1

Introdução

O grande aumento de performance das redes de computadores, combinado com a proliferação de computadores de baixo custo e alto desempenho, trouxe à tona ambientes de meta-computação, ou *grids*[15, 16]. Estes ambientes podem combinar milhares de computadores de centenas de domínios diferentes, conectados através de redes locais ou pela rede mundial. A grande heterogeneidade deste tipo de ambiente, somada a muitas restrições de segurança (tanto em termos de acesso, como de comunicação entre processadores), vêm impulsionando o estudo de um tipo especial de tarefa, as chamadas tarefas divisíveis.

O conceito de tarefas divisíveis foi introduzido originalmente por Robertazzi et al. [11] e tem sido bastante estudado nos últimos anos, sendo mais amplamente difundido por Bharadwaj, Ghose, Mani e Robertazzi [5]. Uma tarefa divisível é caracterizada por poder ser dividida arbitrariamente em um número qualquer de frações de carga, podendo cada parte ser processada em paralelo, sem restrições de precedência. Este modelo permite inclusive uma divisão fracionária da carga, o que simplifica o seu estudo sem reduzir muito sua generalidade, uma vez que pressupõe-se uma alta granularidade de dados.

Muitas aplicações modernas podem ser modeladas como tarefas divisíveis. Um exemplo é o caso da busca por um padrão de imagem num grande banco de imagens, onde o tamanho de uma imagem é muitíssimo inferior ao tamanho do banco inteiro. Este banco poderia ser dividido em partes com a granularidade de uma imagem, enviadas para cada processador efetuar o casamento de padrões. As tarefas executadas em cada processador podem ser realizadas em paralelo, sem restrições de precedência. Neste caso, devido à grande desproporção entre o tamanho de uma única imagem e o tamanho do banco de dados, pode-se modelar o problema como o escalonamento de uma tarefa divisível. Além deste exemplo, existe um grande espectro de problemas que podem ser tratados utilizando-se o conceito de tarefas divisíveis, tais como

o processamento de imagens e vídeo em *hardware* [2], problemas de álgebra linear [6], simulações [1] e outros problemas científicos e de engenharia [9]. Além de poder ser aplicado a um grande número de problemas, este modelo de carga divisível fornece uma maneira simples, porém realista, para o mapeamento de tarefas independentes em plataformas heterogêneas. Em [13], por exemplo, a aderência do modelo às situações reais é estudada e bons resultados são obtidos para algumas das aplicações apresentadas.

Neste trabalho, assim como nos demais trabalhos relacionados ao escalonamento de tarefas divisíveis, considera-se apenas *grids* dedicados, ou seja, sistemas heterogêneos inteiramente dedicados à execução da tarefa divisível. Ao contrário de *grids* não-dedicados, onde pode existir grande flutuação no poder computacional disponível em cada processador, em *grids* dedicados tem-se um cenário mais estável. Isto permite a adoção de um modelo de sistema onde o poder computacional dos processadores é invariável.

Estudando-se o problema de escalonamento de tarefas divisíveis em redes estrela, verificou-se na literatura modelos de programação não-linear inteira mista, métodos ótimos de escalonamento para casos especiais, bem como algumas heurísticas. Neste trabalho, desenvolveram-se novos modelos de programação linear inteira mista, além de elaborar-se algumas desigualdades válidas para o problema. Para um caso especial, desenvolveu-se um novo algoritmo capaz de encontrar o escalonamento ótimo com ordem de complexidade de execução inferior ao algoritmo existente na literatura. Elaborou-se ainda neste trabalho heurísticas eficientes que apresentaram ótimos resultados experimentais. Experimentos estes que foram conduzidos com várias das técnicas desenvolvidas e algumas das existentes na literatura, confrontando-se os resultados obtidos.

1.1 Modelo de sistema

Os termos tarefa divisível e carga divisível serão usados com o mesmo sentido. O termo processador será utilizado para denotar uma unidade de processamento com memória, disco e dispositivo de rede. Os termos informação, dados e carga serão igualmente utilizados com o mesmo significado, sendo discretizados em unidades de informação (o que é dependente do problema tratado).

