
Sistemas de Tempo Real: Multiprocessadores

Rômulo Silva de Oliveira
Departamento de Automação e Sistemas – DAS – UFSC

romulo@das.ufsc.br
<http://www.das.ufsc.br/~romulo>

Referências

- A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms
 - John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson, and Sanjoy Baruah
 - Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Chapman & Hall/CRC Press (2003)

Introdução

- Sistemas multiprocessadores são agora comuns
 - Difícil comprar um desktop com apenas um processador
 - Sistemas de tempo real não são diferentes
- Arquiteturas multiprocessadoras variam bastante
- Quando não existe acesso comum à memória física (acoplamento fraco)
 - Sistemas Distribuídos
- Quando existe acesso comum à memória física (acoplamento forte)
 - Sistemas Multiprocessadores
- Máquinas multiprocessadoras variam muito
 - Soluções single-chip, com poucos processadores
 - Soluções com dezenas, centenas, milhares de processadores
 - Processamento de sinais, como os synthetic-aperture radar systems

Introdução

- Escalonamento tempo real foi muito estudado no passado para sistemas com um único processador
- Entretanto, pesquisadores estão apenas começando a entender os aspectos fundamentais do escalonamento tempo real em multiprocessadores
- O que acontece quando são vários processadores e não apenas um ?
 - Muda tudo !

Terminologia

- Um conjunto de tarefas τ é escalonável por um algoritmo A
Se A garantir os requisitos temporais de todas as tarefas em τ
- τ é dito viável com uma classe C de algoritmos de escalonamento
Se τ for escalonável por algum algoritmo $A \in C$.
- Um algoritmo A é dito ser ótimo com respeito a classe C
Se $A \in C$
e
 A escalona corretamente todos os conjuntos de tarefas que foram viáveis com a classe C

Escalonamento Global versus Particionado

- As duas principais abordagens para escalonar tarefas periódicas em multiprocessadores são:

Escalonamento Particionado

Escalonamento Global

Escalonamento Global

- No escalonamento global todas as tarefas aptas são mantidas em uma única fila ordenada pela prioridade
- O escalonador global seleciona para execução as tarefas com prioridade mais alta desta fila (conforme número de processadores)
- Infelizmente, no escalonamento global, usar algoritmos de escalonamento que são ótimos para monoprocessador
 - Tais como rate-monotonic (RM) e earliest-deadline-first (EDF)
- Pode resultar em utilizações baixíssimas do multiprocessador

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 7

Escalonamento Particionado

- No particionamento, cada tarefa é alocada a um único processador
 - No qual todos os seus jobs irão executar
- Processadores são escalonados de forma independente
- Principal vantagem:
 - Reduzir o problema de escalonar um multiprocessador ao problema de escalonar um conjunto de monoprocessadores
- Principais desvantagens:
 - Achar uma alocação ótima das tarefas aos processadores é um problema do tipo “bin-packing”, o qual é NP-hard no sentido forte
 - Tarefas são usualmente particionadas com heurísticas sub-ótimas
 - Existem conjuntos de tarefas que são escalonáveis se e somente se as tarefas não forem particionadas
- Entretanto, particionamento é muito usado na prática (determinismo)

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 8

Migração de Tarefas e Jobs

- A migração de tarefas entre processadores não é usual
 - Migrar: transferir o contexto de um processador para outro
- Em muitos sistemas o custo associado com a migração é proibitivo
 - Sistemas fracamente acoplados
- Desenvolvimentos em arquitetura de computadores diminuíram este custo
 - Multiprocessadores single-chip
 - Redes de interconexão muito rápidas
- Para sistemas de tempo real a teoria somente agora começa a permitir a análise de escalonabilidade de sistemas que permitem migração
- Resultado: particionamento tem sido a abordagem preferida na prática
 - Porém resultados recentes mostram que algoritmos de escalonamento que permitem migração são competitivos em termos de escalonabilidade apesar dos overheads

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 9

Classificação conforme a migração

- As soluções podem ser classificadas conforme o grau de migração permitido
- (1mn) nenhuma migração é permitida
 - particionamento de tarefas
- (2mj) migração é permitida somente na fronteira entre jobs
 - Particionamento dinâmico no nível de jobs
- (3mq) migração irrestrita é permitida
 - jobs podem migrar no meio da sua execução

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 10

Classificação conforme a migração

- (1mn) Nenhuma Migração
 - As tarefas são particionadas entre os processadores
 - Todos os jobs gerados por uma tarefa devem executar no processador ao qual ela foi alocada
- (2mj) Migração de Jobs
 - Cada job deve executar completamente em um mesmo processador
 - Diferentes jobs de uma mesma tarefa podem executar em processadores diferentes
 - Contexto de job não migra, mas contexto de tarefa sim
- (3mq) Migração Irrestrita
 - Nenhuma restrição é colocada sobre migração entre processadores

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 11

Uso de Prioridades

- Algoritmos de escalonamento normalmente executam nos mesmos processadores que as tarefas da aplicação
- É importante que tais algoritmos sejam simples e eficientes
- A cada instante, uma prioridade é associada com cada job ativo
- Os jobs de prioridade mais alta são selecionados para execução nos vários processadores disponíveis
- Um job é dito estar ativo no instante t se
 - O job chegou em t ou antes disto
 - Seu deadline absoluto é depois de t
 - Ele ainda não completou sua execução

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 12

Classificação conforme as prioridades

- As soluções também podem ser classificadas conforme a complexidade do esquema de prioridades
- Soluções podem ser classificadas em
- (1pf) prioridade fixa
 - Exemplo: RM
- (2pj) prioridade dinâmica, mas fixa para cada job
 - Exemplo: EDF
- (3pq) prioridades completamente dinâmicas
 - Exemplo: Least-Laxity-First (LLF)

