Universidade Federal da Bahia
Escola Politécnica e do Instituto de Matemática
Pós-Graduação em Mecatrônica

JOSÉ AUGUSTO MATOS SANTOS JÚNIOR

ESCALONAMENTO EM SISTEMAS DE TEMPO REAL MULTIPROCESSADOS COM BAIXO CUSTO DE IMPLEMENTAÇÃO

DISSERTAÇÃO DE MESTRADO

Salvador
2012

Universidade Federal da Bahia
Escola Politécnica e do Instituto de Matemática

José Augusto Matos Santos Júnior

ESCALONAMENTO EM SISTEMAS DE TEMPO REAL MULTIPROCESSADOS COM BAIXO CUSTO DE IMPLEMENTAÇÃO

Trabalho apresentado ao Programa de Pós-Graduação em Mecatrônica da Escola Politécnica e do Instituto de Matemática da Universidade Federal da Bahia como requisito parcial para obtenção do grau de Mestre em Mecatrônica.

Orientador: *Prof. Dr. George Marconi de Araújo Lima*

Salvador
2012

Sistemas de Bibliotecas - UFBA

Santos Júnior, José Augusto Matos.
Escalonamento em sistemas de tempo real multiprocessados com baixo custo de implementação / José Augusto Matos Santos Júnior. - 2012.
79 f. : il.

Orientador: Prof. Dr. George Marconi de Araújo Lima.
Dissertação (mestrado) - Universidade Federal da Bahia, Instituto de Matemática e Escola Politécnica, Salvador, 2012.

1. Processamento eletrônico de dados em tempo real. 2. Multiprocessadores. 3. Otimização matemática - Processamento de dados. 4. Algoritmos. I. Lima, George Marconi de Araújo. II. Universidade Federal da Bahia. Instituto de Matemática. III. Universidade Federal da Bahia. Escola Politécnica. IV. Título.

CDD - 004.33
CDU - 004.415.2.031.43

TERMO DE APROVAÇÃO

JOSÉ AUGUSTO MATOS SANTOS JÚNIOR

ESCALONAMENTO EM SISTEMAS DE TEMPO REAL MULTIPROCESSADOS COM BAIXO CUSTO DE IMPLEMENTAÇÃO

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em Mecatrônica, Universidade Federal da Bahia - UFBA, pela seguinte banca examinadora:

Prof. George Marconi de Araújo Lima (Orientador)
Doutor em Ciência da Computação, University of York, Inglaterra
Professor da Universidade Federal da Bahia

Prof. Eduardo Camponogara (Examinador Externo)
Doutor em Engenharia Elétrica e de Computação, Carnegie Mellon University, EUA
Professor da Universidade Federal de Santa Catarina

Prof. Rômulo Silva de Oliveira (Examinador Externo)
Doutor em Engenharia Elétrica, Universidade Federal de Santa Catarina, Brasil
Professor da Universidade Federal de Santa Catarina

Salvador, 15 de março de 2012.

A Deus, por permitir minha existência.
A minha família, pelo apoio e amor.

Nada cuesta más trabajo que vivir sin trabajar.

—DON RAMÓN

AGRADECIMENTOS

Agradeço aos meus pais, José Augusto Matos Santos (*in memoriam*) e Elenilda Maria Santos Matos, e ao meu irmão, Gustavo Santos Matos, pelo amor e apoio incondicionais. Agradeço ao meu orientador, George Marconi de Araújo Lima, pela paciência na orientação, disponibilidade e incentivo ao longo do desenvolvimento deste trabalho. Ao LaSiD, por me acolher desde minha graduação até a conclusão deste projeto. Aos meus amigos e familiares por entenderem a minha constante ausência e por me incentivarem ao longo desta jornada. Por fim, a FAPESB, pelo financiamento de minha bolsa de mestrado.

RESUMO

Atualmente, muitos sistemas mecatrônicos apresentam comportamentos definidos segundo restrições temporais e são comumente identificados como Sistemas de Tempo Real (STR). Estes sistemas são encontrados em diversas áreas que envolvem tecnologia, como automação industrial, telecomunicações e sistemas espaciais. Em todas essas áreas, há um rápido progresso tecnológico, que contribuí com o aumento na complexidade do software e na demanda de processamento.