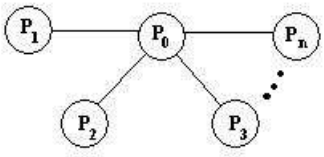

Figura 1.1: Rede estrela

Assim como em [3, 22], segue-se o paradigma mestre/escravo (ou mestre/trabalhador), onde n processadores escravos são referenciados como P_1, P_2, ..., P_n e o processador mestre como P_0. Inicialmente o mestre possui a carga total W (em unidades de informação, e.g. um milhão de imagens) a ser processada e é responsável por particioná-la em blocos e enviá-los para cada escravo processar.

Na literatura encontram-se modelos onde o processador mestre pode apenas enviar dados ou também participar da computação. Neste trabalho, optou-se pelo caso onde o processador mestre não participa da computação, sem perda de generalidade. Basta agregar-se mais um processador escravo com latência de comunicação nula e taxa de transferência de dados infinita para simular o outro caso.

Assim como na maioria dos trabalhos sobre escalonamento de tarefas divisíveis (e.g. [3, 21, 22]), será desconsiderada a necessidade de retorno de resultados das computações feitas pelos escravos. Esta opção é feita para simplificar a modelagem matemática e manter concisa a apresentação das técnicas sem que se perca muito a generalidade.

A topologia de interconexão dos processadores é do tipo estrela, formada por n enlaces que ligam o processador mestre P_0 aos demais, como exemplificado na Figura 1.1. Assume-se que o mestre utiliza a rede de forma seqüencial, ou seja, ele envia dados a no máximo um processador por vez, sem possibilidade de concorrência. Esta característica, comumente assumida nos trabalhos relacionados, pode ser justificada pela implementação no processador mestre ou por propriedades dos enlaces de rede (por exemplo, uma rede interconectada por comutadores).

Uma vez definido que o processador P_0 enviará uma quantidade de dados α (medida em unidades de informação, e.g. três imagens do banco) para o processador P_i, o enlace de comunicação será utilizado por um período de tempo $g_i + G_i\alpha_i$. A latência g_i (também chamada *startup time*) é definida

como sendo o tempo necessário para a inicialização da comunicação, isto é, um período de tempo fixo necessário para o mestre iniciar a transmissão de dados para o escravo P_i (e.g. estabelecimento de uma conexão TCP/IP e autenticação do processador mestre). O termo linear $G_i\alpha_i$ representa o tempo necessário para a transmissão dos dados, onde G_i é o inverso da taxa de transmissão do enlace entre P_0 e P_i. Ou seja, G_i é definido como sendo o tempo necessário para o envio de uma unidade de informação de P_0 para P_i (e.g. um segundo por imagem transmitida).

Após P_i receber todos os dados a serem processados, o tempo de execução dos α_i dados recebidos por P_i é caracterizado por $w_i\alpha_i$, onde w_i é o tempo necessário para o processamento de uma unidade de informação.

O modelo assume comunicação e computação concorrentes, isto é, um processador pode começar a receber novos dados enquanto processa outros recebidos anteriormente. Nota-se porém que um processador só pode executar uma tarefa por vez e que só pode começar a executar uma tarefa após tê-la recebido de forma integral. Salienta-se que, assim como nos demais trabalhos na literatura, o processamento necessário para o envio e recebimento de dados (e.g. empacotamento dos dados e conferência de integridade) é ignorado.

Dado, por exemplo, um banco de dados que possua um milhão de imagens ($W = 1.000.000$) a serem processadas por um sistema computacional composto por três processadores escravos (P_1, P_2 e P_3) ligados através de uma rede estrela ao processador mestre. Considerando-se que todos os processadores escravos são idênticos, precisando cada um de 10 ms para processar uma imagem ($w = 10$ ms), e possuam enlaces idênticos com latência $g = 1$ s e taxa de transmissão igual a uma imagem a cada 2 ms ($G = 2$ ms), pode-se aproximar este problema como sendo um problema de escalonamento de tarefas divisíveis. Uma solução possível para o problema aproximado é a divisão da carga total W (um milhão de imagens) em três frações de carga, $\alpha_1 = 395.697,8$ imagens a serem enviadas para P_1, $\alpha_2 = 329.664,84$ para P_2 e $\alpha_3 = 274.637,36$ para P_3, o que acarretaria em 4.749,374 segundos para que todas as imagens fossem analisadas. Como não é possível enviar apenas uma fração de imagem (o que poderia impedir o seu correto processamento), os resultados poderiam, por exemplo, ser arredondados para 395.698, 329.665 e 274.637, o que acarretaria em 4.749,376 segundos para que todas as imagens fossem analisadas. Nota-se que, para casos como este, o modelo de tarefas divisíveis é uma ótima aproximação e permite que se encontre um bom escalonamento para os dados.