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 13

Classificação Conforme as Prioridades

- (1pf) Prioridade fixa
 - Cada tarefa recebe uma prioridade única e todos os seus jobs executam com esta prioridade
 - Exemplo é o Rate Monotonic
- (2pj) Prioridade dinâmica, fixa para cada job
 - Para cada par de jobs J_i e J_j , se J_i tem prioridade mais alta do que J_j em algum momento, então J_i sempre terá prioridade mais alta que J_j
 - Exemplo é o Earliest Deadline First
- (3pq) Prioridades completamente dinâmicas
 - A prioridade relativa entre dois jobs quaisquer pode mudar a qualquer momento
 - Exemplo é o Least Laxity First

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 14

Classificação Conforme as Prioridades

- Algoritmos com prioridade dinâmica irrestrita são uma generalização dos algoritmos com prioridade dinâmica por job
- Algoritmos com prioridade dinâmica por job são uma generalização dos algoritmos com prioridade fixa
- Em monoprocessadores, a diferença entre algoritmos com prioridade dinâmica irrestrita e por job não é relevante pois:
 - EDF é ótimo
 - EDF é prioridade dinâmica por job (menos overhead)
- Em multiprocessadores,
 - Algoritmos com prioridade dinâmica irrestrita são estritamente mais poderosos que algoritmos com prioridade dinâmica por job

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 15

Outra classificação: Work-Conserving ou Não

- Algoritmos work-conserving:
 - Um processador nunca é deixado livre enquanto um job ativo existir
- Em multiprocessadores existe uma exceção:
 - Quando as políticas de migração impedem a tarefa de migrar para o processador livre
- Um job é dito estar ativo no instante t se
 - O job chegou em t ou antes disto
 - Seu deadline absoluto é depois de t
 - Ele ainda não completou sua execução
- A maioria dos algoritmos de escalonamento para tempo real são work-conserving

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 16

Outra Classificação: Preemptivo ou Não

- Preempção
 - Suspender o job antes do seu término, para retomá-lo mais tarde, a partir do ponto de sua execução onde foi suspenso
- Algoritmos preemptivos são superiores em termos de escalonabilidade
 - Existem conjuntos de tarefas que somente são viáveis se escalonados por algoritmos preemptivos
- Ausência de preempção não garante exclusão mútua em multiprocessadores
- Algoritmos preemptivos geram mais overhead
 - Gasta processador para fazer as trocas de contexto devido à preempção
- Algoritmos preemptivos sujam mais a cache do processador
 - Fator importante quando existe cache por processador

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 17

Definições: Classes de Escalonamento

- Um algoritmo de escalonamento é (x, y) -restrito para $x \in \{1, 2, 3\}$ e $y \in \{1, 2, 3\}$,
- Se estiver na classe de prioridade x e na classe de migração y
 - x e y correspondem aos rótulos definidos antes
- Por exemplo, um algoritmo $(2,1)$ -restrito usa
 - Prioridades dinâmicas por job
 - Particionamento
- Por exemplo, um algoritmo $(1,3)$ -restrito usa
 - Prioridades fixas
 - Migração irrestrita
- Um par ordenado denotado por $\langle x,y \rangle$ denomina o conjunto de todos os conjuntos de tarefas que são viáveis quando é usado escalonamento (x,y) -restrito

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 18

Classes de Escalonamento

	1. Prioridade fixa	2. Prioridade dinâmica por job	3. Prioridade qualquer
1. Particionado	(1,1)-restrito	(2,1)-restrito	(3,1)-restrito
2. Migração de job	(1,2)-restrito	(2,2)-restrito	(3,2)-restrito
3. Migração livre	(1,3)-restrito	(2,3)-restrito	(3,3)-restrito

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 19

Comparação entre Classes de Escalonamento

- Premissas:
 - Apenas algoritmos work-conserving
 - Apenas algoritmos preemptivos
- A eficácia de um algoritmo de escalonamento depende:
 - Do seu custo de implementação (overhead) e
 - Da sua habilidade em escalonar conjuntos de tarefas viáveis

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 20

Comparação entre Classes de Escalonamento

- É geralmente verdade que o custo de implementação (runtime overhead) é maior para modelos mais gerais e menor para modelos menos gerais
- O overhead de um algoritmo (w,x) -restrito é menor ou igual ao overhead de um algoritmo (y,z) -restrito se $y \geq w$ e $z \geq x$
- Abaixo é apresentado um estudo sobre as diferentes habilidades em escalonar conjuntos de tarefas viáveis, apresentadas pelas diferentes classes de algoritmos de escalonamento

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 21

Comparação entre Classes de Escalonamento

- Em termos de escalonabilidade as comparações não são tão simples
- Existem 3 possíveis relacionamentos entre as classes de escalonamento (w,x) -restrito e (y,z) -restrito
- Elas estão baseadas em um entendimento parcial dos relacionamentos que podem ter as seguintes formas:

Qualquer conjunto de tarefas em $\langle w,x \rangle$ está também em $\langle y,z \rangle$

$$\langle w, x \rangle \subseteq \langle y, z \rangle$$

ou

Existe pelo menos um conjunto de tarefas que está em $\langle w,x \rangle$ mas não está em $\langle y,z \rangle$

$$\langle w, x \rangle \not\subseteq \langle y, z \rangle$$

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 22

Relacionamento possível entre Classes: $\langle y,z \rangle C \langle w,x \rangle$

- $\langle y, z \rangle C \langle w, x \rangle$ (onde C significa subconjunto próprio)
 - A classe de algoritmos (w,x) -restritos é estritamente mais poderosa que a classe dos algoritmos (y, z) -restrito
- Ou seja, qualquer conjunto de tarefas que é viável com a classe (y, z) -restrito é também viável com a classe (w,x) -restrito
- Além disto, existem pelo menos um conjunto de tarefas que é viável com a classe (w,x) -restrito mas não é com a classe (y,z) -restrito
- $\langle y, z \rangle C \langle w, x \rangle$ pode ser provado mostrando-se que
 - $\langle y, z \rangle C = \langle w, x \rangle$
 - e
 - $\langle w, x \rangle \not\subseteq \langle y, z \rangle$