Uma tendência crescente de utilização de plataformas com múltiplas unidades de processamento vem ocorrendo nos últimos anos. Chips contendo 100 núcleos de processamento são agora uma realidade. Neste contexto, o problema de escalonamento de tarefas deve ser avaliado levando em consideração esta tendência de paralelismo, pois a garantia das restrições temporais dos STR depende de como suas tarefas são escalonadas.

O problema de escalonar n tarefas esporádicas num sistema de tempo real executando numa plataforma composta de m processadores idênticos é abordado nesta dissertação. A solução proposta é nomeada HIME (*HIghest-priority Migration managed by EDF*), a qual possui várias características interessantes: a maioria das tarefas executadas num único processador; existem no máximo $\lfloor 0{,}5m \rfloor$ tarefas migratórias; tanto os custos de utilização e a complexidade de sua implementação são baixos. O desempenho de HIME foi avaliado analiticamente e por simulação, os quais constataram que qualquer conjunto de tarefas com utilização não superior a $72{,}\bar{2}\%$ do sistema cumpre seus requisitos temporais e que HIME pode lidar com sistemas que possuem utilização maior que 95% da capacidade de processamento do sistema. Tais resultados vêm contribuir com a área de STR multiprocessados ao mesmo tempo em que indicam possíveis ramos de investigação.

Palavras-chave: Tempo Real, Sistemas Multiprocessados, Escalonamento

ABSTRACT

Currently, many mechatronic systems exhibit behaviors defined by time constraints and are commonly identified as Real-Time Systems (RTS). These systems are found in various fields involving technology, such as industrial automation, telecommunications and space systems. In all these areas, there is rapid technological progress, which contributed to the increased complexity of software and processing demand.

A growing trend of using platforms with multiple processing units has occurred in recent years. Chips containing 100 cores are now a reality. In this context, the problem of scheduling tasks should be assessed taking into account the trend of parallelism, because the guarantee of the time constraints of the RTS depends on how tasks are scheduled.

The problem of scheduling a real-time system composed of a set of n independent, preemptive sporadic tasks in a platform composed of m identical processors is discussed in this dissertation. The proposed solution is named HIME (HIghest-priority Migration managed by EDF), which has several interesting features: most tasks execute on a single processor; there are at most $\lfloor 0{,}5m \rfloor$ migrating tasks; both the overhead and complexity of its implementation are low. The performance of HIME was evaluated analytically and by simulation, which confirmed that any set of tasks using no more than $72{,}\bar{2}\%$ of the system meets its timing requirements, and that HIME can handle systems using more than 95% of the system processing capacity. These results are contributing to the area of multiprocessor RTS at the same time indicate possible lines of investigation.

Keywords: Real-Time, Multiprocessor System, Scheduling

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

DP	*Decreasing Periods*
DU	*Decreasing Utilization*
EDF	*Earliest Deadline First*
FF	*First-Fit*
HIME	*HIghest-priority Migration managed by EDF*
NF	*Next-Fit*
NPS-F	*Notional Processor Scheduling-Fractional capacity*
RM	*Rate Monotonic*
SPA2	*Semi-Partition Algorithm 2*
STR	Sistema de Tempo Real
WF	*Worst-Fit*

LISTA DE SÍMBOLOS

Γ	Conjunto de tarefas
α	Fator que representa o número de tarefas dividido pelo número de processadores
τ	Tarefa
C	Tempo de computação
Γ_j	Subconjunto de Γ alocado a um processador
m	Número de processadores
n	Número de tarefas
$\Omega(\tau)$	O conjunto de todas as tarefas alocadas no mesmo processador que τ
$\mho(\tau)$	$\Omega(\tau)\backslash\{\tau\}$
$\sigma(\Gamma, T)$	Máxima capacidade de processamento disponível em Γ para executar uma tarefa de mais alta prioridade de período T
T	Intervalo de tempo mínimo entre a ativação de duas instancias consecutiva de uma mesma tarefa
τ^i	A *i-ésima* porção de um tarefa migratória τ alocado em algum processador
U(τ)	Utilização da tarefa τ
U(Γ)	Utilização do conjunto de tarefas Γ