1.2 Escalonamento de tarefas divisíveis

Com o objetivo de facilitar o estudo do comportamento de diferentes escalonadores, foi desenvolvido um módulo que permite a visualização do comportamento do sistema dada uma série de decisões de escalonamento. Os gráficos de Gantt gerados por este módulo, descrito em detalhe no Apêndice A, serão utilizados durante todo o trabalho.

No problema de escalonamento de tarefas divisíveis, tem-se como principal parâmetro de performance, o chamado *makespan*, que consiste no tempo de término do último processador que participa da computação da carga W. Sua minimização é o objetivo das técnicas aqui apresentadas.

É dito que um escalonamento é feito em um único período (ou rodada) se a tarefa divisível é particionada e, dada uma ordem de envio, cada processador P_i que participará do processamento recebe uma única parcela α_i numa única vez, $i = 1, \ldots, n$. No gráfico de Gantt da Figura 1.2, por exemplo, é apresentado um escalonamento ótimo onde P_6 foi o primeiro processador a receber carga, seguido por P_1, P_8, P_7 e assim por diante. Em contraste com este escalonamento ótimo, pode ser visto na Figura 1.3 um exemplo de escalonamento não ótimo. Como descrito no Apêndice A, as barras mais escuras indicam os períodos de comunicação entre P_0 e cada processador escravo. As barras mais claras indicam o período de processamento de cada processador escravo (que começa após o recebimento integral da carga). Nota-se que todos os processadores receberam carga uma única vez e terminaram de processá-las no mesmo instante de tempo. Enviando-se carga uma única vez para cada processador, perde-se pouco tempo com a latência de comunicação, porém os últimos processadores a receber dados ficam muito tempo ociosos.

Com o objetivo de aproveitar mais o paralelismo e reduzir este tempo ocioso dos processadores, são utilizadas técnicas com múltiplos períodos ou rodadas (*rounds* ou *installments*, como se encontra na literatura) de envio de dados. Isto pode vir a reduzir o tempo ocioso dos processadores e, mesmo gastando mais em latência, levar a um menor *makespan*. Na maioria das técnicas descritas na literatura (e.g. [3, 23]), tem-se uma ordem fixa de envio de dados para os processadores. O processador mestre começa então a enviar $\alpha_{i,1}$ dados para cada processador P_i segundo a ordem já definida, $i = 1, \ldots, n$. Após o término do envio de dados para todos os processadores na primeira rodada, P_0 envia $\alpha_{i,2}$ dados para cada processador P_i na mesma ordem, e

