2

Escalonamento usando apenas um período

O escalonamento usando apenas um período (também encontrado na literatura como *round* ou *installment*) consiste em dividir a carga total em cargas menores e enviar cada fração uma única vez apenas para cada processador. Um exemplo de escalonamento em apenas um período é apresentado na Figura 2.1, onde o processador P_3 é o primeiro a receber dados, seguido pelos processadores P_9, P_{10}, P_4, P_8 e P_2.

Neste capítulo inicialmente serão apresentados os resultados encontrados na literatura para o problema quando latências de comunicação não são consideradas, assim como os poucos resultados disponíveis quando as mesmas são consideradas. A seguir, um novo algoritmo de complexidade $O(n)$ será apresentado para resolver um caso especial do problema, contrastando com o algoritmo de complexidade $O(n \log n)$ encontrado na literatura.

Na busca pela solução ótima, são então propostos um modelo de programação linear inteira mista e algumas desigualdades válidas. Isso permite encontrar um limite inferior [20] e resolver de forma exata instâncias de menor porte.

Além da solução exata, uma heurística construtiva também é proposta, o que permite encontrar soluções muito boas rapidamente. Por fim, são apresentadas duas técnicas de busca local para este problema, que, em conjunto com a heurística construtiva apresentada, obtém resultados muito bons.

2.1

Resultados anteriores para sistemas sem latências

Em [5] foi apresentado um estudo detalhado do problema desprezando-se as latências de comunicação. Neste trabalho, diversos resultados foram apresentados, dentre eles um algoritmo para alcançar a distribuição ótima

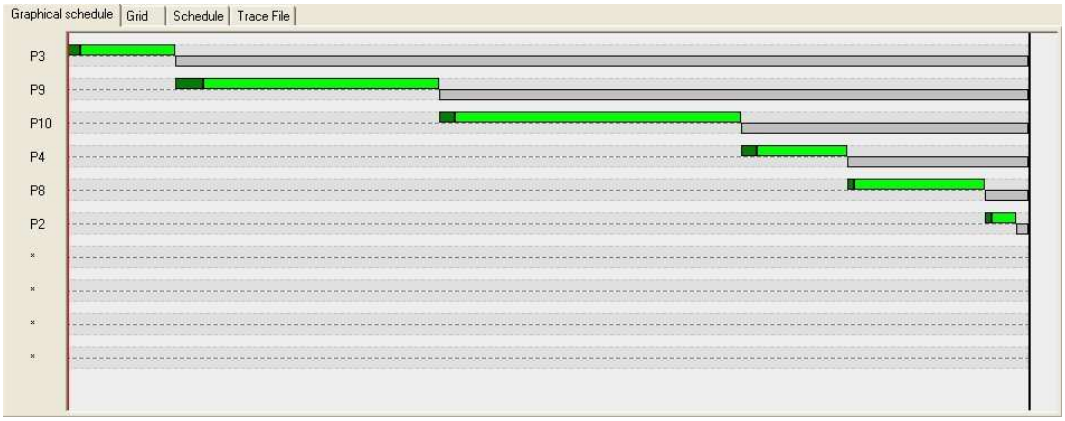

Figura 2.1: Exemplo de escalonamento em período único (sistema com latências)

de cargas, isto é, quais processadores devem participar da computação, qual a ordem ótima de envio e quantos dados cada processador deve receber (um exemplo de escalonamento pode ser visto na Figura 2.2). Esta técnica pode ser resumida nos seguintes passos:

Escolha da ordem de envio de cargas: A ordem na qual envia-se as cargas para os processadores pode influenciar o *makespan*. Para obter-se a distribuição ótima, processadores que estão associados aos enlaces mais rápidos devem receber carga antes dos demais. Ou seja, a ordem de envio de dados deve seguir a mesma ordem de velocidade dos enlaces, começando pelos de enlaces mais rápidos (menores valores de G_i) e terminando pelos de enlaces mais lentos (maiores valores de G_i). Como também demonstrado em [8], no caso dos enlaces serem idênticos e sem latência, a ordem de envio não altera o *makespan*. Desta forma, deve-se renomear os processadores de modo que $G_1 \leq G_2 \leq G_3 \leq \cdots \leq G_n$.

Escolha dos processadores que participarão da computação:
Existem alguns casos onde não é vantajoso que se utilize todos os processadores disponíveis para a computação. Para se alcançar a otimalidade (obter-se o menor *makespan* possível), deve ser utilizado o seguinte algoritmo para escolha de quais processadores devem participar do processamento:

1. Coloca-se todos os processadores em uma lista Σ, cada um representado por suas características w_i (inverso da taxa de processamento) e G_i (inverso da taxa de comunicação), $i = 1, \ldots, n$.

2. Elimina-se de Σ o processador P_k de maior índice que viola a seguinte condição para algum $r \in \{1, ...n-k\}$ e $1 \leq k \leq n-1$:

$$G_k < \frac{(G_{k+1} + w_{k+1}) \prod_{p=k+2}^{k+r} \frac{w_p + G_p}{w_{p-1}}}{1 + \sum_{i=k+2}^{k+r} \prod_{p=i}^{k+r} \frac{w_p + G_p}{w_{p-1}}}$$

Obtém-se este critério a partir de um conceito de equivalência entre processadores descrito em [5]. Diz-se desta forma que um processador só deverá receber dados caso seja mais vantajoso enviar dados para ele do que simplesmente enviar aos próximos processadores na ordem de envio.

3. Repita o passo anterior enquanto existirem enlaces que violem a condição.

Definição da carga a ser enviada para cada processador: Somente é possível atingir a otimalidade se todos os processadores que participam da computação, terminam de processar sua carga ao mesmo tempo. Desta afirmação, pode-se concluir que a carga a ser enviada para cada processador deve ser:

$$\begin{aligned} \alpha_k &= W \frac{\prod_{j=k+1}^{n} \frac{w_j + G_j}{w_{j-1}}}{(1 + \sum_{i=2}^{n} \prod_{j=i}^{n} \frac{w_j + G_j}{w_{j-1}})}, k = 1...n-1 \\ \alpha_n &= W \frac{1}{(1 + \sum_{i=2}^{n} \prod_{j=i}^{n} \frac{w_j + G_j}{w_{j-1}})}. \end{aligned}$$

Na Figura 2.2 apresenta-se o escalonamento resultante da aplicação desta técnica a um exemplo com dez processadores. Neste exemplo utilizou-se todos os processadores, porém o último recebeu quantidades ínfimas de carga (cerca de um milésimo da carga recebida pelo primeiro). Isto mostra que tal técnica, mesmo trazendo um valor ótimo de *makespan*, pode resultar numa utilização muito desequilibrada dos recursos.

2.2

Resultados anteriores para sistemas com latências

Ao contrário de quando não se considera latências de comunicação, neste problema ainda não foi encontrado um algoritmo polinomial que resulte no *makespan* ótimo. Existem porém resultados para alguns casos especiais onde a otimalidade pode ser conseguida com facilidade.

