante àquela em que não existe o desbalanceamento e, neste caso, a LAM/MPI estendida é pouco pior pelo *overhead* causado pela função adicional de obter as taxas de carga dos nós no início da computação, além daquele causado pela carga artificial. Mas este fato não é preocupante, pois tal *overhead* apresenta valor máximo constante para um mesmo *cluster*, independentemente da aplicação, sendo desprezível em muitos casos.

Uma observação importante é que o modelo ORIG quase não se afeta pelo desbalanceamento artificial. Este fato ocorre porque o tempo de duração de uma aplicação distribuída é basicamente imposto pelo processo mais lento, pois o término da aplicação deve necessariamente esperá-lo. Neste sentido, é indiferente se a aplicação tem que esperar um, dois ou todos processos terminarem, desde que ela necessariamente já tenha que esperar um processo mais lento.

Para verificar se os resultados obtidos não são específicos para uma única aplicação, outros experimentos foram também realizados com outra aplicação, a multiplicação matricial. Os resultados são apresentados na Figura 2. Nesta situação, o escalonamento também se mostrou bastante favorável e, no melhor caso, a redução do tempo de execução da aplicação atinge aproximadamente 59% quando quatro nós estão sobrecarregados. As outras situações possuem comportamentos parecidos com aqueles da Figura 1.

Nas Figuras 1 e 2, pode-se observar um prejuízo no desempenho quando sete nós são sobrecarregados com carga artificial, sendo bem mais expressivo na aplicação cálculo integral. Após a

análise da situação, constatou-se que a justificativa para o ocorrido é a seguinte: estando apenas um nó com pouca carga, uma quantidade muito maior de processos é alocada sobre este nó e considerando a capacidade de processamento das máquinas e a complexidade dos processos, o escalonamento não é matematicamente viável para esta situação. Provavelmente, se fossem mudados os processadores do *cluster*, o número de nós e o número de processos, outra situação se configuraria. De uma forma mais prática, este acúmulo poderia ser evitado se o limiar de carga fosse outro.

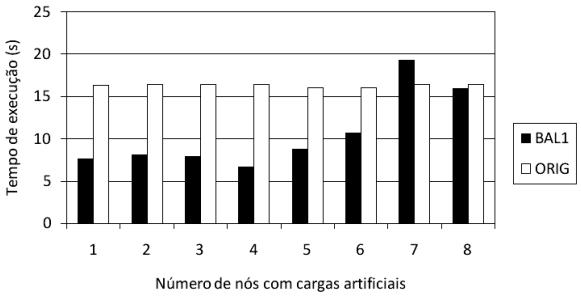

Figura 2. BAL1 na multiplicação matricial.

Observe ainda nas Figuras 1 e 2 que o ganho é alto até quando quatro nós estão sobrecarregados e, a partir daí, o ganho passa a diminuir, tornando-se inclusive negativo conforme já mencionado. Este fato está relacionado com a forma pela qual é feita a sobrecarga artificial do sistema. Nos experimentos realizados nesta pesquisa, as sobrecargas foram primeiramente aplicadas nas estações lentas (1.8 GHz) e posteriormente nas estações rápidas (3 GHz). Assim, quando cinco ou mais nós são sobrecarregados, a sobrecarga artificial passa a atingir os nós de maior capacidade de processamento, gerando queda no desempenho. O ganho médio em ambas as aplicações, em termos de redução no tempo de execução, é visto na Figura 3.

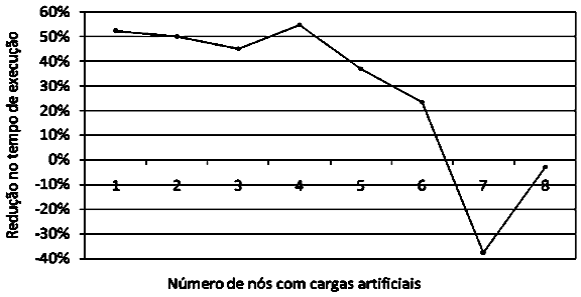

Figura 3. Ganho médio do BAL1.

Avaliando BAL2

Os resultados apresentados na Figura 4 ilustram um comparativo entre a segunda versão e a versão original. O algoritmo reduz o tempo de execução em diferentes situações, alcançando mais de 63% de redução quando um nó é sobrecarregado e mais de 50% quando dois ou quatro nós estão sobrecarregados. Apesar de prover melhorias na maioria das situações, pode-se observar ainda o problema quando sete nós estão sobrecarregados. Enquanto o algoritmo encontrar um nó com índice de carga inferior ao limiar, este nó receberá toda a nova carga.