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 23

Relacionamento possível entre Classes: $\langle w,x \rangle = \langle y,z \rangle$

- $\langle w, x \rangle = \langle y, z \rangle$ (onde = significa equivalência)
 - A classe de algoritmos (w,x) -restrito e a classe de algoritmos (y,z) -restrito são equivalentes
- Ou seja, um conjunto de tarefas é viável com a classe (w, x) -restrito se e somente se for viável com a classe (y, z) -restrito
- $\langle w, x \rangle = \langle y, z \rangle$ pode ser provado mostrando-se que
 - $\langle w, x \rangle C = \langle y, z \rangle$
 - e
 - $\langle y, z \rangle C = \langle w, x \rangle$

Rômulo Silva de Oliveira, DAS-UFSC, novembro/2009 24

Relacionamento possível entre Classes: $\langle w, x \rangle \otimes \langle y, z \rangle$

- $\langle w, x \rangle \otimes \langle y, z \rangle$ (onde \otimes significa incomparável)
 - A classe de algoritmos (w, x) -restrito e a classe de algoritmos (y, z) -restrito são incomparáveis
- Ou seja, existe ao menos um conjunto de tarefas que é viável com a classe (w, x) -restrito mas não é viável com a classe (y, z) -restrito, e vice versa
- $\langle w, x \rangle \otimes \langle y, z \rangle$ pode ser provado mostrando-se que
 - $\langle w, x \rangle \sim C = \langle y, z \rangle$
 - e
 - $\langle y, z \rangle \sim C = \langle w, x \rangle$

Relacionamentos possíveis entre Classes

- Quadro resumo

Notation	Semantically	Proof Obligation
$\langle w, x \rangle = \langle y, z \rangle$	(w, x) - and (y, z) -restricted classes are equivalent	$\langle w, x \rangle \subseteq \langle y, z \rangle \wedge \langle y, z \rangle \subseteq \langle w, x \rangle$
$\langle w, x \rangle \otimes \langle y, z \rangle$	(w, x) - and (y, z) -restricted classes are incomparable	$\langle w, x \rangle \not\subseteq \langle y, z \rangle \wedge \langle y, z \rangle \not\subseteq \langle w, x \rangle$
$\langle w, x \rangle \subset \langle y, z \rangle$	(y, z) -restricted class dominates (w, x) -restricted class	$\langle w, x \rangle \subseteq \langle y, z \rangle \wedge \langle y, z \rangle \not\subseteq \langle w, x \rangle$

Relacionamentos do (3,3)-restrito

- A classe de algoritmos (3,3)-restrito é a mais geral no sentido que qualquer conjunto de tarefas que é viável na classe (x,y)-restrito também será viável na classe (3,3)-restrito, para todo x e y
- Infelizmente, o custo da execução (runtime overhead) de um algoritmo do tipo (3,3)-restrito pode ser inaceitável para algumas aplicações, em termos de
 - Complexidade durante a execução
 - Frequência das preempções das tarefas
 - Frequência das migrações

Relacionamentos pelas Prioridades

- TEOREMA 1: Uma vez que
- todo algoritmo de prioridade fixa é, por definição, um algoritmo com prioridade dinâmica por job e
- Todo algoritmo de prioridade dinâmica por job é, por definição, um algoritmo com prioridade dinâmica irrestrita
 - $\langle 1, 1 \rangle \subseteq \langle 2, 1 \rangle \subseteq \langle 3, 1 \rangle$
 - $\langle 1, 2 \rangle \subseteq \langle 2, 2 \rangle \subseteq \langle 3, 2 \rangle$
 - $\langle 1, 3 \rangle \subseteq \langle 2, 3 \rangle \subseteq \langle 3, 3 \rangle$

Relacionamentos pelo EDF

- TEOREMA 2
- EDF é ótimo para monoprocessadores
 - É ótimo localmente para sistemas particionados
- A optimalidade do EDF (prioridade dinâmica por job) implica em:

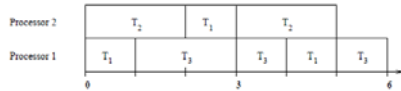
$$\langle 2, 1 \rangle = \langle 3, 1 \rangle$$

Relacionamentos do (1,1)-restrito

- TEOREMA 3
- A princípio pode-se imaginar que a classe dos algoritmos (1,1)-restrito é a classe menos geral de todas
- No sentido de que qualquer conjunto de tarefas viável com a classe (1,1)-restrito seria também viável com a classe (x, y)-restrito, para qualquer x e y
- Entretanto, Leung e Whitehead mostraram em 1982 que $\langle 1, 1 \rangle \otimes \langle 1, 3 \rangle$
- As classes (1, 1)-restrito e (1, 3)-restrito são incomparáveis

$A \in \langle 1, 2 \rangle, A \in \langle 2, 2 \rangle, A \in \langle 3, 2 \rangle$

- LEMA 1
- A: $T_1 = (1,2), T_2 = (2,3), T_3 = (2,3); M=2$
- $A \in \langle 1, 2 \rangle$, implica que
 - $A \in \langle 2, 2 \rangle$ e
 - $A \in \langle 3, 2 \rangle$
- Suponha T_2 tem prioridade mais alta que T_1 , que tem prioridade mais alta que T_3
 - A escala repete após o mínimo múltiplo comum dos períodos

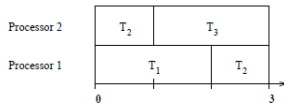


$A \notin \langle 1, 1 \rangle, A \notin \langle 2, 1 \rangle, A \notin \langle 3, 1 \rangle$

