Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

MAC 5758 - Introdução ao Escalonamento de Aplicações
Tema do projeto: Escalonamento com reserva de recursos

Vinicius Gama Pinheiro
vinicius@ime.usp.br

Resumo

Este projeto consiste em estudar trabalhos relacionados ao problema de escalonamento com reserva de recursos no processamento paralelo de aplicações. Em especial, será analisado o trabalho publicado por Lionel Eyraud-Dubois, Grégory Mounié e Denis Trystam intitulado "Analysis of Scheduling Algorithms with Reservations"que se concentra no estudo de algoritmos de *list scheduling* tendo como background os trabalhos publicados por Graham.

Esse trabalho é uma análise do problema de escalonamento de um conjunto de tarefas independentes em um computador paralelo homogêneo, tal como um aglomerado ou uma grade computacional dedicada. Nessas arquiteturas é comum que os middlewares disponham de ferramentas que possibilitam a reserva de uma quantidade de recursos para um período futuro. Os autores inicialmente fazem uma análise preliminar do problema a partir de um modelo básico, sem reserva de recursos, demonstrando que mesmo nesse caso, o problema é de difícil solução (NP-Hard). Em seguida, algumas restrições são inseridas no modelo, o que torna possível a obtenção de *lower* e *upper bounds* e garantias de desempenho para o problema de escalonamento com reservas.

1 Introdução

O escalonamento de tarefas em máquinas paralelas é uma área repleta de desafios. As inúmeras variáveis encontradas nos ambientes de execução e os diferentes tipos de tarefas a serem considerados, modelam problemas complexos que por vezes não possuem soluções que podem ser implementadas em tempo polinomial. Nessa área, é natural que problemas do mundo real sejam transportados para um modelo teórico, afim de que possamos considerar elementos que sejam relevantes e desconsiderar outros que não sejam relevantes para a solução. Neste trabalho, estudaremos o escalonamento de aplicações em sistemas com reservas de recursos. Essas

reservas constituem uma restrição comumente encontrada em sistemas distribuídos reais, tais como grades e aglomerados.

Para direcionar o nosso estudo sobre escalonamento com reservas, iremos explorar o trabalho publicado por Lionel Eyraud-Dubois, Grégory Mounié e Denis Trystam intitulado "Analysis of Scheduling Algorithms with Reservations". Esse artigo analisa o problema de escalonar um conjunto de trabalhos independentes em uma arquiteutra de computação paralela homogênea, na qual um ou mais processadores podem estar reservados para aplicações específicas em períodos futuros. Os autores demonstram que, em um cenário no qual as reservas podem ser realizadas sem restrições não é possível sequer obter soluções aproximadas. Isso faz com que o escopo da análise seja reduzido aos cenários nos quais as reservas obedecem algumas restrições que, segundo os autores, são plausíveis de serem encontradas em sistemas reais.

Essa análise é fortemente baseada nos trabalhos publicados por Ronald L. Graham, em especial no trabalho que estabelece limites inferiores para algoritmos de list scheduling onde os recursos limitam-se a processadores. Referências aos trabalhos de Chung-Yee Lee sobre escalonamento com restrições de disponibilidade também são referenciadas com frequência durante a análise apresentada no trabalho.

Dessa forma, para entender melhor o problema sobre escalonamento com reservas, precisamos nos aprofundar um pouco nos seguintes problemas:

- Limites para escalonamento em arquiteturas paralelas com restrição de recursos
- Escalonamento com restrições de disponibilidade
- Escalonamento online em máquinas paralelas

Na próxima seção fazemos uma descrição mais detalhada sobre o trabalho publicado por Eyraud-Dubois et al.

2 Artigo: Analysis of Scheduling Algorithms with Reservations

Neste artigo, inicialmente é proposto o modelo padrão de um sistema de multiprocessamento composto por um número fixo n de processadores, um conjunto finito de tarefas a serem executadas, cada uma com um tempo específico para ser executada, e uma ordenação parcial das tarefas para indicar precedência de execução entre elas.