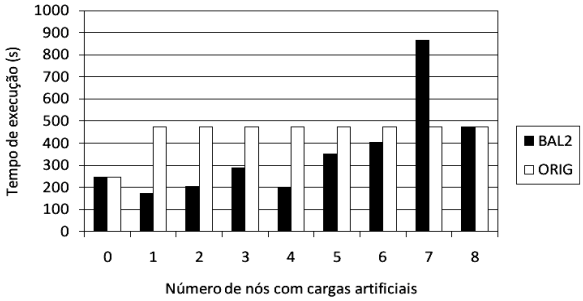

Figura 4. BAL2 no reconhecimento de DNA.

É possível adequar o algoritmo para que, em situações nas quais o número de nós disponíveis sem sobrecarga é limitante, distribua os processos da forma original. Testou-se esta variação quando o número de nós ociosos foi inferior a 25% e os resultados são apresentados na Figura 5. De fato, os desempenhos foram melhores, e o "ponto fora da curva" foi removido. Entretanto, não se pode afirmar que 25% seja o valor ideal da situação limite.

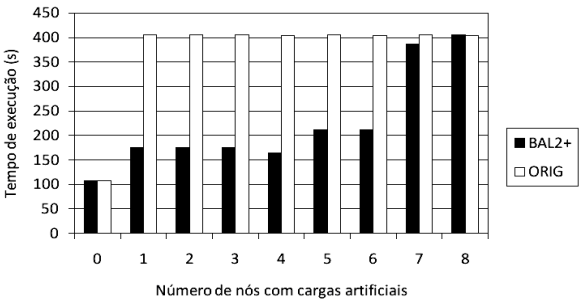

Figura 5. BAL2$^+$ no reconhecimento de DNA.

Nos experimentos realizados, observou-se que alterações no limiar de carga podem influenciar o desempenho da aplicação para melhor ou pior. Neste sentido, o ideal seria um limiar auto-ajustável. Por mais que o limiar fosse adotado como um parâmetro passado pelo administrador do *cluster*, que conhece a utilização do mesmo, o limiar se torna inútil quando os nós apresentam carga superior ao limiar.

Outro ponto detectado é que basear a tomada de decisão no limiar (média de carga) despreza a diferença de poder computacional dos nós, podendo gerar, no longo prazo, estações lentas sobrecarregadas e estações rápidas ociosas. Tal situação pode ocorrer quando todos os nós apresentarem a média de carga de CPU superior ao limiar; então a distribuição de carga respeitará a ordem decrescente do poder computacional. Mas se as aplicações disparadas solicitarem um número de processos igual ao número de nós, todos os nós receberão novas cargas em igual quantidade.

Avaliando BAL3

A Figura 6 apresenta o desempenho do algoritmo BAL3 em função do número de cargas artificiais. O problema devido à sobrecarga do *cluster* foi reduzido, pois a distribuição dos novos processos respeita a relação poder computacional/processo. Além disso, a técnica mostrou ser capaz de aproveitar a diferença de poder computacional existente entre os nós do *cluster*. Um resultado interessante foi o ganho de desempenho alcançado pela técnica de balanceamento com o *cluster* ocioso (ponto 0), alcançando mais de 57%.

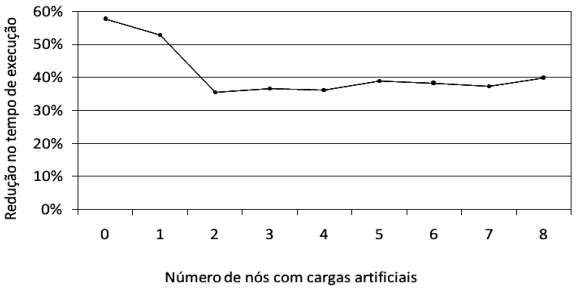

Figura 6. BAL3 no reconhecimento de DNA.

Este fato ocorre pelo seguinte. O processo-mestre é responsável por transmitir as seqüências genéticas para os demais processos. Na LAM/MPI ORIG, a ordem de distribuição dos processos respeita a ordem do arquivo de definição de nós (*boot schema*), a qual permite que o processo-mestre seja alocado a uma estação lenta. Este fato não acontece quando o poder computacional é considerado. Nos demais casos, o ganho se manteve entre 30 e 40%.

Para certificar o desempenho desta técnica, em situações imprevisíveis, uma nova bateria de experimentos foi realizada utilizando um gerador de carga aleatório. O gerador de carga tem como funcionalidade básica definir aleatoriamente quais nós receberão carga artificial durante a execução da aplicação de DNA. Os algoritmos BAL2 e BAL3, por serem os melhores, foram experimentados e comparados com a versão original, conforme ilustra a Figura 7.