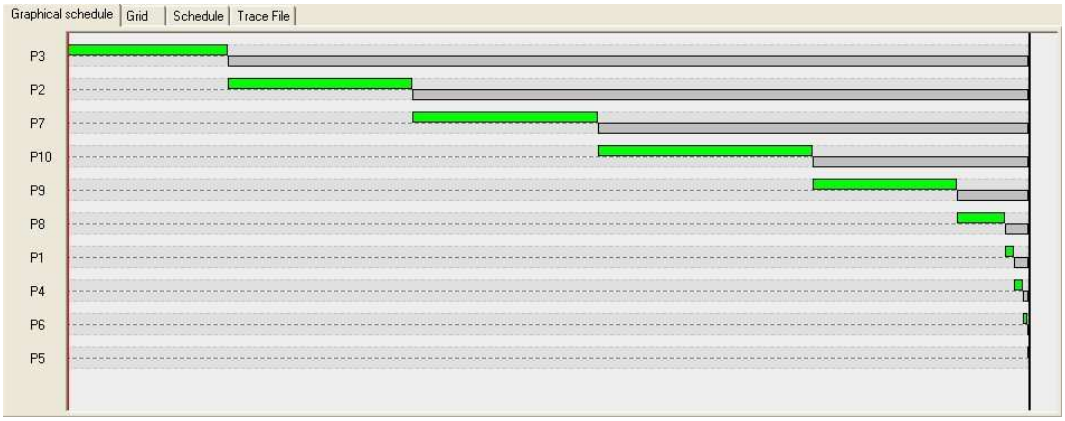

Figura 2.2: Escalonamento ótimo em apenas um período (sistema sem latências)

Em [8] foi apresentado um estudo da influência da inclusão de latências de comunicação em sistemas com diferentes topologias, sendo que casos especiais foram identificados para sistemas com rede estrela. Estes resultados, juntamente com aqueles encontrados em [4], permitem solucionar o problema de maneira ótima para sistemas:

Com taxas de transmissão idênticas [4]: A ordem de envio $j_1, j_2, \ldots, j_n$, deve ser tal que $g_{j_1} w_{j_1} \leq g_{j_2} w_{j_2} \leq \cdots \leq g_{j_n} w_{j_n}$.

Com enlaces idênticos [8]: A ordem de envio $k_1, k_2, \ldots, k_n$, deve ser tal que os processadores mais rápidos devem receber dados antes dos mais lentos, ou seja, $w_{k_1} \leq w_{k_2} \leq \cdots \leq w_{k_n}$.

Com ordem de envio pré-determinada [8]: Dado que os índices dos processadores foram renomeados de forma a espelhar a ordem de envio de dados, e que sabe-se o número ℓ de processadores que participarão da computação, pode-se calcular a quantidade de dados a ser enviada para cada processador observando-se as seguintes relações:

$$\alpha_k w_k = g_{k+1} + \alpha_{k+1}(w_{k+1} + G_{k+1}), \forall (1 \leq k \leq \ell - 1) \tag{2-1}$$

$$\sum_{k=1}^{\ell} \alpha_k = W. \tag{2-2}$$

A relação 2-2 implica que toda a carga W deve ser enviada e processada. As relações 2-1 implicam que o tempo total de comunicação e processamento da carga α_{k+1} referente ao processador P_{k+1} deve ser igual

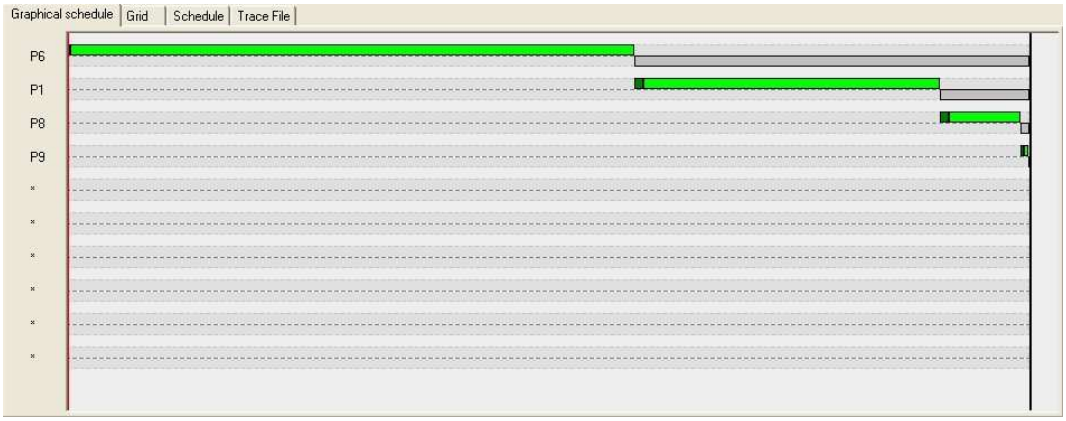

Figura 2.3: Escalonamento ótimo em apenas um período (sistema com latências)

ao tempo $\alpha_k w_k$ de processamento do anterior na ordem de envio (P_k). Garante-se dessa forma que todos terminarão o processamento ao mesmo tempo e não haverá ociosidade de comunicação do processador mestre (ou seja, só deixará de enviar dados quando todos já tiverem sido enviados). Este comportamento pode ser visto no escalonamento apresentado na Figura 2.3, onde o tempo total de computação do processador P_6 (barra cinza escuro) coincide com o tempo total de comunicação e computação dos dados do processador P_1, e assim por diante.

Nota-se que as relações 2-1 e 2-2 formam um sistema de ℓ equações e ℓ incógnitas, que pode ser resolvido em $O(\ell)$ segundo [8]. Propõe-se em [8] um algoritmo de complexidade $O(n \log n)$ capaz de definir o número ideal de processadores a serem utilizados (dada uma ordem fixa de envio). Nesta dissertação apresenta-se um novo algoritmo de complexidade $O(n)$ capaz de definir o número ideal de processadores e a quantidade ótima de dados a serem enviados a cada processador, dada a ordem de envio a ser seguida, como será visto na Seção 2.3.

Outro resultado interessante encontrado na literatura foi o Modelo 1, que é a primeira tentativa de se formular o problema geral com latências como um modelo de programação não-linear inteira [14].

Neste modelo tem-se como constantes as características do sistema (g_i, G_i e w_i, $i = 1, \ldots, n$) e o total W de dados a serem processados. Tem-se também n^2 variáveis inteiras e $n+1$ variáveis contínuas, definidas em 2-10, 2-11 e 2-12.

$$
\begin{aligned}
&Minimize \quad T && && (2\text{-}3)\\
&s.a.\\
&\textstyle\sum_{i=1}^{n} \alpha_i = W && && (2\text{-}4)\\
&\textstyle\sum_{i=1}^{n} x_{i,j} = 1 && \forall(1 \leq j \leq n) && (2\text{-}5)\\
&\textstyle\sum_{j=1}^{n} x_{i,j} = 1 && \forall(1 \leq i \leq n) && (2\text{-}6)\\
&\textstyle\sum_{k=1}^{j-1} \sum_{i=1}^{n} x_{i,k}(g_i + \alpha_i G_i) + \\
&\textstyle + \sum_{i=1}^{n} x_{i,j}(g_i + \alpha_i G_i + \alpha_i w_i) = T && \forall(1 \leq j \leq n) && (2\text{-}7)\\
&x_{i,j} \in \{0, 1\} && \forall(1 \leq i, j \leq n) && (2\text{-}8)\\
&\alpha_i \geq 0 && \forall(1 \leq i \leq n) && (2\text{-}9)
\end{aligned}
$$

Modelo 1: Modelo não-linear inteiro proposto em [14]

$$x_{i,j} = \begin{cases} 1, & \text{caso } P_i \text{ seja o } j\text{-ésimo processador a receber dados,} \\ 0, & \text{caso contrário;} \end{cases} \tag{2-10}$$

$$\alpha_i = \begin{cases} 0, & \text{caso } x_{i,j} = 0, \\ \geq 0, & \text{quantidade de dados transmitida a } P_i, \text{ caso } x_{i,j} = 1; \end{cases} \tag{2-11}$$

$$T = \text{tempo total ou } \textit{makespan}. \tag{2-12}$$

As restrições 2-4 implicam que o total de dados enviados aos escravos deve ser igual à carga W a ser tratada.

As restrições 2-5 implicam que deve existir um processador alocado para cada uma das n posições da ordem de envio de dados. Por exemplo, para $j = 2$, a restrição implica que um processador será o segundo na ordem de envio.

As restrições 2-6 implicam que todo processador deve ser alocado em apenas uma posição dentro da ordem de envio. Por exemplo, para $i = 3$ a restrição implica que o processador P_3 deverá receber dados uma única vez (ser alocado em apenas uma posição na ordem de envio).

