Algoritmos para escalonamento de tarefas em plataformas heterogêneas usando o paradigma mestre-escravo

Fabio Henrique Nishihara e Alfredo Goldman
Instituto de Matemática e Estatística - Universidade de São Paulo
São Paulo/SP, Brasil
{fhn, gold}@ime.usp.br

Resumo

Neste trabalho abordamos o problema de escalonar tarefas independentes e de mesmo tamanho em uma plataforma heterogênea, onde os tempos de comunicação e de processamento são diferentes. Propomos algoritmos para escalonar tarefas usando o paradigma mestre-escravo em diversos tipos de rede que são casos especiais de uma grade heterogênea. Desta forma estendemos os trabalhos anteriores para plataformas heterogêneas. Através de simulações analisamos o desempenho dos algoritmos propostos em diferentes situações.

1. Introdução

Atualmente processamento paralelo em plataformas heterogêneas é um dos assuntos mais importantes em processamento de alto desempenho. Aplicações paralelas famosas tais como SETI@home [3] e Mersenne Prime Search [2] estão usando uma grande variedade de recursos computacionais para aumentar, ao máximo, os recursos disponíveis.

Neste trabalho abordamos o problema de escalonar tarefas de tamanhos iguais e independentes em plataformas heterogêneas, onde os canais (*links*) de comunicação e os processadores podem ser de qualquer tipo, e portanto têm velocidades diferentes. Esse problema consiste em determinar em qual processador cada tarefa será executada, de forma que o tempo de execução total destas tarefas seja minimizado.

No trabalho de Beaumont et al. [5], esse problema foi estudado para dois tipos especiais de rede, grafos *fork* e árvores. Neste trabalho é mostrado como cada nó pode conseguir a melhor alocação de tarefas aos recursos de modo a maximizar o número de tarefas processadas por unidade de tempo. Uma das redes que estudamos é a rede em árvore.

No trabalho de Dutot [9] foram abordados os casos de cadeia de processadores e de um caso especial de árvores, chamado de grafo aranha (*spider*). Nesse grafo, apenas o nó mestre pode ter vários filhos. O problema de escalonamento de tarefas foi abordado no contexto de uma grade, com processadores e conexões homogêneos. Neste trabalho, damos um passo além, estudando o problema de escalonamento em grades heterogêneas, que consistem em aglomerados de aglomerados. Com uma extensão do modelo original apresentamos novos resultados.

Estudamos também o caso de cadeia de nós, onde cada nó é um recurso computacional (um processador ou um aglomerado - *cluster*), capaz de processar e de se comunicar com seus vizinhos a taxas de velocidade diferentes. A motivação para usar uma rede em cadeia está relacionada aos resultados apresentados no trabalho de Li [11], onde o autor transforma uma grade homogênea com n-portas de comunicação em uma rede em cadeias com nós heterogêneos.

Este trabalho também está relacionado a tarefas divisíveis, que foram introduzidas inicialmente em [7]. Robertazzi et al. estudaram várias variações desse tópico. Em [4], inicialmente estudaram o problema da árvore homogênea. Então, em [13], abordaram o problema de barramento que é idêntico ao grafo *fork* com tempos de comunicação homogêneos e tempos de processamento heterogêneos. Em [6], trabalharam em um grafo em estrela com tempos de comunicação e processamento heterogêneos. A principal diferença é que estamos trabalhando com tarefas unitárias enquanto que uma tarefa divisível pode ser dividida em frações de qualquer tamanho.

A nossa contribuição consiste em apresentar dois modelos para redes heterogêneas, assim como a proposição de algoritmos, baseados no algoritmo apresentado em Dutot [9]. Apresentamos algoritmos para o escalonamento de tarefas independentes e de mesmo tamanho em sub-classes de grades heterogêneas (redes em árvores e redes em cadeia formadas por nós que podem ser apenas um processador ou um conjunto de processadores).

Na seção seguinte apresentamos as definições ne-

cessárias à apresentação dos algoritmos. Em seguida introduzimos o modelo e o algoritmo para redes de aglomerados na Seção 3. Na Seção 4 apresentamos o modelo e o algoritmo para redes em árvore. Experimentos numéricos são apresentados na Seção 5, logo antes da conclusão.

2. Definições

Consideramos uma rede em árvore e uma rede em cadeia de nós, onde cada nó é um aglomerado ou apenas um processador. Os tempos de comunicação e de processamento são diferentes.

Cada nó da rede tem apenas uma porta de entrada de comunicação e apenas uma porta de saída de comunicação. Supomos que para todo processador é possível ter sobreposição de comunicação e processamento.