- LEMMA 2
- A: $T_1 = (1,2), T_2 = (2,3), T_3 = (2,3); M=2$
- $A \notin \langle 1, 1 \rangle$
- $A \notin \langle 2, 1 \rangle$
- $A \notin \langle 3, 1 \rangle$
- As tarefas não podem ser divididas em dois conjuntos de tal forma que cada conjunto tenha utilização menor ou igual a 1

$B \in \langle 3, 3 \rangle$

- LEMMA 3
- B: $T_1 = (2,3), T_2 = (2,3), T_3 = (2,3); M=2$
- $B \in \langle 3, 3 \rangle$



$(x \neq 3) \vee (y \neq 3) \rightarrow B \notin \langle x, y \rangle$

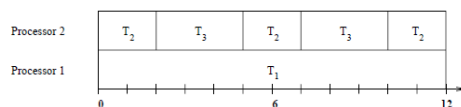
- LEMMA 4
- B: $T_1 = (2,3), T_2 = (2,3), T_3 = (2,3); M=2$
- $((x \neq 3) \vee (y \neq 3)) \rightarrow B \notin \langle x, y \rangle$
- Consider the first job of each task in B
- Se $(x \neq 3) \vee (y \neq 3)$
 - Então estes jobs não podem migrar ou suas prioridades relativas são fixas
- Todos os 3 jobs são liberados no instante 0 e precisam terminar até o instante 3
- Se os jobs não podem migrar
 - Dois jobs precisam executar completamente no mesmo processador, impossível
- Se as prioridades são fixas entre jobs
 - O job de mais baixa prioridade não pode começar a execução antes do instante 2 e vai perder seu deadline no instante 3
- Logo,
 - $B \notin \langle x, y \rangle$

$C \in \langle 2, 1 \rangle, C \in \langle 2, 2 \rangle, C \in \langle 2, 3 \rangle$

- LEMMA 5
- C: $T_1 = (12,12), T_2 = (2,4), T_3 = (3,6); M=2$
- $C \in \langle 2, 1 \rangle$
 - Implica que $C \in \langle 3, 1 \rangle$
- $C \in \langle 2, 2 \rangle$
 - Implica que $C \in \langle 3, 2 \rangle$
- $C \in \langle 2, 3 \rangle$
 - Implica que $C \in \langle 3, 3 \rangle$
- Este algoritmo escala corretamente C:
 - Todos os jobs de T_1 recebem prioridade mais alta
 - Jobs de T_2 e T_3 são escalonados conforme EDF
 - Este algoritmo usa prioridade dinâmica por Job

$C \in \langle 2, 1 \rangle, C \in \langle 2, 2 \rangle, C \in \langle 2, 3 \rangle$

- C: $T_1 = (12,12), T_2 = (2,4), T_3 = (3,6); M=2$
- Uma vez que a utilização de T_1 é 1, vai executar somente em um processador
- Este algoritmo vai produzir a mesma escala de tempo caso as tarefas fossem particionadas nos conjuntos $\{T_1\}$ e $\{T_2, T_3\}$
- Corretude segue da optimalidade de EDF para monoprocessadores
 - T_2 e T_3 podem ser corretamente escalonadas por EDF em um processador



$C \sim \epsilon \langle 1,1 \rangle, C \sim \epsilon \langle 1,2 \rangle, C \sim \epsilon \langle 1,3 \rangle$

- LEMMA 6
- C: T1 = (12,12), T2 = (2,4), T3 = (3,6); M=2
- $C \sim \epsilon \langle 1, 1 \rangle$
- $C \sim \epsilon \langle 1, 2 \rangle$
- $C \sim \epsilon \langle 1, 3 \rangle$
- Este conjunto de tarefas não é viável quando prioridades fixas são usadas pois nenhum esquema de prioridade fixa escalona {T2,T3} no mesmo processador
- Claramente T1 deve executar sozinha em um processador
- Não importa como T2 e T3 recebam prioridades fixas, a tarefa de menor prioridade vai perder o deadline

$D \in \langle 1,1 \rangle, D \in \langle 2,1 \rangle, D \in \langle 3,1 \rangle$

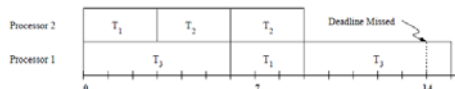
- LEMMA 7
- D: T1 = (3,6), T2 = (3,6), T3 = (6,7); M=2
- $D \in \langle 1,1 \rangle$
 - Implica em $D \in \langle 2,1 \rangle$
 - Implica em $D \in \langle 3,1 \rangle$
- Particione as tarefas tal que T1 e T2 sejam escalonadas em um único processador com T1 recebendo maior prioridade
- E T3 é escalonado no outro processador

$D \sim \epsilon \langle 3,2 \rangle, D \sim \epsilon \langle 2,2 \rangle, D \sim \epsilon \langle 1,2 \rangle$

- LEMMA 8
- D: T1 = (3,6), T2 = (3,6), T3 = (6,7); M=2
- $D \sim \epsilon \langle 3, 2 \rangle$
 - Implica que $D \sim \epsilon \langle 2, 2 \rangle$
 - Implica que $D \sim \epsilon \langle 1, 2 \rangle$
- Considere os três jobs liberados no intervalo [0, 6)
- Não importando como as prioridades são definidas, Se os deadlines são cumpridos então um algoritmo work-conserving vai completar os três em 6
- Neste instante, tarefas T1 e T2 liberam um novo job cada
- Qualquer algoritmo work-conserving deve agora escalonar o novo job de T1 em um processador e o novo job de T2 no outro processador

$D \sim \epsilon \langle 3,2 \rangle, D \sim \epsilon \langle 2,2 \rangle, D \sim \epsilon \langle 1,2 \rangle$