As restrições 2-7 implicam que, para cada processador, todo o tempo gasto com comunicação até o início do envio de dados para ele, mais o tempo total de comunicação e processamento gasto com ele, deve ser igual ao *makespan*. A igualdade pode ser introduzida no modelo pois sabe-se [8] que na solução ótima todos os processadores terminarão no mesmo instante. Nota-se porém que nestas igualdades aparecem termos não-lineares como $x_{i,j}\alpha_i$.

Como salientado em [14], esta formulação pode não ter uma solução viável, uma vez que não seja possível fazer com que todos os processadores participem da computação e terminem o processamento no mesmo instante de tempo. Dessa forma, como constatado também por [4], esta formulação além de ser não-linear, também não é geral.

Também tratando deste mesmo problema de escalonamento, mas considerando uma restrição de memória dos processadores (a maior carga que um processador P_i pode receber deve ser igual ou inferior à constante B_i), foi proposto em [21] o Modelo 2.

$$
\begin{array}{lll}
Minimize \quad T & & (2\text{-}3) \\
s.a. & & \\
\sum_{i=1}^{n} \alpha_i = W & & (2\text{-}4) \\
\sum_{i=1}^{n} x_{i,j} \leq 1 & \forall (1 \leq j \leq n) & (2\text{-}13) \\
\sum_{j=1}^{n} x_{i,j} \leq 1 & \forall (1 \leq i \leq n) & (2\text{-}14) \\
\sum_{k=1}^{j-1} \sum_{i=1}^{n} x_{i,k}(g_i + \alpha_i G_i) & & \\
+ \sum_{i=1}^{n} x_{i,j}(g_i + \alpha_i G_i + \alpha_i w_i) \leq T & \forall (1 \leq j \leq n) & (2\text{-}15) \\
B_i \geq \alpha_i \geq 0 & \forall (1 \leq i \leq n) & (2\text{-}16) \\
x_{i,j} \in \{0, 1\} & \forall (1 \leq i, j \leq n) & (2\text{-}8)
\end{array}
$$

Modelo 2: Modelo não-linear inteiro proposto em [21]

Nesse modelo tem-se as mesmas variáveis e constantes do Modelo 1 e ainda um conjunto de n constantes B_i, $i = 1, \ldots, n$, que indicam os valores máximos que podem ser assumidos pelas variáveis α_i, restringidos pelas desigualdades 2-16.

As restrições 2-13 implicam que no máximo um processador pode ser alocado como sendo o j-ésimo a receber dados. Por exemplo, para $j = 2$, a restrição implica que poderá haver no máximo um processador como sendo o segundo na ordem de envio.

As restrições 2-14 implicam que um processador pode ser alocado em no máximo uma posição dentro da ordem de envio. Por exemplo, para $i = 3$ a restrição implica que o processador P_3 poderá receber dados no máximo uma vez.

As restrições 2-15, apesar de serem mais fracas que as restrições 2-7, produzem o mesmo efeito, uma vez que sabe-se [8] que na solução ótima todos os processadores terminarão no mesmo instante.

Devido ao fato das desigualdades 2-13 e 2-14 presentes nesta formulação serem mais fracas que as igualdades 2-5 e 2-6 na formulação apresentada por [14], existe a possibilidade de alguns processadores não serem utilizados. Mesmo este modelo sendo mais geral que o anterior, continua sendo um modelo não-linear. Até onde sabe-se, não existe uma formulação linear inteira mista para o problema, que foi um dos resultados obtidos no desenrolar deste trabalho.

2.3

Novo algoritmo rápido para resolução com ordem pré-determinada

Como apresentado anteriormente na Seção 2.2, sabida a ordem de envio de dados, existe na literatura [8] a descrição de um algoritmo de complexidade $O(n \log n)$ para determinar quantos processadores devem ser utilizados e qual o escalonamento ótimo (carga a ser enviada a cada um). Neste trabalho elaborou-se um algoritmo simples de complexidade $O(n)$ para sua solução.

Com o objetivo de simplificar as fórmulas apresentadas a seguir, define-se $f_j = (w_j + G_j)/(w_{j-1})$, nomenclatura utilizada em [5].

Como ponto de partida, serão utilizadas as equações 2-1 e 2-2 apresentadas na Seção 2.2. Com estas equações, é possível calcular a distribuição ótima de carga quando da utilização de ℓ processadores em uma determinada ordem pré-fixada.

$$\alpha_k w_k = g_{k+1} + \alpha_{k+1}(w_{k+1} + G_{k+1}) \quad ,\forall(1 \leq k \leq \ell - 1); \tag{2-1}$$

$$\sum_{k=1}^{\ell} \alpha_k = W. \tag{2-2}$$

Ao resolver-se a equação de recorrência 2-1, obtém-se

$$\alpha_k = \alpha_\ell \prod_{j=k+1}^{\ell} f_j + \sum_{j=k+1}^{\ell} (\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i), \forall(1 \leq k \leq \ell - 1). \tag{2-17}$$

Desta forma, substituindo-se α_k na igualdade 2-2 pela sua fórmula não recursiva 2-17, obtém-se

$$\alpha_\ell = \frac{(W - \sum_{k=1}^{\ell-1} \sum_{j=k+1}^{\ell} (\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i))}{(1 + \sum_{k=1}^{\ell-1} \prod_{j=k+1}^{\ell} f_j)}. \quad (2\text{-}18)$$

Pela equação 2-1, vê-se que $\alpha_i w_i$ é não-crescente para $i = 1, \ldots, \ell$, dado que os valores de G_i, w_i e g_i são sempre não-negativos. Como $w_i > 0, i = 1, \ldots, \ell$ e $\alpha_1 w_1 \geq \alpha_2 w_2 \geq \cdots \geq \alpha_\ell w_\ell$, tem-se que $\alpha_i \geq 0, i = 1, \ldots, \ell$ se e somente se $\alpha_\ell \geq 0$, o que caracteriza uma solução viável para o problema.

Pela equação 2-18, dado que f_i é sempre não-negativo, vê-se que α_ℓ terá valores viáveis (não negativos) sempre que $\sum_{k=1}^{\ell-1} \sum_{j=k+1}^{\ell} (\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i)$ for menor ou igual a W.

Define-se então a função $V(\ell) = \sum_{k=1}^{\ell-1} \sum_{j=k+1}^{\ell} \left(\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i\right)$ que faz parte do numerador da equação 2-18 e pode ser reescrita de forma recursiva:

$$\begin{aligned}
V(\ell+1) &= \sum_{k=1}^{\ell} \sum_{j=k+1}^{\ell+1} (\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i) \\
V(\ell+1) &= \sum_{k=1}^{\ell} \left[\frac{g_{\ell+1}}{w_\ell} \prod_{i=k+1}^{\ell} f_i + \sum_{j=k+1}^{\ell} \left(\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i \right) \right] \\
V(\ell+1) &= \sum_{k=1}^{\ell} \left(\frac{g_{\ell+1}}{w_\ell} \prod_{i=k+1}^{\ell} f_i \right) + \sum_{k=1}^{\ell} \sum_{j=k+1}^{\ell} \left(\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i \right) \\
V(\ell+1) &= \frac{g_{\ell+1}}{w_\ell} \sum_{k=1}^{\ell} \prod_{i=k+1}^{\ell} f_i + \sum_{k=1}^{\ell-1} \sum_{j=k+1}^{\ell} \left(\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i \right) \\
V(\ell+1) &= \frac{g_{\ell+1}}{w_\ell} \sum_{k=1}^{\ell} \prod_{i=k+1}^{\ell} f_i + V(\ell) \\
V(\ell) &= \frac{g_\ell}{w_{\ell-1}} \sum_{k=1}^{\ell-1} \prod_{i=k+1}^{\ell-1} f_i + V(\ell-1). \quad (2\text{-}19)
\end{aligned}$$

$V(\ell)$ é uma função crescente. Dado um número de processadores k, tal que $V(k) > W$ (solução inviável), se forem adicionados r processadores, $V(k+r)$ também será maior que W, ou seja, a solução continuará inviável.