Definição 1 *Uma rede G é chamada de rede seqüencial se satisfaz as seguintes condições:*

- *os processadores da rede podem ser divididos em grupos disjuntos $G_0, G_1, ..., G_m$;*
- *o grupo G_0 tem apenas um processador;*
- *existe apenas um caminho do processador do grupo G_0 para cada um dos outros processadores da rede;*
- *os processadores de um mesmo grupo estão a mesma distância do processador do grupo G_0, isto é, o número de canais de comunicações entre o processador do grupo G_0 e cada um dos processadores de um mesmo grupo são os mesmos.*

Dessa maneira, para uma tarefa chegar em um processador do grupo G_i, ela precisa passar por um processador de cada grupo G_j, onde $0 < j < i$.

Definição 2 *Um escalonamento de t tarefas em uma rede seqüencial com $n+1$ grupos é definido para toda tarefa i, onde:*

- *$P(i)$ é o processador onde a tarefa i será executada;*
- *$T(i)$ é o tempo de início de processamento da tarefa i no processador $P(i)$;*
- *$C(i)$ é um conjunto de tempos de início de comunicação da tarefa i de todos os canais de comunicação usados por ela. $C(i)$ é chamado de vetor de comunicação.*

O vetor de comunicação $C(i)$ é definido como:

$$C(i) = \{C_1^i; C_2^i; ...; C_k^i\},$$

onde $1 \leq k \leq n$ e C_j^i é o tempo que a tarefa i começa a ser transmitida de um processador do grupo G_{k-1} para um processador do grupo G_k.

Embora dois vetores de comunicação possam ter tamanhos diferentes, eles podem sempre ser comparados da seguinte maneira:

Definição 3 *Sejam $A = (a_1, ..., a_i)$ e $B = (b_1, ..., b_j)$, dois vetores de comunicação diferentes. A é inferior a B ($A < B$) se e somente se uma das seguintes condições é verificada:*

- *$\exists k \in [1; \min(i,j)]$ tal que $a_k \neq b_k$ e seja l o menor número tal que $a_l \neq b_l$, nós temos $a_l < b_l$;*
- *$i > j$ e $\forall k \in [1; j]$, $a_k = b_k$.*

Para verificar se um escalonamento é possível, é preciso verificar se o modelo da rede que é considerado está sendo satisfeito.

Definição 4 *Um escalonamento é possível se e somente se satisfaz as seguintes propriedades:*

- *Uma tarefa deve ser completamente recebida antes de começar a ser retransmitida para outro processador;*
- *Uma tarefa deve ser completamente recebida antes de começar a ser executada por um processador;*
- *Um processador não pode executar duas tarefas ao mesmo tempo;*
- *Um processador não pode receber duas tarefas ao mesmo tempo;*
- *Um processador não pode enviar duas tarefas ao mesmo tempo.*

O *makespan* é definido como o menor tempo onde todas as tarefas estarão prontas.

Definição 5 *O makespan T_{max} é definido como:*

$$T_{max} = \max_{i \in [1;t]} (T(i) + w_{P(i)}).$$

A seguir, vamos definir uma função simples que ajudará nos nossos algoritmos.

Definição 6 *Seja p um processador no grupo G_i. A função:*

$$G(p) = i,$$

define qual é o número do grupo a qual o processador p pertence.

3. Algoritmo para redes em cadeia de aglomerados

Consideramos uma rede heterogênea em cadeia de nós onde cada nó pode ser um único processador ou um conjunto de processadores (aglomerado), como mostrado na Figura 1.

Assumimos que os tempos de comunicação dentro de cada aglomerado são muito menores em relação aos tempos de comunicação entre os nós. Com isso, desconsideramos

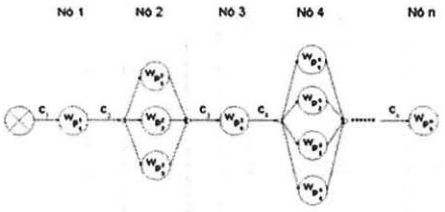

Figura 1. Rede em cadeia de aglomerados, onde o primeiro nó é o nó mestre.

os tempos de comunicação dentro de cada aglomerado. Assim, podemos transformar qualquer aglomerado com qualquer topologia em um aglomerado onde os processadores estão em paralelo. Isso é melhor representado pela Figura 2. Cada nuvem representa um aglomerado, não importando a sua topologia. Vale a pena ressaltar que isso só é possível, pois os tempos de comunicação dentro dos aglomerados não são considerados.

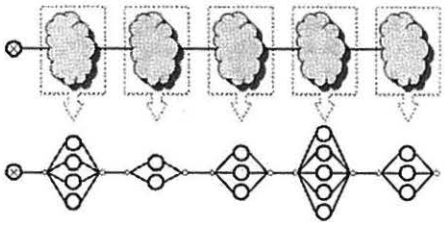

Figura 2. Transformação de um aglomerado qualquer para a nossa topologia.