É considerado um sistema composto de n processadores idênticos. A função do sistema é executar um certo conjunto de tarefas T = T_1, ..., T_r sendo que é ordenado por uma relação "¡". Essa relação estabelece a ordem de execução das tarefas da seguinte forma: se T_i ¡ T_j então a execução de T_i deve ser encerrada antes do início de T_j. À cada tarefa T_i é associada um número real p_i representando o tempo de execução requerido por T_i e o modelo é não preemptivo. A ordem na qual as tarefas são escolhidas é determinada da seguinte forma: uma lista de L = T_{i1}, ..., Tir é dada como uma permutação de T. O sistema inicia no tempo $t = 0$ e começa executando T. O tempo w é considerado o tempo no qual todas as tarefas de T estão encerradas. Nesse momento é citado o resultado publicado em outro artigo do Graham intitulado "Bounds on multiprocessing timing anomalies" [2]. Nesse artigo é estabelecido que, dado T' = T'_1, ..., T'_r com T_i ¡ T_j T'_i ¡ T'_j e para todo $p_i \leq p'_i$ para todo i e j, e sendo T' executado pelo sistema usando uma lista L', então o tempo de execução w' satisfaz

$w'/w \leq 2 - 1/n.$ (1)

A partir disso, os autores analisam uma versão extendida desse problema e, para tal, são definidas algumas notações extras. É definido F(Ti) = [r_i, $r_i + p_i$], onde r_i é o tempo no qual a execução de T_i foi iniciada e f(t) corresponde ao conjunto de tarefas que está sendo executada no tempo t. A restrição que corresponde ao número máximo de processadores pode ser expresso através de $|f(t)| \leq n$ para todo $t \in [0, w]$.

No trabalho original publicado por Garey e Graham eles assumem um conjunto de recursos R = R_1 , ..., R_s e definem propriedades para esses recursos.

Mas, no que se refere ao problema de escalonamento de tarefas em máquinas, vamos nos ater somente ao suproblema onde o único recurso são processadores, isto é, s = 1 já que, a partir daí, o trabalho publicado por Lionel Eyraud-Dubois, Grégory Mounié e Denis Trystam [1] propõe uma prova novas e mais simples para o limite de Graham. Eles definem p_max como o tempo de execução máximo das tarefas $max_1 \leq i \leq n(p_i)$ e W(I) como o trabalho total do algoritmo de lista L, definida por W(I) = $\sum I \leq i \leq n.pi.qi$. O f(t) é definido como It = $i \in [1..n]|r_i \leq t \leq r_i + p_i$ e r(t) é definido como o número de máquinas usadas no tempo t pelo algoritmo de lista L, onde $r(t) = \sum_{i \in L} q_i$.

Lemma 1

Para todo t, com $t' \in [0, C_max[$, $t' \geq t + p_max \rightarrow r(t) + r(t') > m$

Prova: Se $t' \geq t + p_max$, então necessariamente $I'_t \cup I_t =$. Por outro lado, já que $t' \leq C_max$, então há pelo menos uma tarefa T_i executando no tempo t'. O algoritmo L escolheu não iniciar a tarefa no tempo t. Por definição, isto significa que a tarefa T_i não poderá ser executada junto com as tarefas de I_t . Já que a única causa para isso é a falta de recursos, nos temos que $r(t)+qi > m$. Como tanto r(t) quanto r(t') são inteiros, pode-se escrever $r(t)+r(t') > m+1$.

A partir disso o resultado principal pode ser estabelecido:

Teorema 2

Se L é um algoritmo de lista, então para cada instância T com r máquinas,

$C_max(I) \leq (2?`1/m)C * max(I)$

A prova para esse teorema é realizada por contradição, isto é, considera-se uma instância I com m processadores e tenta-se provar que se existe um número real x tal que $C_max \geq (2 - x)C * max$, então $x \geq 1/m$. Isso leva a uma integral e através de sucessivas simplificações chega-se ao resultado esperado ($x \geq 1/m$).