Nota-se que a função $F(\ell) = \sum_{k=1}^{\ell-1} \prod_{i=k+1}^{\ell-1} f_i$, que faz parte da equação de recorrência 2-19, pode ser descrita de forma recursiva:

$$\begin{aligned} F(1) &= 0 \\ F(2) &= 1 \\ F(\ell) &= 1 + F(\ell-1)f_{\ell-1}. \end{aligned}$$

Finalmente, é possível descrever $V(\ell)$ em função de $V(\ell-1)$ e $F(\ell)$, o que permite calcular $V(\ell)$ em tempo constante a partir de $V(\ell-1)$ e $F(\ell)$:

$$\begin{aligned} V(1) &= 0 \\ V(\ell) &= \frac{g_\ell}{w_{\ell-1}}F(\ell) + V(\ell-1). \end{aligned}$$

Dada uma solução que utiliza um determinado número de processadores numa ordem definida, considera-se a colocação de mais um processador a participar da computação. Este receberá dados e a quantidade de carga enviada a todos os demais se reduzirá ou permanecerá a mesma. Logo, o *makespan* será reduzido ou permanecerá o mesmo. Conclui-se então que, para chegar ao *makespan* mínimo, deve-se incluir o máximo de processadores possível sem tornar a solução inviável.

Desta forma, pode-se partir de um escalonamento com apenas um processador participando da computação e incrementar o número ℓ de processadores utilizados, enquanto $\ell \leq n$ e $V(\ell) \leq W$. O maior número ℓ^* de processadores onde as duas desigualdades são respeitadas é então o número ótimo de processadores a serem utilizados.

Uma vez calculado em $O(n)$ o número ℓ^* ótimo de processadores e $V(\ell^*)$, pode-se calcular α_{ℓ^*} também com complexidade $O(n)$. Por fim, também é necessário apenas um algoritmo de complexidade $O(n)$ para se calcular o valor dos demais α_i utilizando-se a equação 2-2, permitindo a elaboração de um algoritmo completo com complexidade $O(n)$, uma vez que $O(n) + O(n) + O(n) = O(n)$ [12].

Uma possível descrição desta técnica pode ser vista no Algoritmo *AlgRap* (Algoritmo 1). Este algoritmo recebe o vetor *order*, que possui os índices dos processadores de forma que o envio seja feito segundo a ordem que aparecem no vetor. Caso por exemplo $order =< 2, 3, 1 >$, as cargas seriam enviadas na seguinte ordem: P_2, P_3 e P_1. Após executado, o algoritmo terá calculado

um vetor α com a quantidade de dados a ser enviada para cada processador, bem como o *makespan* total e o número ótimo l^* de processadores a serem utilizados.

No Algoritmo *AlgRap*, da linha 1 à 5 são calculados todos os valores de $F(\ell), \forall(1 \leq \ell \leq n)$. Da linha 6 à 9 são calculados todos os valores de $V(\ell), \forall(1 \leq \ell \leq n)$. Da linha 10 à 14 é encontrado o maior número de processadores ℓ^* que satisfaz às desigualdades $\ell^* \leq n$ e $V(\ell^*) \leq W$. Com estes valores, da linha 15 à 24 são calculados o numerador *num* e o denominador *den* da equação 2-18, de modo que $\alpha_{order[\ell^*]}$ é determinado (a quantidade de dados a serem enviados ao último processador $P_{order[\ell^*]}$). A partir do valor de $\alpha_{order[\ell^*]}$, utilizando-se as equações 2-1, da linha 25 à 27 é calculado quanto de carga será enviado a cada um dos demais processadores que participarão da computação ($\alpha_{order[k]}, \forall(1 \leq k \leq \ell^* - 1)$). Da linha 28 à 30 é dito que os processadores restantes não devem receber carga. Por fim, o *makespan* é calculado segundo a quantidade de dados enviada ao primeiro processador na ordem de envio, uma vez que este sempre participará da computação e que todos os demais terminarão o processamento ao mesmo tempo.

2.4

Novos modelos de programação linear inteira mista

Como visto na Seção 2.2, na literatura podem ser encontradas basicamente duas formulações não-lineares inteiras mistas, sendo que uma implica na utilização de todos os processadores. Neste trabalho foi elaborado um modelo de programação linear inteira mista que soluciona o problema na sua forma geral.

Inicialmente, nesta seção apresenta-se um método para obter-se um limite inferior para o *makespan* no problema geral, o qual é utilizado durante a resolução dos modelos, bem como na implementação de algumas desigualdades válidas para o problema. Adiante é apresentado um modelo simplificado onde todas as restrições do problemas são satisfeitas e o *makespan* ótimo é representado. A seguir são apresentadas modificações que melhoram a resolução do modelo por bibliotecas como CPLEX e GLPK. Por fim, é proposto um modelo com desigualdades válidas que melhoram o valor da relaxação linear, aumentando porém o número total de variáveis contínuas e de desigualdades, podendo piorar o tempo de resolução.

Algoritmo 1 AlgRap : Algoritmo rápido para resolução com ordem pré-determinada

Requer: *order*

Retorna: *makespan*, l^* e $\alpha_i, 1 \leq i \leq n$

1: $F[1] \leftarrow 0$
2: $F[2] \leftarrow 1$
3: **para** $i \leftarrow 3$ até n **faça**
4: $\quad F[i] \leftarrow 1 + F[i-1] * ((w_{order[i-1]} + G_{order[i-1]})/w_{order[i-2]})$
5: **fim para**
6: $V[1] \leftarrow 0$
7: **para** $i \leftarrow 2$ até n **faça**
8: $\quad V[i] \leftarrow (g_{order[i]}/w_{order[i-1]})F[i] + V[i-1]$
9: **fim para**
10: **para** $\ell \leftarrow 1$ até n **faça**
11: $\quad$ **se** $V[\ell] \leq W$ **então**
12: $\quad\quad \ell^* \leftarrow \ell$
13: $\quad$ **fim se**
14: **fim para**
15: $num \leftarrow W - V[\ell^*]$
16: $prod \leftarrow (w_{order[\ell^*]} + G_{order[\ell^*]})/w_{order[\ell^*-1]}$
17: $den \leftarrow 1$
18: **para** $k \leftarrow \ell^* - 1$ até 1 **faça**
19: $\quad den \leftarrow den + prod$
20: $\quad$ **se** $k \neq 1$ **então**
21: $\quad\quad prod \leftarrow prod * ((w_{order[k]} + G_{order[k]})/w_{order[k-1]})$
22: $\quad$ **fim se**
23: **fim para**
24: $\alpha_{order[\ell^*]} \leftarrow num/den$
25: **para** $k \leftarrow \ell^* - 1$ até 1 **faça**
26: $\quad \alpha_{order[k]} \leftarrow (g_{order[k+1]} + \alpha_{order[k+1]} \times (w_{order[k+1]} + G_{order[k+1]}))/w_{order[k]}$
27: **fim para**
28: **para** $k \leftarrow \ell^* + 1$ até n **faça**
29: $\quad \alpha_{order[k]} \leftarrow 0$
30: **fim para**
31: $makespan \leftarrow \alpha_{order[1]} \times (w_{order[1]} + G_{order[1]}) + g_{order[1]}$

2.4.1

Construção do limite inferior para o problema geral

Visando uma maior eficiência na resolução dos modelos apresentados, um dado importante a ser obtido é um limite inferior T_{min} para o *makespan* ótimo $M(\Gamma, W)$ gerado por um sistema Γ (que consiste no conjunto de todos os processadores e enlaces) ao receber uma carga total W. Este limite inferior é obtido a partir do *makespan* ótimo $M(\Gamma'', W)$ encontrado para um sistema simplificado Γ''.