A rede é formada por n nós, numerados de 1 a n. Os nós são representados como N_i, onde $1 \leq i \leq n$, sendo que o primeiro nó é o nó mais próximo do nó mestre. Cada nó N_i tem s_i processadores, numerados de 1 a s_i. Um processador é representado como p_j^i, onde i é o número do nó a qual o processador pertence e j é o número do processador dentro do nó N_i, com $1 \leq i \leq n$ e $1 \leq j \leq s_i$.

Cada processador p_j^i precisa de $w_{p_j^i}$ unidades de tempo para processar uma tarefa (então quanto menor $w_{p_j^i}$ for, mais rápido o processador p_j^i é). Denotamos por c_i as unidades de tempo necessárias para transmitir uma tarefa através do canal de comunicação conectando o nó N_{i-1} e o nó N_i.

3.1. Algoritmo

Nosso algoritmo escalona tarefas em uma rede em cadeia formada por processadores e aglomerados. Ele é um algoritmo guloso, as tarefas são escalonadas uma a uma e as já escalonadas nunca mais são reconsideradas. A construção da solução é feita de trás para frente.

O algoritmo recebe como entrada o número t de tarefas, o número n de nós, os tempos de comunicação c_i, os tempos de processamento $w_{p_j^i}$ e a quantidade de processadores s_i de cada nó, com $1 \leq i \leq n$ e $1 \leq j \leq s_i$. A saída do algoritmo é um escalonamento ótimo para as t tarefas. O algoritmo devolve os processadores onde as tarefas serão executadas (P), os tempos de início de processamento delas (T) e os vetores de comunicação (C) com os tempos de todos os canais de comunicação usados por essas tarefas.

Quando todas as tarefas estão escalonadas, é feito um deslocamento de C_1^1 unidades de tempo nos tempos de início de processamento $T(i)$ e nos tempos de início de transmissão $C(i)$ das tarefas, para todo i, para colocar o tempo do início do escalonamento no tempo 0.

Como o algoritmo constrói a solução na ordem inversa, foi introduzida uma constante T_∞ que é definida como:

$$T_\infty = c_1 + (t-1) * \max(w_{p_j^1}, c_1) + w_{p_j^1},$$

onde p_j^1 é o processador mais rápido do nó N_1 e $1 \leq j \leq s_1$. Esse tempo nada mais é do que o tempo de um escalonamento onde todas as tarefas são executadas no processador mais rápido do primeiro nó.

Foram necessárias mais duas variáveis auxiliares para o algoritmo, que guardam os tempos que os canais de comunicação e os processadores podem ser usados. Seja h_i, o tempo que o canal de comunicação pode ser usado para transmitir uma tarefa do processador p_{i-1} para o processador p_i, onde $1 \leq i \leq n$. Seja $o_{p_j^i}$ o tempo que o processador p_j^i do nó N_i pode ser usado para executar uma tarefa, onde $1 \leq i \leq n$. Elas são inicializadas com T_∞ em todos os seus valores.

O algoritmo inicia escalonando a tarefa t. Para isso são calculados n vetores de comunicação ${}^kC(t)$, com $1 \leq k \leq n$, correspondendo à execução da tarefa t em um processador de cada um dos n nós terminando no tempo T_∞. Os vetores ${}^kC(t)$ são definidos como:

$${}^kC(t) = \{{}^kC_1^t; ...; {}^kC_k^t\},$$

onde k é o número do nó do processador que executará a tarefa t e $1 \leq k \leq n$.

Para calcular os valores do vetor de comunicação ${}^kC(t)$, também precisamos escolher o processador do nó N_k que executará a tarefa. Como o algoritmo escalona as tarefas na ordem inversa, escolhemos o processador onde a tarefa poderá começar a ser executada o mais tarde possível. No caso da tarefa t, escolhemos o processador mais rápido (com o menor tempo de processamento w) do nó N_k, pois todos os processadores estão disponíveis no tempo T_∞. Assim os valores do vetor de comunicação ${}^kC(t)$ são:

$$^{k}C_i^t = T_\infty - w_{p_j^k} - \sum_{m=i}^{k} c_m,$$

onde p_j^k é o processador mais rápido do nó N_k e $1 \leq i \leq k$.

Calculados esses n vetores de comunicação, o maior vetor é escolhido de acordo com a condição especificada na definição 3. Assim, seja $^{k}C(t)$ o vetor de comunicação escolhido, então $C(t)$ é atualizado com esse vetor e a tarefa é colocada no processador $P(t) = p_k$ com tempo de início de processamento $T(t) = T_\infty - w_{P(t)}$. Além disso, as variáveis auxiliares h e o são atualizadas como:

$$o_{P(t)} = T(t),$$
$$h_i = C_i^t,$$

onde $1 \leq i \leq G(P(t))$.