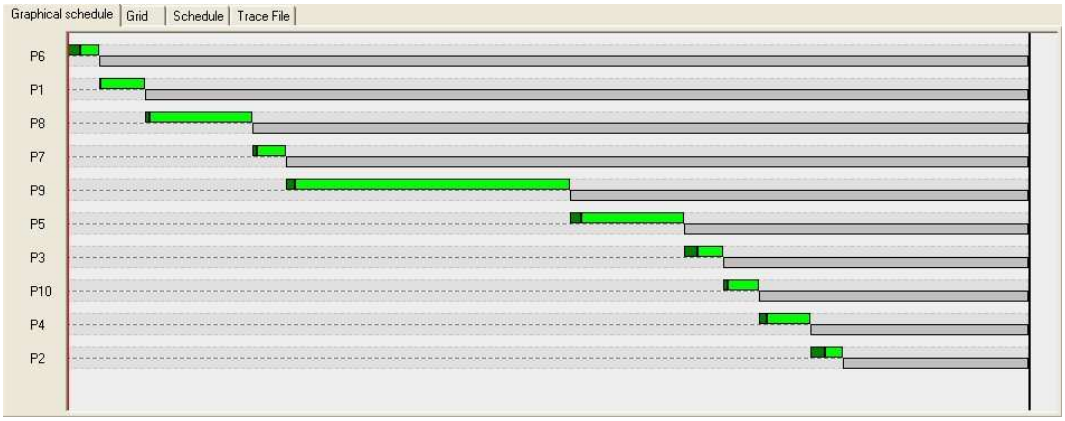

Figura 1.2: Escalonamento ótimo em apenas um período

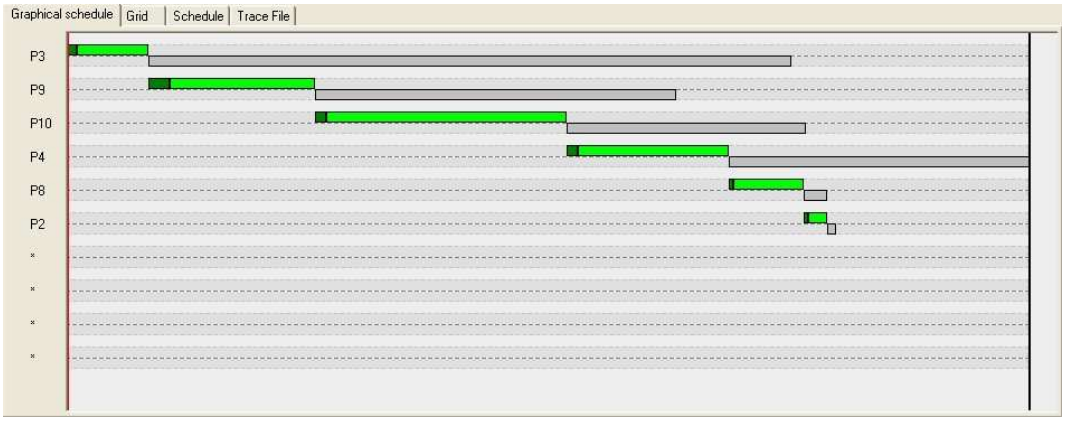

Figura 1.3: Escalonamento não ótimo em apenas um período

assim por diante. O gráfico de Gantt da Figura 1.4 ilustra um exemplo onde são utilizados três períodos de envio (delimitados por barras verticais). Salienta-se a existência de concorrência de comunicação e computação nos dois últimos períodos, onde todos os processadores começaram a receber dados (barras mais escuras) antes de terminar de processar os dados recebidos no período anterior (barras mais claras). Esta é uma característica muito importante a ser explorada por técnicas de múltiplos período.

1.3 Trabalhos correlatos

O modelo de processamento de tarefas divisíveis foi introduzido em [11] para analisar o custo de computação e comunicação em redes de sensores inteligentes. Depois disso o modelo foi generalizado e aplicado a várias topologias

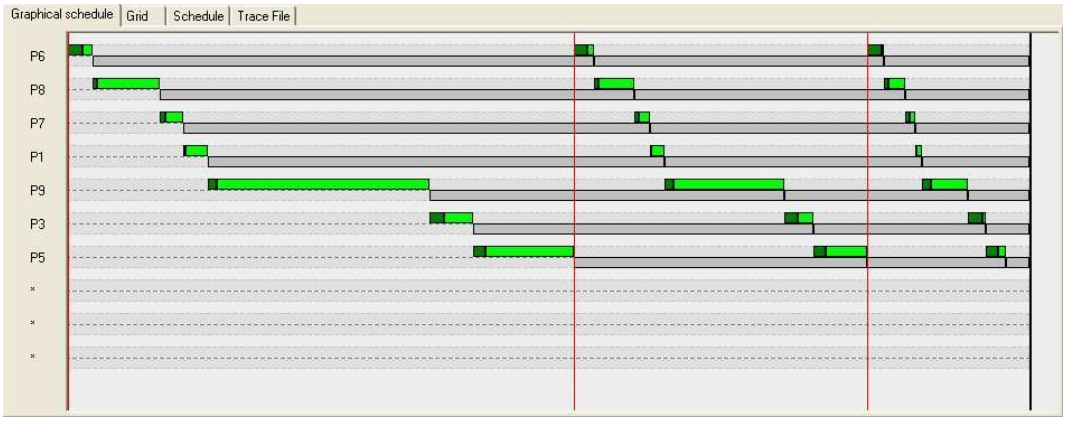

Figura 1.4: Escalonamento em múltiplos períodos

de interconexão, tais como barramentos [5], redes estrela [5] e hipercubos [7]. No entanto, a topologia de rede estrela é a mais comumente estudada, principalmente por se adequar mais às tecnologias atuais de interligação de *clusters* e redes locais.

No modelo apresentado em [5], as latências de comunicação não foram consideradas, caracterizando um modelo linear. Porém, considerar latências é muito importante para que se obtenha resultados mais próximos da realidade, o que faz com que a maioria dos trabalhos mais recentes as considerem (e.g. [4, 8, 21, 23]).