- LEMMA 8
- D: T1 = (3,6), T2 = (3,6), T3 = (6,7); M=2
- Dado que um algoritmo (3,2)-restrito não pode migrar jobs em execução, esses jobs precisam concluir no processador onde iniciaram
- Quando o segundo job de T3 é liberado no instante 7, ele não será escalonado até o instante 9, perdendo o deadline no instante 14

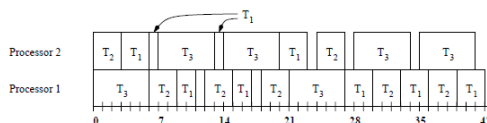


$D \in \langle 1,3 \rangle, D \in \langle 2,3 \rangle, D \in \langle 3,3 \rangle$

- LEMMA 9
- D: T1 = (3,6), T2 = (3,6), T3 = (6,7); M=2
- $D \in \langle 1, 3 \rangle$
 - Implica que $D \in \langle 2, 3 \rangle$
 - Implica que $D \in \langle 3, 3 \rangle$
- Considere um algoritmo que atribui para T3 a mais alta prioridade e para T1 a mais baixa prioridade
- Sobre o intervalo de tempo $[6k, 6k+6)$,
 - Onde k é um inteiro qualquer
 - T3 não pode executar mais do que 6 unidades de tempo

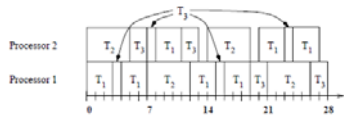
$D \in \langle 1,3 \rangle, D \in \langle 2,3 \rangle, D \in \langle 3,3 \rangle$

- LEMMA 9
- D: T1 = (3,6), T2 = (3,6), T3 = (6,7); M=2
- Uma vez que jobs podem migrar livremente entre processadores, existem 6 unidades de tempo consecutivas de processador disponível para T1 e T2 executarem sobre cada intervalo
- Logo, eles vão cumprir os seus deadlines



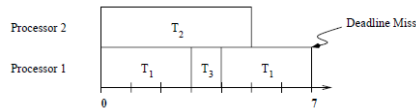
$E \in \langle 1,3 \rangle$

- LEMMA 10
- E: $T_1 = (3,4)$, $T_2 = (5,7)$, $T_3 = (3,7)$; $M=2$
- $E \in \langle 1, 3 \rangle$
 - Implica que $E \in \langle 2, 3 \rangle$
 - Implica que $E \in \langle 3, 3 \rangle$
- Se a ordem de prioridades for (alta para baixa) T_1, T_2, T_3 , então todos os jobs completam antes do deadline quando migração irrestrita é permitida



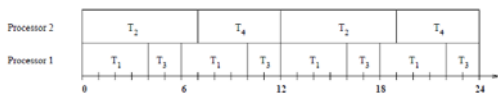
$E \notin \langle 1,2 \rangle$

- LEMMA 11
- E: $T_1 = (3,4)$, $T_2 = (5,7)$, $T_3 = (3,7)$; $M=2$
- $E \notin \langle 1, 2 \rangle$
- Apenas migração restrita é permitida
- Pode-se verificar que para cada uma das $3! = 6$ possíveis atribuições de prioridades fixas
- Alguma tarefa do conjunto E perde o seu deadline
- Exemplo mostra escala para ordem de prioridades T_1, T_2, T_3



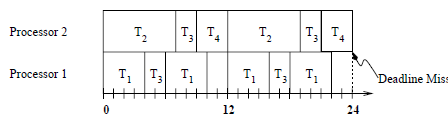
$F \in \langle 1,2 \rangle, F \in \langle 2,2 \rangle, F \in \langle 3,2 \rangle$

- LEMMA 12
- F: $T_1 = (4,6)$, $T_2 = (7,12)$, $T_3 = (4,12)$, $T_4 = (10,24)$; $M=2$
- $F \in \langle 1, 2 \rangle$
 - Implica que $F \in \langle 2, 2 \rangle$ e que $F \in \langle 3, 2 \rangle$
- Se a ordem de prioridades (alta para baixa) for T_1, T_2, T_3, T_4
 - Então todos os jobs cumprem o seu deadline
- Considerando os instantes $[7, 10)$, como jobs não podem migrar
 - T_3 não preempta T_4 neste intervalo apesar de sua prioridade mais alta



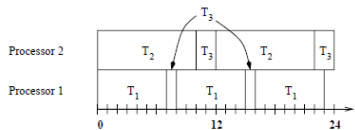
$F \notin \langle 1,3 \rangle$

- LEMMA 13
- F: $T_1 = (4,6)$, $T_2 = (7,12)$, $T_3 = (4,12)$, $T_4 = (10,24)$; $M=2$
- $F \notin \langle 1, 3 \rangle$
- Todas as $4! = 24$ possíveis atribuições de prioridade resultam em perda de algum deadline
- Exemplo mostra escala para ordem de prioridade T_1, T_2, T_3, T_4



$G \in \langle 1,3 \rangle, G \in \langle 2,3 \rangle, G \in \langle 3,3 \rangle$

- LEMMA 14
- G: $T_1 = (7,8)$, $T_2 = (10,12)$, $T_3 = (6,24)$; $M=2$
- $G \in \langle 1, 3 \rangle$
 - Implica que $G \in \langle 2, 3 \rangle$
 - Implica que $G \in \langle 3, 3 \rangle$
- Se a ordem de prioridade for (alta para baixa) T_1, T_2, T_3 , então todos os jobs cumprem os deadlines com migração irrestrita

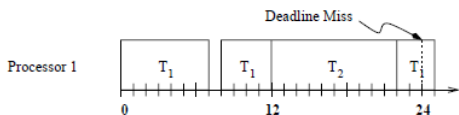


$G \notin \langle 3,2 \rangle, G \notin \langle 2,2 \rangle, G \notin \langle 1,2 \rangle$