Escalonada a tarefa $j+1$, a próxima a ser escalonada é a tarefa j. O processo é parecido com o escalonamento da tarefa t, onde são calculados os n vetores de comunicação $^{k}C(j)$, com $1 \leq k \leq n$, correspondendo a execução da tarefa j em um processador de cada um dos n nós e escolhemos o maior deles. Porém a escolha do processador do nó N_k que executará a tarefa e o cálculo dos valores do vetor $^{k}C(j)$ são diferentes. Continuamos escolhendo o processador onde a tarefa poderá começar a ser executada o mais tarde possível, mas não necessariamente será o processador mais rápido do nó, pois também depende do tempo que o processador poderá ser usado (a variável o guarda esse tempo para cada processador).

O processador escolhido do nó N_k para processar a tarefa será aquela onde o valor $o_{p_i^k} - w_{p_i^k}$ for maior, com $1 \leq k \leq n$ e $1 \leq i \leq s_k$. Denotaremos por $q(k)$ esse processador.

Assim os valores do vetor $^{k}C(j)$ são:

$$^{k}C_k^j = \min(o_{q(k)} - w_{q(k)} - c_k, h_k - c_k),$$
$$^{k}C_i^j = \min(^{k}C_{i+1}^j - c_i, h_i - c_i),$$

onde $1 \leq i \leq k-1$.

Seja $^{k}C(j)$ o vetor de comunicação escolhido, então atualizamos as funções $C(j)$, $P(j)$ e $T(j)$ como:

$$C(j) = {}^{k}C(j),$$
$$P(j) = q(k),$$
$$T(j) = o_{P(j)} - w_{P(j)}.$$

Além disso, atualizamos as variáveis auxiliares h e o como:

$$o_{P(j)} = T(j),$$
$$h_i = C_i^j,$$

```
// Calcula T∞. Processador p_j^1
// é o mais rápido do nó N_1.
T∞ = c_1 + (t − 1) * max(c_1, w_{p_j^1}) + w_{p_j^1}

// Inicializa as variáveis
// auxiliares h e o com T∞
Para i = 1 até n faça
  h_i = T∞
  Para j = 1 até s_i faça
    o_{p_j^i} = T∞

// Inicializa C(i)
Para i = 1 até t faça
  C(i) = 0, ..., 0

Para i = t até 1 faça
  // Calcula os vetores de
  // comunicação
  Para k = n até 1 faça
    // Escolha do processador do
    // nó N_k que executará a tarefa
    m_k = −1
    Para j = 1 até s_k faça
      Se m_k ≤ o_{p_j^k} − w_{p_j^k} então
        m_k = o_{p_j^k} − w_{p_j^k}
        q(k) = p_j^k

    ^kC_k^i = min(m_k − c_k, h_k − c_k)
    Para j = k − 1 até 1 faça
      ^kC_j^i = min(^kC_{j+1}^i − c_j, h_j − c_j)
    Se C(i) < ^kC(i) então
      C(i) = ^kC(i)

  k = tamanho(C(i))
  P(i) = q(k)
  T(i) = o_{P(i)} − w_{P(i)}

  // Atualiza h e o
  o_{P(i)} = T(i)
  Para j = 1 até k faça
    h_j = C_j^i

// Faça um deslocamento no tempo
Para i = t até 1 faça
  T(i) = T(i) − C_1^1
  Para k = G(P(i)) até 1 faça
    C_k^i = C_k^i − C_1^1
```

Figura 3. Algoritmo para redes em cadeia de aglomerados com qualquer topologia.

onde $1 \leq i \leq G(P(j))$.

Depois de escalonadas todas as tarefas, aplicamos o deslocamento de C_1^1 unidades de tempo em $T(i)$ e em $C(i)$. O makespan T_{max} do escalonamento é:

$$T_{max} = \max_{i \in [1;t]} (T(i) + w_{P(i)}).$$

O algoritmo em pseudo-código é mostrado na Figura 3. A complexidade do algoritmo é $O(t(p+n^2))$ e da comparação do vetor de comunicação é $O(n)$, onde t é

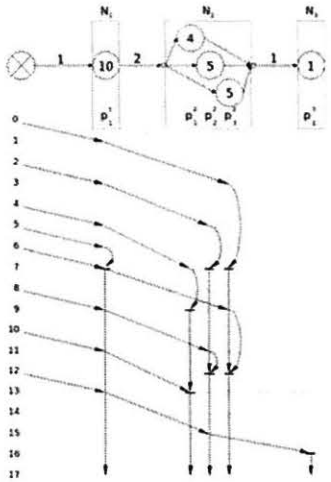

Figura 4. Exemplo de um escalonamento usando o algoritmo da Figura 3.

o número de tarefas a serem escalonadas, p é o número de processadores na rede e n é o número de nós. A Figura 4 mostra um exemplo de um escalonamento.