A partir de um sistema Γ, cria-se um novo sistema Γ' com as mesmas características de comunicação e processamento que os presentes em Γ, porém com latências $g'_i = g_i - \underline{g}, \forall (1 \leq i \leq n)$, onde $\underline{g} = \min\{g_i \mid 1 \leq i \leq n\}$. Garante-se desta forma $M(\Gamma, W) \geq M(\Gamma', W) + \underline{g}$. A partir de Γ', constrói-se o sistema Γ'' onde as taxas de comunicação e processamento se mantêm, enquanto que as latências são desprezadas. Desta forma $M(\Gamma', W) \geq M(\Gamma'', W)$. Como apresentado na Seção 2.1, pode-se encontrar facilmente o *makespan* ótimo $M(\Gamma'', W)$ para Γ'', o que permite obter o limite inferior $T_{min} = M(\Gamma'', W) + \underline{g}$ para $M(\Gamma, W)$.

2.4.2

Modelo simplificado

No Modelo simplificado 3 tem-se como constantes as características do sistema (g_i, G_i e w_i, $i = 1, \ldots, n$) e o total W de dados a serem processados, além das seguintes variáveis:

$$x_{i,j} = \begin{cases} 1, & \text{caso } P_i \text{ seja o } j\text{-ésimo processador a receber dados,} \\ 0, & \text{caso contrário;} \end{cases}$$

$$\alpha_{i,j} = \begin{cases} 0, & \text{caso } x_{i,j} = 0, \\ \geq 0, & \text{quantidade de dados transmitida a } P_i \text{ quando } x_{i,j} = 1; \end{cases}$$

$T =$ tempo total ou *makespan*;

$t_j =$ momento em que o j-ésimo processador começará a receber dados.

Nota-se que o número de variáveis inteiras é o mesmo existente no Modelo 1, porém o número de variáveis contínuas aumentou de $n + 1$ para $n^2 + n + 1$

para permitir a linearização do modelo.

$$
\begin{array}{lll}
Minimize \quad T & & (2\text{-}3) \\
s.a. & & \\
\sum_{i=1}^{n} x_{i,j} \leq 1 & \forall (1 \leq j \leq n) & (2\text{-}13) \\
\sum_{j=1}^{n} x_{i,j} \leq 1 & \forall (1 \leq i \leq n) & (2\text{-}14) \\
\sum_{i=1}^{n} x_{i,j} \geq \sum_{i=1}^{n} x_{i,j+1} & \forall (1 \leq j < n) & (2\text{-}20) \\
\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{i,j} = W & & (2\text{-}21) \\
\alpha_{i,j} \leq W x_{i,j} & \forall (1 \leq i,j \leq n) & (2\text{-}22) \\
t_1 = 0 & & (2\text{-}23) \\
t_j \geq t_{j-1} + \sum_{i=1}^{n} (g_i x_{i,j-1} + G_i \alpha_{i,j-1}) & \forall (2 \leq j \leq n) & (2\text{-}24) \\
t_j + \sum_{i=1}^{n} (g_i x_{i,j} + (G_i + w_i)\alpha_{i,j}) \leq T & \forall (1 \leq j \leq n) & (2\text{-}25) \\
x_{i,j} \in \{0,1\} & \forall (1 \leq i,j \leq n) & (2\text{-}8) \\
\alpha_{i,j} \geq 0 & \forall (1 \leq i,j \leq n) & (2\text{-}26)
\end{array}
$$

Modelo 3: Modelo linear inteiro simplificado

As restrições 2-20 implicam que só pode ser alocado um processador como sendo o j-ésimo caso haja um processador alocado para a posição $j-1$ da ordem de envio. Por exemplo, para $j = 4$ tem-se que só poderá haver um quinto processador na ordem de envio caso haja um processador alocado como sendo o quarto a receber dados.

As restrições 2-21 implicam que o total de dados enviados para os processadores deve ser igual ao total W de dados a serem tratados.

As restrições 2-22 implicam que um processador P_i só poderá receber dados como sendo o j-ésimo, caso este processador esteja alocado como tal.

A restrição 2-23 indica que o primeiro processador na ordem de envio começará a receber dados no instante 0.

As restrições 2-24 indicam que o j-ésimo processador a receber dados só pode começar a receber carga após o anterior na ordem de envio ter terminado de receber seus dados.

As restrições 2-25 indicam que o *makespan* T deverá ser maior que o tempo t_j de início de recebimento do j-ésimo processador mais o tempo $\sum_{i=1}^{n} (g_i x_{i,j} + (G_i + w_i)\alpha_{i,j})$ que o mesmo levou para receber e processar os dados, $j = 1, \ldots, n$.

2.4.3

Modelo com limites melhorados

Visando melhorar a performance de resolução do modelo, foram introduzidos dois aperfeiçoamentos: a troca da desigualdade 2-25 pela igualdade 2-28 e a redução do limite superior variável de $\alpha_{i,j}$ em 2-22 por aquele em 2-27.

$$
\begin{array}{lll}
Minimize \quad T & & (2\text{-}3) \\
s.a. & & \\
\sum_{i=1}^{n} x_{i,j} \leq 1 & \forall (1 \leq j \leq n) & (2\text{-}13) \\
\sum_{j=1}^{n} x_{i,j} \leq 1 & \forall (1 \leq i \leq n) & (2\text{-}14) \\
\sum_{i=1}^{n} x_{i,j} \geq \sum_{i=1}^{n} x_{i,j+1} & \forall (1 \leq j < n) & (2\text{-}20) \\
\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{i,j} = W & & (2\text{-}21) \\
\alpha_{i,j} \leq W_{i,j} x_{i,j} & \forall (1 \leq i,j \leq n) & (2\text{-}27) \\
t_1 = 0 & & (2\text{-}23) \\
t_j \geq t_{j-1} + \sum_{i=1}^{n} (g_i x_{i,j-1} + G_i \alpha_{i,j-1}) & \forall (2 \leq j \leq n) & (2\text{-}24) \\
t_j + \sum_{i=1}^{n} (g_i x_{i,j} + (G_i + w_i)\alpha_{i,j}) = T & \forall (1 \leq j \leq n) & (2\text{-}28) \\
x_{i,j} \in \{0,1\} & \forall (1 \leq i,j \leq n) & (2\text{-}8) \\
\alpha_{i,j} \geq 0 & \forall (1 \leq i,j \leq n) & (2\text{-}26)
\end{array}
$$

Modelo 4: Modelo linear inteiro com limites melhorados

A igualdade pode ser introduzida no modelo, pois sabe-se [8] que na solução ótima todos os processadores terminarão no mesmo instante. Para a realização do segundo melhoramento é necessária a obtenção de um limite superior T_{max} dado por alguma heurística construtiva. Com este limite, o limite superior de $\alpha_{i,j}$ pode ser reduzido da constante W para constantes $W_{i,j}$.

Sabendo-se que o primeiro processador a receber dados não poderá exigir um tempo de comunicação e processamento maior que T_{max}, tem-se $g_i + (w_i + G_i)\alpha_{i,1} \leq T_{max}, i = 1, \ldots, n$. Dessa forma, para todo processador P_i pode-se obter um limite superior $W_{i,1} = (T_{max} - g_i)/(w_i + G_i)$ para $\alpha_{i,1}$, impondo-se então $\alpha_{i,1} \leq W_{i,1} x_{i,1}$. Sabendo-se o máximo de carga que cada processador poderá receber caso venha a ser escolhido como primeiro a receber dados, pode-se calcular o período máximo de tempo disponível para comunicação e processamento do segundo processador. Este período será igual ao maior tempo de processamento possível do primeiro processador, ou seja, $\max\{W_{i,1} w_i \mid 1 \leq i \leq n\}$. Ao seguir-se indutivamente esta técnica, o Algoritmo *CalcLimS* (Algoritmo 2) calcula os limites $W_{i,j}, \forall (1 \leq i,j \leq n)$.