3 Escalonamento clássico (RIGIDSCHEDULING

Nesta parte do trabalho analisaremos um modelo clássico de escalonamento (chamado pelos autores do artigo de RIGIDSCHEDULING). Esse modelo servirá como base para um outro modelo com mais restrições quanto ao uso dos recursos, que será explicado na parte 4 deste trabalho.

No modelo RIGIDSCHEDULING, um conjunto de n aplicações independentes (*jobs*) são escalonadas para serem processadas em m processadores idênticos. Cada trabalho j requer o uso de q_j processadores. A duração de cada trabalho é denotado por p_j. Neste caso, uma solução de escalonamento para este modelo é denotado por um conjunto de n tempos de início σ_i, com

i de 1 até n. Obviamente, se procuramos uma solução que seja realmente possível, o número de máquinas utilizadas pelas aplicações nunca pode exceder o número de máquinas disponíveis. Dessa forma, temos que:

$\forall t \geq 0, \sum_{i \in I_t} q_i \leq m$

onde $I_t = \{i \in [1..n] | \sigma_i \leq t < \sigma_i + p_i\}$

O objetivo é minimizar o *makespan*, isto é, o maior tempo de término entre todas as tarefas:

$C_max = max\sigma_i + p_i$, com i de 1 a n

Portanto, a definição formal para este problema, usando a notação de 3 campos aprendida durante as aulas, é

$P_j | p_j, | sigma_j | C_max.$

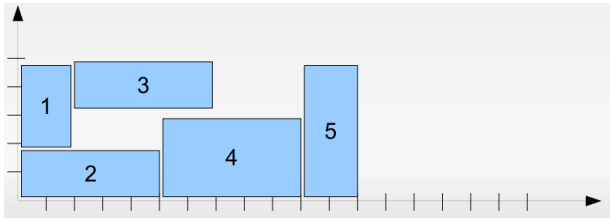

Figura 1: Modelo RIGIDSCHEDULING

Na Figura 1 temos um exemplo de escalonamento com RIGIDSCHEDULING. os índices de cada aplicação indicam a ordem na qual elas foram submetidas. As aplicações são compostas por uma ou mais tarefas que são executadas em paralelo, ocupando um ou mais processadores.

Sabe-se que o escalonamento de tarefas sequenciais em dois processadores idênticos é um problema NP-Difícil (é igual ao problema da partição e só pode ser tratado com algoritmos de aproximação). Sendo assim, como o modelo RIGIDSCHEDULING é uma especialização desse último, o RIGIDSCHEDULING também é NP-Difícil.

A execução dos trabalhos é geralmente representada por um gráfico de Gantt, no qual blocos bidimensionais representam uma aplicação a ser executada em um conjunto de máquinas

específico. Esses blocos podem passar a impressão de que as aplicações necessitam de máquinas contíguas, quando na realidade, qualquer subconjunto de máquinas pode ser usada para escalonar uma aplicação, desde que tenham o mesmo número de máquinas.

Um dos algoritmos mais comuns para a execução de tarefas no modelo RIGIDSCHEDULING é o First Come First Served (FCFS). Neste modelo as aplicações submetidas são organizadas em uma fila, ordenada pela tempo de submissão. Assim que existam recursos disponíveis suficientes, a aplicação que está na fila há mais tempo é escalonada. Esse modelo possui a vantagem de ser simples de ser entendido e implementado. Contudo, leva a um desperdício de recursos, dado que tarefas que utilizam um número grande de recursos não serão escalonadas até que recursos suficientes se tornem disponíveis, não aproveitando, assim, o tempo ocioso dos recursos que já estão disponíveis e que poderiam estar executando tarefas que exigem menos recursos.