4. Algoritmo para redes em árvore

Consideramos uma rede heterogênea em árvore, onde o processador raiz da árvore é o processador mestre. A Figura 5 exibe um exemplo dessa rede.

A rede é formada por n processadores, numerados de 1 a n, que são representados como p_i, onde $1 \leq i \leq n$. O processador raiz é representado como p_0.

Em uma rede em árvore, os processadores que têm profundidade i na árvore, estarão no grupo G_i. p_0 está no grupo G_0.

Sem perda de generalidade, assumimos que os números dos processadores do grupo G_i são menores que do grupo G_{i+1}. Isto é, considere um processador p_i no grupo G_a e um outro processador p_j no grupo G_b. Se $a > b$, então $i > j$.

Representamos por w_i as unidades de tempo necessárias para executar um tarefa no processador p_i. Representamos por c_i as unidades de tempo necessárias para transmitir uma tarefa utilizando o canal de comunicação chegando no processador p_i.

4.1. Algoritmo

Esse algoritmo também é um algoritmo guloso e faz o escalonamento de tarefas considerando uma rede em árvore. O algoritmo recebe como entrada o número t de tarefas, o número n de nós, os tempos de comunicação c_i, os tempos

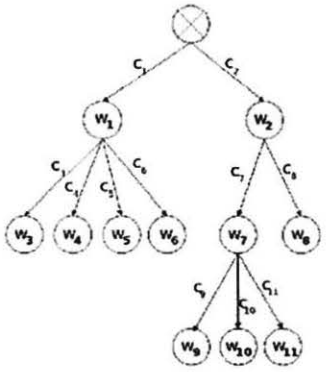

Figura 5. Rede em árvore.

de processamento $w_{p_j^i}$) e os números dos processadores pais pai_i de cada processador, onde $1 \leq i \leq n$ (pai_i é o número do processador pai de p_i). A saída do algoritmo é a mesma do algoritmo anterior.

Introduzimos uma constante T_∞ que é definida como:

$$T_\infty = c_1 + (t-1) * \max(c_1, w_1) + w_1,$$

onde p_1 é um processador do primeiro grupo.

Seja h_i, o tempo que o processador p_i pode ser usado para transmitir uma tarefa para um dos seus filhos, onde $0 \leq i \leq n$. Seja o_i, o tempo que o processador p_i pode ser usado para executar uma tarefa, onde $1 \leq i \leq n$. Note que a definição de h_i é um pouco diferente da seção anterior. Essas variáveis h e o são inicializadas com T_∞.

Começamos o algoritmo escalonando a tarefa t. Para todo processador p_k, com $1 \leq k \leq n$, calculamos o vetor de comunicação ${}^kC(t)$ correspondendo a execução da tarefa t no processador p_k terminando no tempo T_∞. Os vetores ${}^kC(t)$ são definidos como:

$${}^kC(t) = \{{}^kC_1^t; ...; {}^kC_{G(k)}^t\},$$

onde

$${}^kC_i^t = T_\infty - w_k - \sum_{m=i}^{G(k)} c_m,$$

onde $1 \leq i \leq G(k)$.

Depois de calculados os n vetores de comunicação, escolhemos o maior vetor de acordo com a Definição 3.

Seja ${}^kC(t)$ o vetor de comunicação escolhido, então atualizamos $C(t)$ com esse vetor. O processador que executará a tarefa é $P(i) = k$ iniciando a execução em $T(t) = T_\infty - w_{P(t)}$.

Além disso, precisamos atualizar as variáveis auxiliares h e o como:

$$o_{P(t)} = T(t),$$

$$h_{pai_i} = C_i^t,$$

onde $1 \leq i \leq pai_{P(t)}$.

Depois de escalonada a tarefa $j+1$, escalonamos a tarefa j. Os passos são parecidos com o escalonamento da tarefa t descrito acima. Escolhemos o maior vetor dos n vetores de comunicação ${}^kC(j)$ calculados, com $1 \leq k \leq n$, correspondendo à execução da tarefa j nos n processadores. Porém o cálculo dos valores do vetor ${}^kC(j)$ é um pouco diferente do que o mostrado acima.

$${}^kC^j_{G(k)} = \min(o_k - w_k - c_k, h_{pai_k} - c_k),$$

$${}^kC^j_i = \min({}^kC^j_{i+1} - c_i, h_{pai_i} - c_i),$$

onde $1 \leq i \leq pai_k - 1$.