Neste algoritmo, as variáveis L_j representam o período máximo de tempo que o j-ésimo processador poderá obter para receber e processar dados. Sabendo-se o tempo máximo que um processador P_i obterá caso seja o j-ésimo a receber dados, tem-se

$$W_{i,j} = \frac{L_j - g_i}{w_i + G_i} \quad \forall (1 \leq i, j \leq n). \tag{2-29}$$

No caso do primeiro processador, $L_1 = T_{max}$. Para os demais,

$$L_{j+1} = \max\{W_{i,j} w_i \mid 1 \leq i \leq n\}, \quad \forall (1 \leq j \leq n-1). \tag{2-30}$$

Falta no entanto provar que não é possível existir uma solução ótima onde algum processador P_i receba uma fração de carga $\alpha^*_{i,j} > W_{i,j}$, sendo o *makespan* total $T^* \leq T_{max} = L_1$.

Dado que um processador P_v é o h-ésimo a receber dados, é sabido que $g_v + (w_v + G_v)\alpha^*_{v,h} \leq L_h$ (o que se verifica trivialmente quando $h = 1$). Caso $\alpha^*_{v,h} > W_{v,h}$, então $g_v + (w_v + G_v)\alpha^*_{v,h} > g_v + (w_v + G_v)W_{v,h} = g_v + (w_v + G_v)(L_h - g_v)/(w_v + G_v) = L_h$, que é uma contradição. Prova-se desta forma que $\alpha^*_{v,h} \leq W_{v,h}$. Com este resultado e com a relação $w_v W_{v,h} \leq L_{h+1}$ que é verificada pela definição 2-30, tem-se que, se algum processador P_k receber dados pós P_v, $g_k + (w_k + G_k)\alpha^*_{k,h+1} \leq w_v \alpha^*_{v,h} \leq w_v W_{v,h} \leq L_{h+1}$, o que conclui o passo indutivo.

Algoritmo 2 CalcLimS: Cálculo dos limites $W_{i,j}$

Requer: T_{max}
Retorna: $W_{i,j}, 1 \leq i, j \leq n$

```
1: L_1 = T_max
2: para j = 1 até n faça
3:    para i = 1 até n faça
4:       W_{i,j} ← (L_j − g_i)/(w_i + G_i)
5:    fim para
6:    L_{j+1} ← max{W_{i,j} w_i | 1 ≤ i ≤ n}
7: fim para
```

2.4.4

Modelo aprimorado com desigualdades válidas

Visando melhorar o limite dado pela relaxação linear do modelo inteiro, foi desenvolvido o Modelo 5, onde adicionou-se três conjuntos de desigualdades

válidas. Um refere-se apenas às variáveis x, outro estipula limites inferiores para as variáveis α e o último refere-se ao problema como um todo.

$$
\begin{array}{lll}
Minimize & T & (2\text{-}3)\\
s.a. & & \\
\sum_{i=1}^{n} x_{i,j} \leq 1 & \forall(1 \leq j \leq n) & (2\text{-}13)\\
\sum_{j=1}^{n} x_{i,j} \leq 1 & \forall(1 \leq i \leq n) & (2\text{-}14)\\
\sum_{i=1}^{n} x_{i,j} \geq \sum_{i=1}^{n} x_{i,j+1} & \forall(1 \leq j < n) & (2\text{-}20)\\
\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{i,j} = W & & (2\text{-}21)\\
\alpha_{i,j} \leq W_{i,j} x_{i,j} & \forall(1 \leq i,j \leq n) & (2\text{-}27)\\
\alpha_{i,j} \geq W'_{i,j} x_{i,j} & \forall(1 \leq i,j \leq n) & (2\text{-}31)\\
t_1 = 0 & & (2\text{-}23)\\
t_j \geq t_{j-1} + \sum_{i=1}^{n} (g_i x_{i,j-1} + G_i \alpha_{i,j-1}) & \forall(2 \leq j \leq n) & (2\text{-}24)\\
t_j + \sum_{i=1}^{n} (g_i x_{i,j} + (G_i + w_i)\alpha_{i,j}) = T & \forall(1 \leq j \leq n) & (2\text{-}28)\\
\sum_{i=1}^{n} x_{i,j} \geq \sum_{v=j}^{n} x_{i,v} & \forall(1 \leq j,i \leq n) & (2\text{-}32)\\
F_{i,j} \geq t_j + \sum_{s=1}^{n} (g_s x_{s,j} + G_s \alpha_{s,j}) + w_i \alpha_{i,j} & \forall(1 \leq i,j \leq n) & (2\text{-}33)\\
F_{i,j} = F_{i,j-1} + (g_i x_{i,j} + (G_i + w_i)\alpha_{i,j}) & \forall(1 \leq i \leq n, 2 \leq j \leq n) & (2\text{-}34)\\
T = F_{i,n} & \forall(1 \leq i \leq n) & (2\text{-}35)\\
x_{i,j} \in \{0,1\} & \forall(1 \leq i,j \leq n) & (2\text{-}8)\\
\alpha_{i,j} \geq 0 & \forall(1 \leq i,j \leq n) & (2\text{-}26)
\end{array}
$$

Modelo 5: Modelo aprimorado com desigualdades válidas

Desigualdades válidas (2-32) para as variáveis x

Considera-se inicialmente a primeira parte do modelo, onde são feitas relações apenas entre as variáveis x, assim como segue:

$$
\begin{array}{lll}
\sum_{i=1}^{n} x_{i,j} \leq 1 & \forall(1 \leq j \leq n) & (2\text{-}13)\\
\sum_{j=1}^{n} x_{i,j} \leq 1 & \forall(1 \leq i \leq n) & (2\text{-}14)\\
\sum_{i=1}^{n} x_{i,j} \geq \sum_{i=1}^{n} x_{i,j+1} & \forall(1 \leq j < n) & (2\text{-}20)
\end{array}
$$

A partir destas relações, encontrou-se as seguintes desigualdades válidas:

$$
\sum_{i=1}^{n} x_{i,j} \geq \sum_{v=j}^{n} x_{i,v} \qquad \forall(1 \leq j,i \leq n) \qquad (2\text{-}32)
$$

Para provar estas desigualdades, aponta-se, devido a 2-13, apenas duas possibilidades para $\sum_{i=1}^{n} x_{i,j}, 1 \leq i, j \leq n$:

Caso $\sum_{i=1}^{n} x_{i,j} = 0$, tem-se pelas desigualdades 2-20 que $x_{i,v} = 0, \forall(j \leq v \leq n, 1 \leq i \leq n)$, ou $\sum_{v=j}^{n} \sum_{i=1}^{n} x_{i,v} = 0$, o que verifica as desigualdades 2-32, uma vez que $\sum_{v=j}^{n} \sum_{i=1}^{n} x_{i,v} = 0$ implica em $\sum_{v=j}^{n} x_{i,v} = 0, \forall(1 \leq i \leq n)$.

Caso $\sum_{i=1}^{n} x_{i,j} = 1$, verifica-se também a desigualdade 2-32, uma vez que as desigualdades 2-14 implicam $1 \geq \sum_{v=j}^{n} x_{i,v}$.