- LEMMA 15
- G: $T_1 = (7,8)$, $T_2 = (10,12)$, $T_3 = (6,24)$; $M=2$
- $G \notin \langle 3, 2 \rangle$
 - Implica que $G \notin \langle 2, 2 \rangle$
 - Implica que $G \notin \langle 1, 2 \rangle$
- Durante o intervalo $[0, 24)$ as tarefas do conjunto G liberam 6 jobs com demandas de tempo 7, 7, 7, 10, 10, e 6
- Como jobs não podem migrar, para completar todos os jobs antes do instante 24, esses jobs precisam ser particionados em dois grupos tal que a soma dos tempos de execução em cada grupo não exceda 24
- A única partição que atende é $\{7, 7, 10\}$ e $\{6, 7, 10\}$

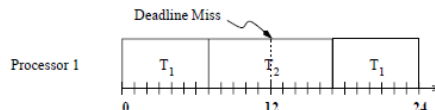
$G \sim C \langle 3,2 \rangle, G \sim C \langle 2,2 \rangle, G \sim C \langle 1,2 \rangle$

- LEMMA 15
- Considere processador executando {7, 7, 10}
- O job que demanda 10 executa em [0, 12) ou [12, 24)
- Se for em [12, 24), então apenas 7 unidades em [0, 8) podem ser utilizadas, pois os jobs com demanda 7 pertencem a tarefa T1
- O processador precisará ficar idle por 1 unidade de tempo
 - Um dos outros dois jobs perde o deadline



$G \sim C \langle 3,2 \rangle, G \sim C \langle 2,2 \rangle, G \sim C \langle 1,2 \rangle$

- LEMMA 15
- Considere processador executando {7, 7, 10}
- O job que demanda 10 executa em [0, 12) ou [12, 24)
- Se for em [0, 12), então um job com demanda de 7 deve executar em [0, 8) ou [8, 16)
- Não importa qual, a demanda sobre [0, 16) é 17
 - Um deadline será perdido no instante de tempo 12 ou 16

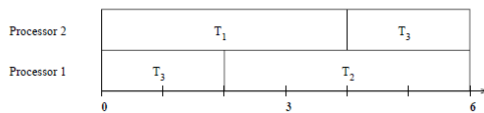


$H \sim C \langle 3,1 \rangle, H \sim C \langle 2,1 \rangle, H \sim C \langle 1,1 \rangle$

- LEMMA 16
- H: T1 = (4,6), T2 = (4,6), T3 = (2,3); M=2
- $H \sim C \langle 3, 1 \rangle$
 - Implica que $H \sim C \langle 2, 1 \rangle$
 - Implica que $H \sim C \langle 1, 1 \rangle$
- H não pode ser particionado em dois conjuntos tal que cada conjunto tenha utilização menor ou igual a 1

$H \in \langle 3,2 \rangle$ e $H \in \langle 2,3 \rangle$

- LEMMA 17
- H: T1 = (4,6), T2 = (4,6), T3 = (2,3); M=2
- $H \in \langle 3, 2 \rangle$ e $H \in \langle 2, 3 \rangle$
- A escala abaixo mostra que $H \in \langle 3, 2 \rangle$
- Esta escala pode produzida pela seguinte ordem de prioridade:
 - T1's job, T3's first job, T2's job, T3's second job
 - Logo, $H \in \langle 2, 3 \rangle$

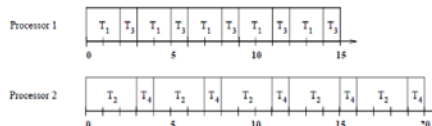


$I \sim C \langle 2,3 \rangle$

- LEMMA 18
- I: T1 = (2,3), T2 = (3,4), T3 = (5,15), T4 = (5,20); M=2
- $I \sim C \langle 2, 3 \rangle$
- No intervalo [0, 12) existem nove jobs liberados
 - E 9! Possíveis atribuições de prioridades
- Foi verificado por simulação que não importa as prioridades:
 - Um deadline é sempre perdido em [0, 12) ou
 - Um processador fica idle por pelo menos 1 unidade em [0, 12)
- Como a utilização total é igual ao número de processadores
 - Processador idle 1 unidade significa que um deadline será perdido em algum instante após o instante 12

$I \in \langle 1,1 \rangle, I \in \langle 2,1 \rangle, I \in \langle 3,1 \rangle$

- LEMMA 19
- I: T1 = (2,3), T2 = (3,4), T3 = (5,15), T4 = (5,20); M=2
- $I \in \langle 1, 1 \rangle$
 - Implica que $I \in \langle 2, 1 \rangle$ e $I \in \langle 3, 1 \rangle$
- O conjunto de tarefas I pode ser escalonado com o particionamento: {T1, T3} e {T2, T4}
- Cada sub-conjunto é escalonável com RM em um processador



$I \in \langle 3, 2 \rangle$

- LEMMA 20
- I: $T1 = (2,3)$, $T2 = (3,4)$, $T3 = (5,15)$, $T4 = (5,20)$; $M=2$
- $I \in \langle 3, 2 \rangle$
- Considere o escalonamento mostrado no Lema 19 (anterior)
- Tal escala pode ser obtida com um algoritmo da classe $\langle 3, 2 \rangle$
- Basta atribuir a prioridade mais alta aos jobs apropriados a cada instante
- Isto é possível pois não existe tempo idle naquela escala

Comparações entre Classes de Escalonamento

- Usando os lemas é possível derivar muitos relacionamentos entre as classes de escalonamento
- Com prioridades fixas todas as três classes de migração são incomparáveis
- Existem ainda dois relacionamentos em aberto