Seja ${}^kC(j)$ o vetor de comunicação escolhido, então atualizamos $C(j)$ com esse vetor e $P(j) = k$ com $T(j) = o_{P(j)} - w_{P(j)}$

Além disso, as variáveis auxiliares h e o são atualizadas como:

$$o_{P(j)} = T(j),$$

$$h_{pai_i} = C_i^j,$$

onde $1 \leq i \leq pai_{P(j)}$.

No final, é feito o deslocamento em $T(i)$ e em $C(i)$ de C_1^1 unidades de tempo. O makespan T_{max} do escalonamento é:

$$T_{max} = \max_{i \in [1;t]} (T(i) + w_{P(i)}).$$

A Figura 6 mostra o algoritmo em pseudo-código. A complexidade do algoritmo é $O(tnm)$ e da comparação do vetor de comunicação é $O(m)$, onde t é o número de tarefas a serem escalonadas, n é o número de processadores e m é a altura da árvore (número de grupos). Um exemplo de um escalonamento é mostrado na Figura 7.

5. Experimentos numéricos

Para comparar os dois algoritmos, propomos um algoritmo para escalonamento de tarefas independentes e de mesmo tamanho em uma rede heterogênea onde existe um único caminho entre o processador inicial (processador mestre) e os outros.

O algoritmo é baseado no SPT [10] (*Shortest Processing Time*). Este método prioriza as tarefas com os menores tempos de processamento. No nosso modelo, como todas as tarefas têm o mesmo tamanho, o algoritmo escolhe o processador que termina de executar a tarefa o quanto antes. Esse algoritmo, assim como os outros, é um algoritmo guloso.

```
// Calcula T∞
T∞ = c1 + (t − 1) * max(c1, w1) + w1

// Inicializa as variáveis
// auxiliares h e o com T∞
Para i = 0 até n faça
  h_i = T∞
  o_i = T∞

// Inicializa C(i)
Para i = 1 até t faça
  C(i) = 0, ..., 0

Para i = t até 1 faça
  // Calcula os vetores de
  // comunicação
  Para k = n até 1 faça
    p = pai_k
    kC^i_G(k) = min(o_k − w_k − c_k, h_p − c_k)
    Para j = G(k) − 1 até 1 faça
      p = pai_p
      kC^i_j = min(kC^i_{j+1} − c_j, h_p − c_j)
    Se C(i) < kC(i) então
      C(i) = kC(i)
      P(i) = k

  T(i) = o_P(i) − w_P(i)

  // Atualiza h e o
  o_P(i) = T(i)
  p = P(i)
  Para k = G(p) até 1 faça
    p = pai_p
    h_p = C^i_k

// Faça um deslocamento no tempo
Para i = t até 1 faça
  T(i) = T(i) − C^1_1
  Para k = P(i) até 1 faça
    C^i_k = C^i_k − C^1_1
```

Figura 6. Algoritmo para rede em árvores.

Para cada tarefa, calculamos quando ela terminará de ser processada em cada processador da rede. O cálculo leva em consideração o tempo de comunicação para a tarefa chegar no processador a partir do processador inicial (mestre) e o tempo de processamento da tarefa. O processador que terminar primeiro o processamento da tarefa é o escolhido.

Como pode ser visto, esse algoritmo é simples e pode ser usado para uma grande variedade de redes. Se uma rede $G = (N, E)$ possui mais de um caminho para um dos seus processadores, podemos utilizar uma sub-rede $G' = (N, E')$ de G equivalente, eliminando esse caminhos duplicados $(E - E')$.

Comparando o algoritmo de Dutot [9] com o algoritmo baseado no SPT em redes em cadeia de processadores, podemos verificar que o algoritmo baseado no SPT obteve soluções próximas da ótima. A diferença entre os resultados dos dois algoritmos é pequena. Quando os tempos de

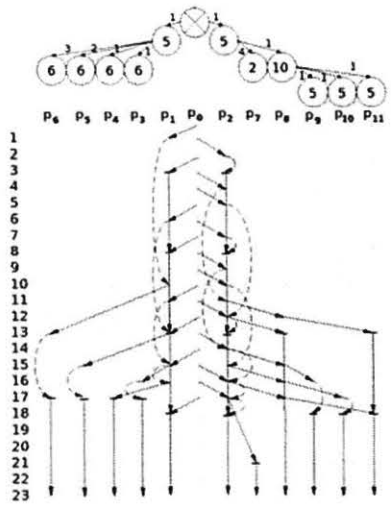

Figura 7. Exemplo de um escalonamento usando o algoritmo da Figura 6.

comunicação são maiores que os tempos de processamento, os resultados dos dois algoritmos são iguais.

Devido a esses resultados semelhantes comparamos nossos dois algoritmos com o baseado no SPT. Para o algoritmo para redes em árvore, comparamos também com o algoritmo apresentado no trabalho de Beaumont et al. [5].