Desigualdades válidas (2-33), (2-34) e (2-35) específicas do problema

Após analisar-se mais profundamente os resultados das relaxações lineares do modelo simplificado, formulou-se as restrições 2-33, 2-34 e 2-35. Estas podem ser validadas de forma simples, utilizando-se 2-23, 2-24 e 2-25.

$$F_{i,j} \geq t_j + \sum_{s=1}^{n} (g_s x_{s,j} + G_s \alpha_{s,j}) + w_i \alpha_{i,j} \qquad \forall(1 \leq i, j \leq n) \qquad (2\text{-}33)$$

$$F_{i,j} = F_{i,j-1} + (g_i x_{i,j} + (G_i + w_i)\alpha_{i,j}) \qquad \forall(1 \leq i \leq n, \; 2 \leq j \leq n) \qquad (2\text{-}34)$$

$$T = F_{i,n} \qquad \forall(1 \leq i \leq n) \qquad (2\text{-}35)$$

$$t_1 = 0 \qquad (2\text{-}23)$$

$$t_j \geq t_{j-1} + \sum_{i=1}^{n} (g_i x_{i,j-1} + G_i \alpha_{i,j-1}) \qquad \forall(2 \leq j \leq n) \qquad (2\text{-}24)$$

$$t_j + \sum_{i=1}^{n} (g_i x_{i,j} + (G_i + w_i)\alpha_{i,j}) = T \qquad \forall(2 \leq j \leq n) \qquad (2\text{-}25)$$

Dada uma solução qualquer para o problema, tem-se que, para todo $1 \leq u, v \leq n$:

Caso A: $x_{u,r} = 0, \forall(r \geq v + 1)$.

Neste caso, $F_{u,v} = T$ segundo 2-34 e 2-35. É necessário então provar que a desigualdade 2-33 com $i = u$ e $j = v$ é verificada. Utilizando-se as restrições 2-25, tem-se que

$$t_j + \sum_{s=1}^{n} (g_s x_{s,j} + G_s \alpha_{s,j}) + w_i \alpha_{i,j} \leq T \quad , \forall (1 \leq i, j \leq n),$$

garantindo desta forma que $F_{u,v} = T \geq t_v + \sum_{s=1}^{n} (g_s x_{s,v} + G_s \alpha_{s,v}) + w_u \alpha_{u,v}$.

Caso B: $\exists (r \geq v+1)$ tal que $x_{u,r} = 1$.

Neste caso, $F_{u,v} = t_r$, uma vez que, segundo 2-34 e 2-35, $F_{u,r} = T$ e $F_{u,v} = T - g_u - (w_u + G_u)\alpha_{u,r}$. Utilizando-se 2-25 e sabendo que $x_{u,r} = 1$, tem-se que $t_r = T - g_u - (w_u + G_u)\alpha_{u,r}$, o que implica $F_{u,v} = t_r$.

É necessário então provar que a desigualdade 2-33 com $i = u$ e $j = v$ é verificada. Como $x_{u,v} = 0$, tem-se que $\alpha_{u,v} = 0$ e conseqüentemente a restrição 2-33 para $i = u$ e $j = v$ torna-se

$$F_{u,v} \geq t_v + \sum_{s=1}^{n} (g_s x_{s,v} + G_s \alpha_{s,v}).$$

Como $r > v$, pelas desigualdades 2-24, tem-se que

$$t_r \geq t_v + \sum_{s=1}^{n} (g_s x_{s,v} + G_s \alpha_{s,v}),$$

que verifica a restrição uma vez que $F_{u,v} = t_r$.

Desigualdades válidas (2-31) que estipulam limites inferiores variáveis para as variáveis $\alpha_{i,j}$

Para a criação destas restrições, é necessária a obtenção de um limite inferior T_{min} para o *makespan*, como descrito na Seção 2.4.1. Com T_{min} é possível calcular o limite inferior variável $W'_{i,j}$ de $\alpha_{i,j}$ da forma apresentada a seguir.

Sabendo-se que o primeiro processador a receber dados não exigirá um tempo de comunicação e processamento menor que T_{min}, tem-se $g_i + (w_i + G_i)\alpha_{i,1} \geq T_{min}, \forall i$. Dessa forma, para todo processador P_i pode-se obter um limite inferior $W'_{i,1} = (T_{min} - g_i)/(w_i + G_i)$ para $\alpha_{i,1}$, impondo-se $\alpha_{i,1} \geq W_{i,1} x_{i,1}$. Sabendo-se o mínimo de carga que cada processador poderá receber caso venha a ser escolhido como primeiro a receber dados, pode-se calcular o período mínimo de tempo que o segundo processador poderá obter. Este período será igual ao menor tempo de processamento possível do primeiro processador, ou

seja, $\min\{W'_{i,1}w_i \mid 1 \leq i \leq n\}$. Ao seguir-se indutivamente esta técnica, o Algoritmo *CalcLimI* (Algoritmo 3) calcula $W'_{i,j}, \forall(1 \leq i,j \leq n)$.

Neste algoritmo, as variáveis L_j representam o período mínimo de tempo que o j-ésimo processador poderá obter para receber e processar dados. Sabendo-se o tempo mínimo que um processador P_i obterá caso seja o j-ésimo a receber dados, tem-se

$$W'_{i,j} = \frac{L_j - g_i}{w_i + G_i} \quad \forall(1 \leq i,j \leq n). \tag{2-36}$$

No caso do primeiro processador, $L_1 = T_{min}$, e para os demais,

$$L_{j+1} = \min\{W'_{i,j}w_i \mid 1 \leq i \leq n\}, \quad \forall(1 \leq j \leq n-1). \tag{2-37}$$

A prova que não é possível existir uma solução ótima onde algum processador P_i receba uma fração de carga $\alpha^*_{i,j} < W'_{i,j}$, sendo o *makespan* total $T^* \geq T_{min} = L_1$ é muito semelhante à prova realizada na definição de um limite superior variável na Seção 2.4.3, portanto será omitida.

Algoritmo 3 CalcLimI: Cálculo dos limites $W'_{i,j}$

Requer: T_{min}
Retorna: $W'_{i,j}, 1 \leq i,j \leq n$
1: $L_1 = T_{min}$
2: **para** $j = 1$ até n **faça**
3: **para** $i = 1$ até n **faça**
4: $W'_{i,j} \leftarrow (L_j - g_i)/(w_i + G_i)$
5: **fim para**
6: $L_{j+1} \leftarrow \min\{W_{i,j}w_i \mid 1 \leq i \leq n\}$
7: **fim para**

2.5

Nova heurística construtiva com retro-alimentação *HeuRet*

Como visto na Seção 2.3, dada uma ordem de envio pré-determinada, a distribuição ótima de dados é facilmente calculada. Desta forma, uma vez tendo a ordem ótima de envio, obtém-se a solução ótima, determinando-se o número de processadores a serem utilizados e a quantidade de carga a ser enviada a cada um. Sendo assim, representa-se uma solução S por um vetor $Sorder$, que indica a ordem de envio, $Smakespan$, que é o *makespan* da distribuição ótima

dada esta ordem de envio e SL^* que é o número ótimo de processadores a serem utilizados.

O vetor *Sorder* possui os índices dos processadores de forma que o envio seja feito segundo a ordem que aparece no vetor. Caso por exemplo $Sorder =< 2, 3, 1 >$, as cargas seriam enviadas na seguinte ordem: P_2, P_3 e P_1.

A heurística construtiva para o problema pode ser resumida como um método que gera uma boa ordem para o envio das cargas. A base da criação desta heurística reside no fato que, dado um período de tempo T e um processador P_i, sabe-se a quantidade de dados $\alpha_i = (T - g_i)/(w_i + G_i)$ que este processador deverá receber para ocupar todo o intervalo T. Verifica-se que o comportamento deste processador neste cenário equivale a um processador cujo poder computacional é o mesmo, porém com latência de comunicação nula e taxa de transmissão equivalente $1/G'_i = 1/(G_i + (g_i/\alpha_i))$. Toma-se por exemplo um processador P_1 com $w_1 = 10$, $G_1 = 5$ e $g_1 = 7$ recebendo uma carga $\alpha_1 = 10$. Neste caso, este processador seria equivalente a um processador P'_1 com $w'_1 = 10$, $g'_1 = 0$ e $G'_1 = 5 + 7/10$, que exigiria um mesmo tempo de comunicação de 57 e um mesmo tempo de processamento de 50.

Lembra-se que, conforme [8], a ordem ótima de envio para sistemas sem latência deve ser tal que os primeiros na ordem devem possuir taxas de transmissão maiores que os últimos. Levando-se em conta estas duas considerações, foi elaborada a heurística *HeuRet* (Algoritmo 4).