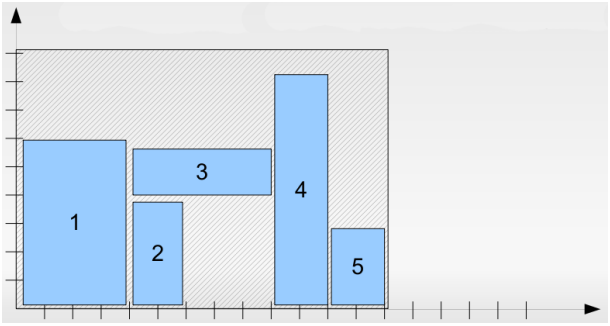

Figura 2: Escalonamento com FCFS

Na Figura 2, temos um exemplo de escalonamento de RIGIDSCHEDULING com a estratégia first-come-first-served: nenhuma tarefa com índice i inicia a sua execução antes da tarefa anterior $i-1$, isto é, o escalonamento respeita a ordem de chegada das aplicações.

3.1 Backfilling

Uma estratégia comum que busca aproveitar esse tempo ocioso é a estratégia de *backfilling*. Essa estratégia consiste em fazer com que tarefas possam ser escalonadas com início anterior ao das tarefas previamente escalonadas. Dessa forma, alguns espaços vazios não preenchidos pelas tarefas já escalonadas poderiam ser ocupados, evitando-se desperdício de recursos.

A depender da prioridade atribuída as tarefas ainda não escalonadas, diversas estratégias de *backfilling* podem ser elaboradas. Estratégias mais conservadoras podem, por exemplo, preencher os espaços vazios mas sempre respeitando o tempo de início das aplicações já escalonadas. Estratégias mais agressivas pode "empurrar"as tarefas já escalonadas para serem iniciadas mais

tarde, desde que isso implique num uso mais racional dos recursos.

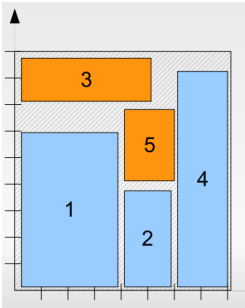

Figura 3: Escalonamento com FCFS e *backfilling*

Na Figura 3, temos um exemplo de escalonamento de RIGIDSCHEDULING com estratégia de *backfilling*: neste caso uma tarefa com índice i pode iniciar a sua execução antes da tarefa anterior $i-1$ desde que existam processadores disponíveis. O escalonamento não precisa respeitar a ordem de chegada das aplicações. Essa estratégia traz benefícios pois permite que aplicações submetidas posteriormente ocupem espaços de processamento ocioso que surgiram quando do escalonamento das aplicações anteriores.

O FCFS não oferece nenhuma garantia de desempenho em relação ao *makespan*, mesmo com uma estratégia de *backfilling* conservadora. Contudo, se adotarmos a estratégia mais agressiva, na qual qualquer tarefa a ser escalonada pode alterar o tempo de início de qualquer tarefa já escalonada, isso nos leva ao trabalho publicado por Granham sobre algoritmos de lista no escalonamento com restrição de recursos. Este trabalho estabelece um *upper bound* de $s + 1$ para o desempenho em relação ao tempo ótimo, no qual s corresponde ao número de tipos de recursos que estão sendo compartilhados. No caso abordado, só há processadores como os recursos a serem utilizados e, assim, a garantia de desempenho é de 2, o que pode ser reduzido para $2 - 1/m$, segundo uma prova fornecida pelos autores do artigo.

Na quarta parte do trabalho, veremos como esse modelo é extendido com a introdução de restrições ao uso dos recursos e quais resultados teóricos são obtidos a partir dessas modificações.

4 Escalonamento com reservas (RESASCHEDULING)

Nesta parte do trabalho, analisamos um modelo restrito de escalonamento (chamado pelos autores do artigo de RESASCHEDULING). Esse modelo surge a partir da inclusão de restrições ao modelo clássico RIGIDSCHEDULING visto na parte 3 do trabalho.