Nas simulações, os tempos de execução w dos processadores e os tempos de transmissão c dos canais de comunicação foram gerados aleatoriamente dentro de um intervalo. Simulações mais completas podem ser encontradas em [12].

5.1. Algoritmo para redes em cadeia de aglomerados

Foram feitas simulações em redes homogêneas e heterogêneas, variando os tempos de comunicação e de processamento, e o número de aglomerados e de processadores em cada aglomerado. Executamos os algoritmos cem vezes para cada cenário para escalonar mil tarefas.

A Figura 8 mostra os resultados obtidos, considerando $80 \leq w \leq 120$ e $6 \leq c \leq 14$, variando o número de aglomerados (esquerda) e de processadores em cada aglomerado (direita). A figura mostra os mínimos, as médias e os máximos percentuais que o algoritmo foi melhor que o baseado no SPT. As linhas cheias são os resultados em redes heterogêneas e as linhas pontilhadas, em redes homogêneas.

Em todas as simulações feitas, o nosso algoritmo não apresentou soluções piores em todos os casos. Quando os tempos de comunicação são maiores que os tempos de processamento, os tempos de escalonamentos das soluções dos dois algoritmos são os mesmos.

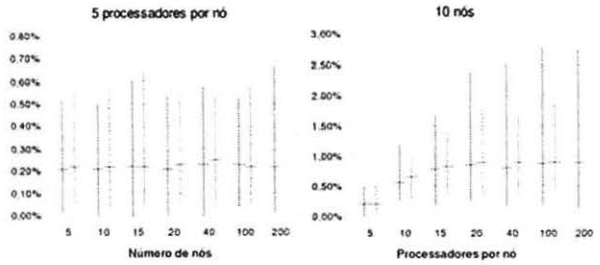

Figura 8. Resultados comparando com SPT.

5.2. Algoritmo para redes em árvore

O algoritmo de Beaumont et al. [5] calcula qual é a melhor alocação das tarefas nos processadores. Os padrões das comunicações e dos processamentos se repetem a cada período de T unidades de tempo. No cálculo de T, é considerado o mínimo múltiplo comum dos tempos de processamento de quase todos os processadores da rede e também dos tempos de comunicação de alguns canais de comunicação. Dependendo dos valores desses tempos e da quantidade de processadores na rede, o valor de T pode ser muito grande.

Em nossos experimentos executamos 10 períodos mais o período de inicialização. Não colocamos mais períodos, pois T poderia ser muito grande e, conseqüentemente, o número de tarefas também. Não simulamos o nosso algoritmo para processadores com muitos filhos e nem com redes com muitos grupos (grande altura), pois isso também levaria a um grande valor de T.

No final do algoritmo descobrimos o número t de tarefas que podem ser escalonadas nesses períodos. Depois de calcular quantas tarefas podem ser escalonadas nesse algoritmo, executamos o algoritmo baseado no SPT e o algoritmo proposto para escalonar essas t tarefas na mesma rede.

Executamos os três algoritmos cem vezes para cada cenário em uma árvore completa. As simulações foram feitas em ambientes homogêneos e heterogêneos, variando os tempos de comunicação e de processamento, a altura da árvore e o número de filhos de cada processador.

A Figura 9 mostra os resultados obtidos em uma rede onde $5 \leq w \leq 15$ e $1 \leq c \leq 5$, comparando com o algoritmo de Beaumont et al. [5] (esquerda) e com o algoritmo baseado no SPT (direita). A figura mostra, em porcentagem, o mínimo, a média e o máximo que o algoritmo foi melhor que os dois algoritmos. As linhas cheias são os resultados em redes heterogêneas e as linhas pontilhadas, em redes homogêneas.

A partir dos resultados podemos ver que nosso algoritmo apresentou melhores soluções na maioria dos casos. Quando os tempos de comunicação são maiores que os tempos de processamento e a rede é homogênea ou em cadeia