Comparações entre Classes de Escalonamento

	(1,1)	(2,1)	(3,1)	(1,2)	(2,2)	(3,2)	(1,3)	(2,3)	(3,3)
(1,1)	=	<	<	⊗	⊗	⊗	⊗	⊗	<
(2,1)	>	=	=	⊗	⊗	⊗	⊗	⊗	<
(3,1)	>	=	=	⊗	⊗	⊗	⊗	⊗	<
(1,2)	⊗	⊗	⊗	=	<	<	⊗	??	<
(2,2)	⊗	⊗	⊗	>	=	⊆	⊗	??	<
(3,2)	⊗	⊗	⊗	>	⊇	=	⊗	⊗	<
(1,3)	⊗	⊗	⊗	⊗	⊗	⊗	=	<	<
(2,3)	⊗	⊗	⊗	??	??	⊗	>	=	<
(3,3)	>	>	>	>	>	>	>	>	=

Análise de Viabilidade com determinada Classe

- Um teste de viabilidade da classe (x, y) -restrito aceita como entrada as descrições de τ e M
- E determina se algum algoritmo (x, y) -restrito pode escalonar com sucesso as tarefas de τ em M processadores
- Tal teste é suficiente se qualquer conjunto de tarefas satisfazendo o teste pode ser garantidamente escalonado com sucesso por algum algoritmo (x, y) -restrito
- O teste é exato se for ambos suficiente e necessário
 - Ou seja, nenhum conjunto de tarefas falhando no teste pode ser escalonado por qualquer algoritmo (x, y) -restrito
- Testes de viabilidade são normalmente enunciados como limites de utilização
- Tal limite é o maior valor U_{max} tal que todo conjunto de tarefas τ satisfazendo $U(\tau) \leq U_{max}$ é garantido ser viável

Teste de Viabilidade para (3,3)-restrito

- TEOREMA 4
- Um conjunto de tarefas periódicas τ pode ser escalonado em M processadores usando algum algoritmo $(3, 3)$ -restrito se e somente se $U(\tau) \leq M$
 - Global LLF
 - Abordagem de escalonamento Pfair
- Não existe perda de escalonabilidade (usa todos os processadores)
- Implementações eficientes são conhecidas
- Mas o número de preempções e migrações entre processadores pode ser alto

Classes com Particionamento

- Existe uma quantidade considerável de pesquisa sobre algoritmos de escalonamento $(x, 1)$ -restrito
- Achar uma alocação ótima de tarefas em processadores é equivalente a um problema “bin-packing”
 - Sabidamente NP-hard no sentido forte
- Muitas heurísticas com complexidade polinomial foram propostas para resolver este problema
- First Fit (FF), Best Fit (BF), etc

Classes com Particionamento

- TEOREMA 5
- Nenhum algoritmo de escalonamento baseado em particionamento pode escalonar com sucesso todos os conjuntos de tarefas τ com $U(\tau) \leq B$ em M processadores, onde $B > (M+1)/2$
- A utilização alcançável no pior caso em M processadores para todas as heurísticas e também para um algoritmo de particionamento ótimo é apenas $(M+1)/2$
- Mesmo quando um algoritmo de escalonamento ótimo para monoprocessador é usado localmente (ex: EDF)
- Em outras palavras, Existem conjuntos de tarefas com utilização minimalmente maior que $(M+1)/2$ tais que é impossível escaloná-los corretamente com uma abordagem particionada
- Suponha um conjunto com $M+1$ tarefas cada uma com demanda de $1+e$ e período 2
 - É impossível particionar com sucesso em M processadores

Classes com Particionamento: FF ou BF c/EDF

- TEOREMA 6
- Se $U(\tau) \leq (\beta M + 1) / (\beta + 1)$
onde $\beta = \lfloor 1/\alpha \rfloor$, $\alpha \geq U(T)$ para toda $T \in \tau$
então τ é viável em M processadores na classe $(2, 1)$ -restrito
- Lopez et al. Mostraram que EDF com FF (ou BF) pode escalonar com sucesso qualquer conjunto de tarefas que passe neste teste
- Corolário:
- Se $U(\tau) \leq (M+1)/2$
 - então τ é viável em M processadores com a classe $(2, 1)$ -restrito
- Segue do teorema 6 fazendo $\alpha = 1$ (e então $\beta = 1$)
- Teorema 5 e este Corolário implicam que usar EDF com FF ou BF é uma ótima abordagem particionada com respeito aos limites de utilização

Classes com Particionamento: FF ou BF c/RM

- TEOREMA 7
- A utilização possível no pior caso é muito menor para conjuntos de tarefas escalonados com RM
 - RM não é ótimo para monoprocessadores
- $U_{RM,FF}$ denota a utilização possível no pior caso usando RM com FF
- Oh e Baker em 1998 mostraram os seguintes limites para $U_{RM,FF}$

$$M(2^{1/2} - 1) < U_{RM,FF} \leq \frac{M+1}{1+2^{M+1}}, M \geq 2$$

- Conjuntos de tarefas nos quais a utilização total não exceda $\approx 0.41 \times M$ são escalonáveis com RM-FF
- Embora este valor seja pequeno, RM é popular devido a sua simplicidade e previsibilidade na sobrecarga

Classes com Particionamento: FFDU c/RM

- Muitos pesquisadores propuseram heurísticas para particionamento que melhoram os resultados de FF e BF
- Oh e Son 1995 propuseram uma variação melhorada da heurística FF chamada First Fit Decreasing Utilization (FFDU)
- Eles mostram que para conjuntos de tarefas escalonados com RM, o número de processadores necessários por FFDU é no máximo $5/3$ do número ótimo de processadores
- Dhall e Liu mostraram antes que o número de processadores necessários por FF e BF é no máximo 2 vezes o número ótimo

Limite de Utilização para (x,y)-restrito, exceto (3,3)