No modelo RESASCHEDULING, um conjunto de n aplicações independentes (*jobs*) são escalonadas para serem processadas em m processadores idênticos. Cada aplicação T_i (com i de 1 até n) requer o uso de $q_i \in [1..m]$ processadores. A duração de cada trabalho é denotado por $p_i > 0$. Além disso, cada instância do RESASCHEDULING possui n' reservas que são modeladas da mesma forma que as aplicações, isto é, cada reserva R_j (com j de $n+1$ até $n+n'$) é caracterizada por uma duração $p_j > 0$, requer o uso de $q_j \in [1..m]$ processadores e tem um tempo de início r_j.

Obviamente, se procuramos uma solução que seja realmente possível, o número de máquinas utilizadas pelas reservas nunca pode exceder o número de máquinas disponíveis. Dessa forma, temos que:

$\forall t \geq 0, \sum_{j \in J_t} q_j \leq m$

onde $J_t = \{j \in [n+1..n+n'] | r_j \leq t < r_j + p_j\}$

A partir disso os autores definem uma função de indisponibilidade U para cada tempo t:

$U(t) = \sum q_j, j \in J_t$

A função U, portanto, define o número de máquinas utilizadas pelas reservas em um dado tempo t. Uma instância de RESASCHEDULING só é possível se para todo tempo t, $U(t) \leq m$.

Neste caso, uma solução de escalonamento para este modelo é denotado por um conjunto de n tempos de início σ_i, com i de 1 até n. Essa solução precisa respeitar o limite de disponibilidade imposto por $U(t)$, isto é:

$\forall t \geq 0, \sum q_i \leq m - U(t)$, com $i \in I_t$, onde $I_t = \{i \in [1..n] | \sigma_i \leq t < \sigma_i + p_i\}$

Na figura 4, as aplicações em vermelho são, na verdade, reservas que estavam presentes anteriormente à submissão das aplicações (em azul). A restrição imposta por este modelo é que, mesmo com o uso de backfilling, não é permitido alterar a data de início das reservas.

Essa modificações introduzidas no modelo RIGIDSCHEDULING, que culminou no modelo RESASCHEDULING, claramente mantém o nosso modelo como NP-Difícil.

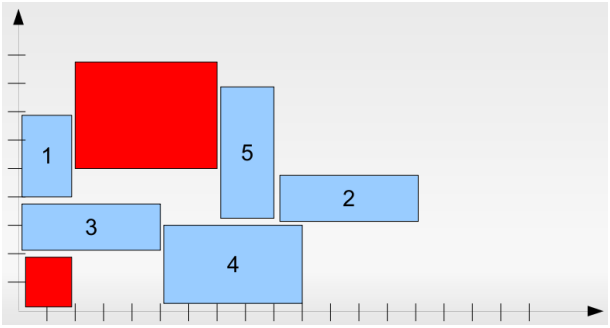

Figura 4: Modelo RESASCHEDULING

Sabe-se que o escalonamento de tarefas sequenciais em dois processadores idênticos é um problema NP-Difícil (é igual ao problema da partição e só pode ser tratado com algoritmos de aproximação). Sendo assim, como o modelo RIGIDSCHEDULING é uma especialização desse último, o RIGIDSCHEDULING também é NP-Difícil. É impossível sequer implementar um algoritmo de aproximação em tempo polinomial para resolvê-lo. Isso vem do fato de que podemos inserir uma reserva muito longa e muita larga (i.e. que requer muitas máquinas) com início no C_max de uma solução ótima. Nesse caso, é possível gerar qualquer solução não ótima com um makespan de valor arbitrário. A prova da inexistência de um algoritmo de tempo polinomial para o RESASCHEDULING é feito a partir de uma redução a partir do problema 3-partição e pode se encontrada no artigo já citado que é o foco deste trabalho.

Diante da impossibilidade de uma solução de tempo polinomial, os autores do artigo procuram inserir algumas restrições ao modelo, sendo estas, calcadas nos sitemas do mundo real. Sendo assim, eles procuram analisar dois modelos derivados: escalonamento com reservas não-crescentes e escalonamento com limite de reservas.