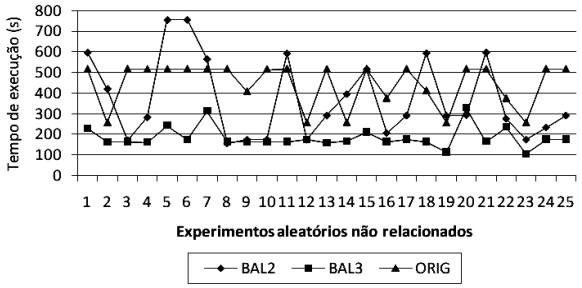

Figura 7. BAL3 com carga aleatória.

Pode-se afirmar que a técnica BAL3 apresentou excelentes resultados. A princípio, em nenhum momento a técnica BAL3 apresentou desempenho inferior ao comportamento-padrão da biblioteca LAM/MPI. O mesmo não pode ser afirmado a respeito da técnica BAL2. Isso acontece porque caso os nós ocupados com a carga artificial sejam as estações rápidas, toda nova aplicação MPI será alocada para as estações lentas. No caso da técnica BAL3, as estações rápidas serão verificadas e, se possível, estas também receberão os processos da nova aplicação. Essa propriedade permite que a técnica BAL3 atinja um desempenho médio 50,2% superior, se comparada com a técnica BAL2. O desempenho da técnica BAL3 chega a ser 57% superior ao modo de distribuição padrão da biblioteca LAM/MPI. A técnica BAL2 apresenta um ganho de desempenho médio de apenas 16,7% em relação ao ORIG.

Conclusão

O escalonamento de tarefas oferece a possibilidade de explorar os recursos de *cluster*s de forma transparente para o usuário da aplicação paralela. Este trabalho apresenta e avalia três diferentes algoritmos que levam em consideração a carga de CPU e o poder computacional dos nós como importantes fatores na alocação de tarefas. Algoritmos de escalonamento mais elaborados do que o estilo *round-robin* podem tirar proveito desses fatores, permitindo que as aplicações tenham melhor desempenho em ambientes LAM/MPI para situações diversas de sobrecarga, alcançando ganhos superiores a 50% na redução do tempo de execução.

Os experimentos realizados também mostram que o uso de um limiar de carga ajuda a normalizar

as discrepâncias no balanceamento de carga e que o desempenho de aplicações MPI é fortemente dependente da configuração usada no *cluster* e da complexidade da aplicação. Os resultados obtidos não permitem definir uma política exata para o escalonamento de tarefas, mas mostram questões importantes que devem ser consideradas.

Os algoritmos propostos, nesta pesquisa, são estáticos e as oscilações no estado de uso dos recursos do *cluster* precisam ser consideradas em trabalhos futuros. Pela dificuldade de migrar processos, uma alternativa ao desenvolvimento de algoritmos dinâmicos será prever a variação de carga do sistema em função do histórico de carga anterior, decidindo onde alocar os processos, considerando que a carga se alterará no tempo, adequando a carga do *cluster* de acordo com a periodicidade e as tendências do seu uso. Outras questões que precisam ser trabalhadas envolvem a mensuração e análise do *overhead* de comunicação e o desenvolvimento de algoritmos para *clusters* homogêneos.

Agradecimentos

Agradecemos a Capes, pelo suporte financeiro dado ao projeto "Laboratório Multidisciplinar de Computação de Alto Desempenho, Serviços Web e Banco de Dados" no contexto do Edital 01/2007 (Pró-equipamentos).

Referências

ATTIYA, G.; HAMAM, Y. Two phase algorithm for load balancing in heterogeneous distributed systems. *In:* EUROMICRO CONFERENCE ON PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING (EUROMICRO-PDP), 12., 2004, La Coruña. *Anais...* Los Alamitos: IEEE CS Press, 2004. p. 434-439.

BAUMGARTNER, K.M.; WAH, B.W. A global load balancing strategy for a distributed computer system. *In:* INTERNATIONAL WORKSHOP ON THE FUTURE TRENDS IN DISTRIBUTED COMPUTING SYSTEMS IN THE 1990´S, 1., 1998, Hong Kong. *Anais…* Los Alamitos: IEEE CS Press, 1998. p. 93-102.

BEVILACQUA, A. Dynamic load balancing method on a heterogeneous cluster of workstations. *Informatica*, Slovenia, v. 23, n. 1, p. 49-56, 1999.