Esta heurística possui retro-alimentação uma vez que, a cada solução gerada de forma construtiva a partir de um limite para o *makespan*, pode-se reduzir este limite e gerar uma nova solução de forma construtiva.

Na linha 1 da heurística *HeuRet* gera-se a ordem inicial *order* de envio de dados, de modo que $G_{order[1]} \leq G_{order[2]} \leq G_{order[n]}$. Com esta ordem definida, utiliza-se na linha 2 o algoritmo *AlgRap* para o cálculo do *makespan* ótimo obtido com esta ordem de envio, o qual servirá de limite inicial para o *makespan* ótimo com uma ordem qualquer.

Tendo-se este limite, da linha 9 à 31 constrói-se uma nova ordem de envio. Inicialmente, considera-se uma fila de envio vazia (nenhum processador escolhido para receber dados). Calcula-se da linha 11 à 25 qual dos processadores (ainda não incluídos na fila) possui maior taxa de transmissão equivalente, dado o limite de tempo existente (conforme descrito anteriormente). Nota-se que valores negativos não são tolerados, fazendo com que $G'_j = \infty$ e $\alpha_j = 0$

para o processador P_j que possua latência maior que o *limite* (acarretando num α_j negativo). Uma vez escolhido o processador P_{j^*} com maior taxa de transmissão equivalente nas linhas 24 e 25, inclui-se este processador no final da fila de envio (linhas 26 e 27) e reduz-se do limite de tempo, o período $G'_{j^*}\alpha_{j^*}$ que P_{j^*} exigirá de comunicação. Nota-se que valores negativos para *limite* não são tolerados, sendo este não atualizado quando a atualização acarretaria em um limite negativo (linha 28 à 30). Tendo-se o limite atualizado, executa-se o *para-faça* da linha 9 à 31 novamente até ter-se criado uma ordem completa de envio.

Após escolhida a ordem, uma nova avaliação é feita utilizando-se o Algoritmo *AlgRap* (descrito na Seção 2.3), que determinará o *makespan* ótimo dada esta ordem. Se o valor encontrado for menor que o *makespan* da melhor solução encontrada, então o processo de ordenação recomeça. A retro-alimentação procede até que um *makespan* pior seja conseguido, o que indica o término do algoritmo, devolvendo a melhor solução obtida.

2.6 Busca local

Na seção anterior foi apresentado um método construtivo que gera soluções viáveis, porém não garante que tais soluções sejam ótimas. Com o objetivo de melhorar-se as soluções encontradas por esta heurística, foram desenvolvidos dois métodos de busca local. Métodos de busca local são métodos que, a partir de uma solução inicial, promovem pequenas modificações na busca por soluções melhores. Estes métodos podem ser vistos como procedimentos de busca do espaço de soluções, procurando sempre melhorar uma solução corrente.

Para o desenvolvimento de métodos de busca local é necessário primeiro a definição do conceito de vizinhança da solução. Nas implementações feitas foi utilizada uma vizinhança de tamanho $O(n^2)$, que consiste na troca da ordem de envio de dois processadores. Como apresentado na Seção 2.5, as soluções são representadas por um vetor com a ordem de envio que deve ser seguida e o *makespan* ótimo para a mesma. Desta forma, por exemplo, dada uma solução para um sistema com três processadores onde a ordem de envio é $< 1, 2, 3 >$, sua vizinhança será $< 2, 1, 3 >$, $< 3, 2, 1 >$ e $< 1, 3, 2 >$.

Definido o conceito de vizinhança, os métodos de busca local partem de

Algoritmo 4 HeuRet: Heurística construtiva com retro-alimentação

Retorna: $MelhorOrder$, $MelhorMakespan$ e $MelhorL^*$

1: Gere o vetor *order* com os índices de todos os processadores de modo que $G_{order[1]} \leq G_{order[2]} \leq G_{order[n]}$.
2: $limite \leftarrow$ *makespan* ótimo encontrado por $AlgRap(order)$
3: **repita**
4: $MelhorOrder \leftarrow order$
5: $MelhorMakespan \leftarrow limite$
6: $MelhorL^* \leftarrow l^*$ encontrado por $AlgRap(order)$
7: $incluidos \leftarrow \emptyset$
8: **para** $i \leftarrow 1$ até n **faça**
9: $j^* \leftarrow 0$
10: **para** $j \leftarrow 1$ até n **faça**
11: **se** $j \notin incluidos$ **então**
12: **se** $limite - g_j \leq 0$ **então**
13: $alpha_j \leftarrow 0$
14: $G'_j \leftarrow \infty$
15: **senão**
16: $alpha_j \leftarrow (limite - g_j)/(w_j + G_j)$
17: $G'_j \leftarrow G_j + (g_j/alpha_j)$
18: **fim se**
19: **se** $j^* = 0$ ou $G'_j \leq G'_{j^*}$ **então**
20: $j^* \leftarrow j$
21: **fim se**
22: **fim se**
23: **fim para**
24: $order[i] \leftarrow j^*$
25: $incluidos \leftarrow incluidos \cup \{j^*\}$
26: **se** $alpha_{j^*} \neq 0$ e $limite - G'_{j^*} \times alpha_{j^*} \geq 0$ **então**
27: $limite \leftarrow limite - G'_{j^*} \times alpha_{j^*}$
28: **fim se**
29: **fim para**
30: $limite \leftarrow$ *makespan* ótimo encontrado por $AlgRap(order)$
31: **até** $limite > MelhorMakespan$

uma solução inicial e buscam em sua vizinhança soluções com *makespan* melhor (soluções aprimorantes). Uma vez escolhida uma solução vizinha aprimorante, a busca continua na vizinhança da mesma, repetindo o processo até encontrar um ótimo local, ou seja, até que não seja mais possível encontrar soluções aprimorantes.

Neste trabalho foram implementadas duas técnicas de busca local, a melhoria iterativa e a descida rápida. Enquanto o método de melhoria iterativa seleciona o primeiro vizinho aprimorante para recomeçar a busca, a descida rápida escolhe o melhor vizinho aprimorante. Desta forma a descida rápida necessita avaliar a vizinhança inteira antes de recomeçar a busca. Freqüentemente na prática ambas técnicas alcançam o mesmo ótimo local, porém a melhoria iterativa tende a levar menos tempo para convergir. É importante também salientar que o método de descida rápida tende a convergir prematuramente para ótimos locais não globais [19].

O Algoritmo 5 é usado para execução de ambas as técnicas, sendo seu comportamento definido pelo parâmetro $MelhoriaIterativa$ que será $Verdadeiro$ caso o comportamento desejado seja este, ou $Falso$ caso o comportamento de descida rápida seja o desejado. Neste mesmo algoritmo é utilizado o procedimento *swap* que recebe dois parâmetros e troca seus valores, de forma que o primeiro fique com o valor do segundo e vice versa.

Algoritmo 5 BL: Busca local

Requer: $IOrder$, $IMakespan$ e $MelhoriaIterativa$
Retorna: $MelhorOrder$ e $MelhorMakespan$

```
1: MelhorOrder ← IOrder
2: MelhorMakespan ← IMakespan
3: repita
4:    melhoria ← Falso
5:    order ← MelhorOrder
6:    sair ← Falso
7:    para i = 1 até n − 1; enquanto sair = Falso faça
8:       para j = i + 1 até n; enquanto sair = Falso faça
9:          swap(order[i], order[j])
10:         makespan ← makespan ótimo encontrado por AlgRap(order)
11:         se makspan < MelhorMakespan então
12:            MelhorOrder ← order
13:            MelhorMakespan ← makespan
14:            melhoria ← Verdadeiro
15:            se MelhoriaIterativa então
16:               sair ← Verdadeiro
17:            fim se
18:         fim se
19:         swap(order[i], order[j])
20:      fim para
21:   fim para
22: até ¬melhoria
```