Na primeira derivação do modelo, é imposto que, uma vez que o sistem possua reservas, elas começam no tempo 0 e descrescem em largura ao longo do tempo, isto é, dado que uma reserva ocupa q_{jt_1} máquinas num tempo t_1, ela ocupará $q_{jt_2} \leq q_{jt_1}$ num tempo $t_2 = t_1 + 1$.

Neste modelo, representado na Figura 5, uma reserva nunca é sucedida por uma outra reserva que ocupe mais máquinas do que a que esta ocupa. Isso quer dizer que, se soubermos o q_j, isto é, o número de processadores ocupados por uma reserva j, temos a informação de que de r_j em diante, $m - q_j$ máquinas sempre estarão disponíveis.

Na segunda derivação, é imposto que a quantidade de recursos em reserva num dado tempo t não pode ultrapassar $(1 - \alpha)m$ máquinas, sendo que α corresponde a um número entre 0 e 1 ($\alpha \in]0; 1]$). Por outro lado, o número de recursos que as aplicações podem, juntas, utilizar em

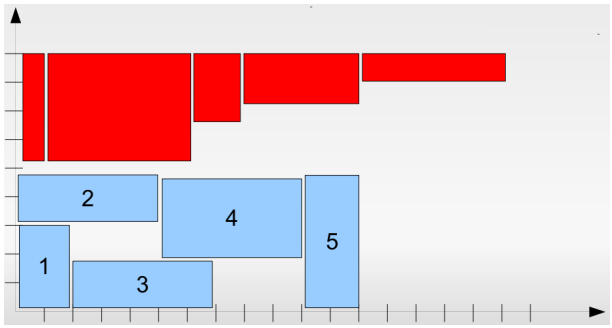

Figura 5: Modelo RESASCHEDULING com reservas não-crescentes

um determino tempo também não pode exceder αm.

Os autores argumentam que essas restrição são muito comuns de serem encontradas na prática, já que administradores de cluster e grades dedicadas frequentemente impõem restrição na quantidade de recursos que as aplicação podem requisitar, afim de evitar abuso por parte dos usuários. Mais formalmente, esse modelo derivado pode ser definido da seguinte forma:

$\forall t \geq 0, U(t) = \sum q_j \leq (1 - \alpha)m$ para $j \in J_t$
$forallt \leq n, q_i \leq \alpha m$

Neste modelo, chamado pelos autores de α-RESASCHEDULING, qualquer instâcia terá ao menos αm máquinas disponíveis, tornando possível o escalonamento de ao menos uma tarefa em qualquer instante t.

É neste ponto que os autores definem, então, um limite inferior (*lower bound*) e um limite superior (*upper bound*) para o problema de escalonamento com reservas (dado modelo com restrição de reservas α-RESASCHEDULING):

Limite inferior: Pode-se garantir que não é possível um desempenho melhor do que $2/\alpha - 1 + \alpha$ 2.

Limite superior: Pode-se garantir um desempenho de ao menos $2/\alpha$.

As provas para os limites inferior e superior descritas no corpo e no apêndice do artigo em questão. Os autores finalizam o artigo destacando mais uma vez que o modelo α-RESASCHEDULING possui um considerável valor agregado já que as restrições nele modeladas pode ser normalmente

encontradas em sistemas reais. Além disso, eles ressaltam que o limite superior encontrado não é muito distante do limite inferior, e que trabalhos futuros envolve o estudo de algumas variantes de escalonamento em lista para o mesmo modelo, com o objetivo de reduzir ainda mais o limite superior.

Referências

[1] Lionel Eyraud Dubois, Grégory Mounié, and Denis Trystram. Analysis of Scheduling Algorithms with Reservations. In *IPDPS 2007*, pages 1–8, Long Beach, California États-Unis d'Amérique, 03 2007. IEEE.

[2] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.