BOHN, C.A.; LAMONT, G.B. Asymmetric load balancing on a heterogeneous cluster of PCs. *In:* PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS (PDPTA), 5., 1999, Las Vegas. *Anais...* Las Vegas: CSREA Press, 1999. p. 2515-2522.

CASAVANT, L.; KUHL, J.G. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, New York, v. 14, n. 2, p. 141-154, 1988.

CHAU, S.C.; FU, A.W. Load balancing between computing clusters. Parallel and Distributed Computing, Applications and Technologies. *In:* INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING, APPLICATIONS AND TECHNOLOGIES, 4., 2003, Chengdu. *Anais...* Los Alamitos: IEEE CS Press, 2003. p. 548-551.

CHOI, M.I. *et al.* Improving performance of a dynamic load balancing system by using number of effective tasks Cluster Computing. *In:* IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING (CLUSTER), 1., 2003, Hong Kong. *Anais...* Los Alamitos: IEEE CS Press, 2003. p. 436-441.

DECKER, T. Virtual Data Space: a universal load balancing scheme. *In:* INTERNATIONAL SYMPOSIUM ON PARALLEL ALGORITHMS FOR IRREGULARLY STRUCTURED PROBLEMS (IRREGULAR), 4., 1997, Paderborn. *Anais...* Holland: Springer, 1997. p. 159-166.

DROZDOWSKI, M.; WOLNIEWICZ, P. Out-of-core divisible load processing. *IEEE Trans on Parallel and Distributed Systems*, Los Alamitos, v. 14, n. 10, p. 1048-1056, 2003.

GEORGE, W. Dynamic load-balancing for data-parallel MPI programs. *In:* MESSAGE PASSING INTERFACE DEVELOPER'S AND USER'S CONFERENCE (MPIDC), 1., 1999, Atlanta. *Anais...* Cambridge: MIT Press, 1999. p. 95-100.

GONÇALVES, R.A.L. *et al.* Proposta de uma Ferramenta para Elaboração e Monitoração de Aplicações Distribuídas. *In:* SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES), 10., 1996, São Carlos. *Anais...* São Carlos: UFSCar, 1996. p. 275-290.

IBRAHIM, M.A.M.; LU XINDA. Performance of dynamic load balancing algorithm on cluster of workstations and PCs. *In:* INTERNATIONAL CONFERENCE ON ALGORITHMS AND ARCHITECTURES FOR PARALLEL PROCESSING (ICA3PP), 5., 2002, Beijing. *Anais…* Washington, D.C.: IEEE CS Press, 2002. p. 23-25.

KACER, M.; TVRDÍK, P. Load balancing by remote execution of short processes on linux clusters. *In:* IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID (CCGRID), 2., 2000, Berlin. *Anais…* Washington, D.C.: IEEE CS Press, 2002. p. 276-276.

LLING, R. *et al.* A study of dynamic load balancing algorithms. *In:* IEEE SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING (SPDP), 3., 1991, Dallas. *Anais…* Los Alamitos: IEEE CS Press, 1991. p. 686-689.

LUSK, E. Programming with MPI on clusters. *In:* IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING (CLUSTER), 1., 2001, Newport Beach. *Anais...* Washington, D.C.: IEEE CS Press, 2001. p. 360-362.

MARTINS JR., A.S.; GONÇALVES, R.A.L. Extensões na

LAM/MPI para automatizar o checkpoint e tolerar falhas em cluster de computadores. *In:* WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (WSCAD), 5., 2005, Rio de Janeiro. *Anais...* Rio de Janeiro: SBC/UFRJ, 2005. p. 129-136.

NGUYEN, V.A.K.; PIERRE, S. Scalability of computer clusters. *In:* CANADIAN CONFERENCE ON ELECTRICAL AND COMPUTER ENGINEERING. 1., 2001, Toronto. *Anais...* Los Alamitos: IEEE CS Press, 2001. p. 405-409.

SAVVAS, I.K.; KECHADI, M.T. Dynamic task scheduling in computing cluster environments. *In:* Parallel and Distributed Computing. In: International Symposium on/Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC), 3., 2004, Cork. *Anais...* Washington, D.C.: IEEE CS Press, 2004. p. 372-379.

SHEN, K. *et al.* Cluster load balancing for fine-grain network services. *In:* INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM (IPDPS), 16., 2002, Fort Lauderdale. *Anais...* Washington, D.C.: IEEE CS Press, 2002. p. 51-58.

Received on October 02, 2007.
Accepted on August 25, 2008.