A modelagem do custo de comunicação foi generalizada por [8] com a inclusão de um tempo de latência fixo (equivalente ao elemento do modelo já apresentado g_i) chamado de tempo de inicialização (*startup time*), que também é utilizado em [4, 21]. Outros tipos de latência de comunicação e computação foram considerados em [23], porém neste trabalho optou-se por seguir a formulação já apresentada que é a mesma encontrada em [4, 8, 21]. A complexidade incluída na modelagem de [23] aumenta muito a dificuldade de descrição do problema, sem contribuir muito para sua aproximação à realidade. É importante notar que os parâmetros do sistema utilizados (g_i, w_i e G_i, $i = 1, \ldots, n$) são de fácil obtenção na prática e bons resultados foram obtidos através de experimentos [13, 21].

Dois tipos de cenários foram estudados em [3, 5], dependendo da possibilidade de escravos fazerem computações enquanto recebem dados do mestre ou não. O modelo que permite a sobreposição de comunicação e computação é muito utilizado na literatura por ser mais próximo das características atuais dos recursos computacionais e, portanto, foi a opção deste trabalho.

Não existe um método na literatura que alcance soluções ótimas de escalonamento de tarefas divisíveis (*makespan* mínimo) de forma eficiente. Porém existem várias heurísticas propostas e algoritmos exatos para alguns casos especiais.

1.3.1

Período único

Para a determinação de como escalonar as tarefas de forma a obter-se o menor *makespan* possível, existe a necessidade de se determinar em que ordem os processadores receberão os dados e a quantidade que cada um receberá.

Quando não existem latências de comunicação ($g_i = 0, i = 1, \ldots, n$), encontram-se em [5] métodos simples para se chegar à distribuição ótima de carga tanto para redes homogêneas como para heterogêneas (incluindo ordenação de envio, escolha de processadores e quantidade a ser enviada).

No caso de modelos mais elaborados com tempos de latência ($g_i \geq 0, i = 1, \ldots, n$), encontram-se em [4, 8] métodos exatos para algumas situações específicas. Uma primeira formulação não-linear inteira do problema com latências é apresentada em [14] e posteriormente aprimorada e adaptada em [21] para o caso onde existam restrições de memória nos processadores.

1.3.2

Múltiplos períodos

Além das escolhas que são feitas pelas técnicas que trabalham com apenas um período (ordem de envio, número de processadores a serem utilizados e quantidade de dados a serem enviados), quando trabalha-se com múltiplos períodos é necessário que se determine o número total de períodos de escalonamento e a duração de cada um. Em todos os trabalhos consultados que propõem técnicas de múltiplos períodos, por motivo de simplificação, é escolhida uma ordem única de envio de dados aos processadores que é seguida em todos os períodos.

Em [5] é proposta uma técnica para o escalonamento em múltiplos períodos em sistemas homogêneos e heterogêneos, que será descrita mais adiante neste trabalho. Esta técnica foi elaborada e analisada para sistemas

onde todas as latências são desprezíveis e, mesmo assim, sua otimalidade não foi comprovada.

No caso de modelos com tempos de latência, encontra-se em [4] métodos para escalonar tarefas em múltiplos períodos que obtém bons resultados. Estes métodos serão apresentados mais adiante neste trabalho.

1.4 Organização da dissertação

No Capítulo 2 são apresentados os resultados encontrados na literatura sobre escalonamento em período único, bem como novos resultados obtidos no decorrer deste trabalho. Como resultados encontrados na literatura, são apresentadas técnicas que alcançam resultados ótimos para sistemas sem latências e para alguns casos especiais de sistemas com latências. Como novos resultados, são apresentados modelos de programação linear inteira mista, uma técnica mais eficiente para um caso especial, uma heurística e dois métodos de busca local.

No Capítulo 3 são apresentados os resultados encontrados na literatura sobre escalonamento em múltiplos períodos, como técnicas para escalonamento em sistemas com e sem latências. Nesse mesmo capítulo são também apresentadas novas modelagens do problema como problemas de programação linear inteira mista e uma nova heurística de escalonamento.

No Capítulo 4 são apresentados os resultados computacionais obtidos pelas implementações das diversas técnicas apresentadas nos Capítulos 2 e 3 sobre um conjunto de casos teste.

Por fim, no Capítulo 5 são relatadas conclusões e apresentados temas para trabalhos futuros nesta linha de pesquisa.