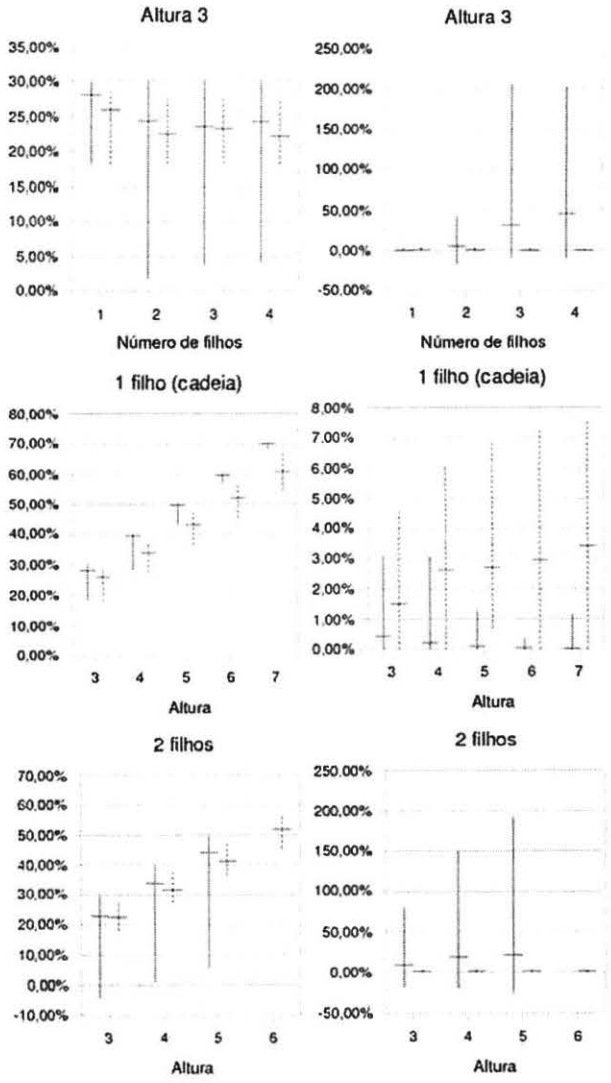

Figura 9. Resultados comparando com os algoritmos de [5] (esquerda) e SPT (direita).

(cada processador tem apenas um filho), os tempos de escalonamento das soluções do nosso algoritmo e do algoritmo baseado no SPT são os mesmos. A diferença entre os resultados em uma rede com 1 e 2 filhos comparando com o algoritmo baseado no SPT, provavelmente ocorre pois o SPT seja melhor para redes em cadeia.

6. Conclusão

Neste trabalho abordamos o problema de escalonar tarefas independentes e de mesmo tamanho em redes heterogêneas, com tempos de comunicação e de processamento diferentes. Para isto propomos dois novos modelos.

Seguindo a mesma linha dos trabalhos anteriores, neste trabalho propomos simplificações de redes heterogêneas de forma a permitir uma modelagem aproximativa, e propomos heurísticas para escalonar tarefas nas mesmas.

Consideramos redes em cadeia de aglomerados com qualquer topologia (onde os tempos de comunicação dentro dos aglomerados são muito menores em relação aos tempos de comunicação entre os aglomerados) e rede em árvores, melhorando os resultados apresentados em [4].

As heurísticas propostas apresentaram bons resultados quando comparadas com a heurística SPT, que apresenta resultados próximos aos ótimos [9] em cadeias de processadores.

Referências

[1] Entropia. URL: http://www.entropia.com.

[2] Mersenne prime search. URL: http://www.mersenne.org.

[3] Seti at home. URL: http://setiathome.ssl.berkeley.edu.

[4] S. Bataineh, T. Hsiung, and T. Robertazzi. Closed form solutions for bus and tree networks of processors load sharing a divisible job. *IEEE Transactions on Computers*, 43(10):1184–1196, 1994.

[5] O. Beaumont, L. Carter, A. Legrand, and Y. Robert. Bandwidth-centric allocation of indepedent tasks on heterogeneous plataform. *Internacional Parallel and Distributed Processing Symposium*, 2002.

[6] S. Charcranoon, T. Robertazzi, and S. Luryi. Optimizing computing costs using divisible load analysis. *IEEE Transactions on Computers*, 49(9):987–991, 2000.

[7] Y. Cheng and T. Robertazzi. Distributed computation for a tree network with communication delays. *IEEE Transactions on Aerospace and Eletronic Systems*, 24(6):700–712, 1988.

[8] J. Cowie, B. Dodson, R. M. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery, and J. Zayer. A world wide number field sieve factoring record: On to 512 bits. *Advances in Cryptology - ASIACRYPT'96*, 1163:382–394, 1996.

[9] P.-F. Dutot. Master-slave tasking on heterogeneous processors. *International Parallel and Distributed Processing Symposium*, 2003.

[10] J. Y.-T. Leung. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004.

[11] K. Li. Scheduling divisible task on heterogeneous linear arrays with applications to layered networks. *Workshop on Parallel and Distributed Scientific and Engineering Computing with Application*, 2002.

[12] F. H. Nishihara. Algoritmos para escalonamento de tarefas em plataformas heterogêneas usando o paradigma mestre-escravo. Master's thesis, Universidade de São Paulo, 2008.

[13] J. Sohn, T. Robertazzi, and S. Luryi. Optimizing computing costs using divisible load analysis. *IEEE Transactions on parallel and distributed system*, 9(3):225–234, 1998.