- TEOREMA 8
- Com exceção de $x=3$ e $y=3$ nenhum algoritmo (x,y) -restrito pode escalonar todos os conjuntos de tarefas τ com $U(\tau) \leq B$ em M processadores, onde $B > (M+1)/2$
- Considere um sistema com $M+1$ tarefas em M processadores
 - Cada uma com um período de 2 e uma computação de $(1+e)$
- Considere o conjunto de jobs liberados por cada tarefa no instante 0
- Se migração de jobs em execução não for permitida, no intervalo $[0, 2)$, dois desses jobs precisam executar no mesmo processador, o que implica na perda do deadline de um deles
- Este conjunto de tarefas não pode ser escalonado por um algoritmo $(1,3)$ - ou $(2,3)$ -restrito por que quando os $M+1$ jobs são liberados no instante 0, o job de mais baixa prioridade vai perder seu deadline
- Quando e tende para zero, a utilização tende para $(M+1)/2$

Overhead dos Algoritmos (2,3)-restrito

- Algoritmos $(2,3)$ -restrito
 - Esses algoritmos associam uma prioridade fixa com cada job mas permitem que jobs migrem entre processadores arbitrariamente
 - Ex: escalonamento global com EDF
- Uma preempção (e portanto uma migração) somente pode ser causada pela liberação de um job ou outra migração
- Logo, o número total de preempções e migrações pode ser limitado pelo número amortizado de uma por job
 - Se o algoritmo de escalonamento for projetado adequadamente
- Existe overhead devido a preempção e migração
 - Mas este custo pode ser limitado
 - E pode ser aceitável para algumas aplicações

Limite de Utilização para (2,3)-restrito

- TEOREMA 9
- Um conjunto de tarefas é viável com a classe (2,3)-restrito se $U(\tau) \leq M^2/(2M-1)$
- Srinivasan e Baruah 2002 apresentaram um algoritmo (2, 3)-restrito baseado em EDF
- Tarefas com utilização maior ou igual a $M/(2M-1)$ recebem estaticamente a prioridade mais alta no sistema
- As demais tarefas recebem prioridade conforme EDF
- Eles provaram que com este algoritmo pode-se escalonar todos os conjuntos de tarefas τ com $U(\tau) \leq M^2/(2M-1)$

Limite de Utilização para (1,3)-restrito

- TEOREMA 10
- Um conjunto de tarefas τ é viável com a classe (1,3)-restrito se $U(\tau) \leq M^2/(3M-2)$
- Andersson et al. 2001 desenvolveram um algoritmo similar ao do teorema anterior, mas baseado em RM no lugar de EDF
- Tarefas com utilizações maiores ou iguais a $M/(3M-2)$ recebem estaticamente a mais alta prioridade do sistema
- As demais tarefas recebem prioridade conforme o RM
- Eles provaram que estão garantidos os deadlines de todos os conjuntos de tarefas τ com $U(\tau) \leq M^2/(3M-2)$

Limite de Utilização Harmônica para (1,3)-restrito

- TEOREMA 11
- Um conjunto de tarefas τ , no qual todos os períodos das tarefas são harmônicos entre si, é viável com a classe (1,3)-restrito se $U(\tau) \leq M^2/(2M-1)$
- Pode-se mostrar que se os períodos das tarefas forem harmônicos entre si, a escala de tempo produzida pelo RM será a mesma escala que o EDF produziria
- Logo, o mesmo limite pode ser usado

Limite de Utilização para (2,2)-restrito

- TEOREMA 12
- Se $U(\tau) \leq M - \alpha(M-1)$, onde α satisfaz $\alpha \geq U(T)$ para todo $T \in \tau$ então τ é viável em M processadores com na classe (2,2)-restrito
- Algoritmos da classe (2,2)-restrito
 - associam uma prioridade fixa com cada job
 - e restringem cada job para executar exclusivamente em um único processador
- Entretanto, diferentes jobs da mesma tarefa podem executar em diferentes processadores
- Baruah and Carpenter 2003 projetaram um algoritmo (2,2)-restrito que escala com sucesso qualquer conjunto de tarefas periódicas τ que satisfaça $U(\tau) \leq M - \alpha(M-1)$ onde α satisfaz $\alpha \geq U(T)$ para todo $T \in \tau$

Resumo dos Limites de Utilização

3: full migration	$\frac{M^2}{2M-1} \leq U \leq \frac{M+1}{2}$	$\frac{M^2}{2M-1} \leq U \leq \frac{M+1}{2}$	$U = M$
2: restricted migration	$U \leq \frac{M+1}{2}$	$M - \alpha(M-1) \leq U \leq \frac{M+1}{2}$	$M - \alpha(M-1) \leq U \leq \frac{M+1}{2}$
1: partitioned	$(\sqrt{2}-1)M \leq U \leq \frac{M+1}{1+\sqrt{2M+1}}$	$U = \frac{M+1}{2}$	$U = \frac{M+1}{2}$
1: static		2: job-level dynamic	3: unrestricted dynamic

Resumo dos Limites de Utilização

- O limite de utilização exato para algoritmos (3, 3)-restrito seguem do teorema 4
- Os limites para a utilização possível no pior caso para sistemas particionados seguem dos teoremas 5, 6 e 7
- Os limites para algoritmos (3, 1)-restrito seguem daqueles limites para algoritmos (2, 1)-restrito pois $\langle 3, 1 \rangle = \langle 2, 1 \rangle$
- Os limites superiores para as demais classes seguem do teorema 8
- Os limites inferiores para a utilização possível no pior caso para algoritmos (1,3)-restrito, (2,3)-restrito e (2,2)-restrito seguem dos teoremas 10, 9 e 12, respectivamente
- O limite inferior para algoritmos (3,2)-restrito seguem porque $\langle 2, 2 \rangle C = \langle 3, 2 \rangle$

Escalonamento Tempo Real em Multiprocessadores

- Não foi falado em
 - Recursos, exclusão mútua
 - Servidores de aperiódicas
 - Release Jitter
 - Deadline menor que o período
 - Deadline maior que o período
 - Relações de precedência
 - Efeito da preempção nas caches
 - Efeito da migração nas caches
 - Tarefas esporádicas
- Vários resultados novos depois de 2003
- Muita coisa ainda por fazer