CAPÍTULO 1

INTRODUÇÃO

Neste capítulo é descrito uma introdução ao presente trabalho. Na Seção 1.1, aborda-se principalmente o que são, como funcionam e a utilidade dos sistemas de tempo real. A motivação e a proposta deste projeto são apresentadas na Seção 1.2, enquanto que os trabalhos relacionados e as contribuições geradas são apresentados na Seção 1.3. E, finalmente, a Seção 1.4 informa a estrutura dos demais capítulos deste documento.

1.1 SISTEMAS MECATRÔNICOS DE TEMPO REAL

Os sistemas mecatrônicos geralmente integram componentes eletromecânicos e controle computacional para a realização de uma determinada atividade. Atualmente, há uma grande quantidade destes sistemas que apresentam comportamentos definidos segundo restrições temporais. Alguns exemplos desses sistemas estão presentes no controle de tráfego aéreo ou ferroviário, nas telecomunicações, na eletrônica embarcada, em equipamentos médicos, em sistemas de multimídia, etc. Estes sistemas devem reagir a estímulos oriundos do ambiente em prazos específicos e são comumente identificados como Sistemas de Tempo Real (STR) (FARINES; FRAGA; OLIVEIRA, 2000). O comportamento correto de um STR, portanto, depende da integridade dos resultados obtidos (correção lógica) e do tempo em que são produzidos (correção temporal). A garantia das restrições temporais destes sistemas está estritamente relacionada como suas ações, comumente chamadas de tarefas, são escalonadas. Uma ação que ocorra além do prazo especificado pode ser sem utilidade ou até comprometer a integridade do sistema. Portanto, o escalonador, mecanismo responsável pela gestão da execução das tarefas, possui papel central na garantia da correção temporal destes sistemas.

Para efeito de especificação do problema de escalonamento e de derivação de suas

soluções, um STR é geralmente modelado como um conjunto de tarefas $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. Tarefas são unidades de processamento, ou blocos de ações sequenciais, que usam um ou mais recursos computacionais (BERNSTEIN; HADZILACOS; GOODMAN, 1987). Geralmente, as tarefas consistem de uma sequência infinita de invocações idênticas, suas instâncias ou *jobs*, que são ativadas em momentos distintos. As tarefas pertencentes ao conjunto Γ e suas instâncias são especificadas por atributos temporais, tais como:

- Tempo de chegada: este é o tempo no qual uma instância torna-se pronta para ser executada pelo sistema. Diferentes instâncias de uma mesma tarefa possuem diferentes tempos de chegada;
- Tempo de computação: é o tempo máximo necessário para executar uma tarefa num processador dedicado. Instâncias de uma mesma tarefa geralmente possuem o mesmo tempo de computação;
- *Deadline* relativo: é o intervalo de tempo máximo que uma instância tem para executar. Instâncias de uma mesma tarefa comumente possuem o mesmo *deadline* relativo;
- *Deadline* ou *deadline* absoluto: é o tempo antes do qual uma determinada instância deve completar sua execução para ser considerada correta. A soma do tempo de chegada com o *deadline* relativo é igual ao *deadline* absoluto;
- Tempo de resposta: é o intervalo de tempo que uma instância gasta para finalizar sua computação;

A partir do tipo de recorrência das tarefas, estas podem ser classificadas como periódicas, esporádicas ou aperiódicas. Diz-se que uma tarefa é periódica quando cada uma de suas instâncias é ativada em intervalos de tempo fixos e regulares. Uma tarefa é dita esporádica quando se conhece apenas o intervalo mínimo de tempo entre ativações consecutivas de suas instâncias. Isto significa que não se sabe exatamente quando cada instância será ativada no sistema, mas é possível assumir que a próxima não será ativada antes deste intervalo mínimo. São denominadas como aperiódicas as tarefas cujas

frequências de ativação de suas instâncias é desconhecida ou inexistente. Um sistema composto apenas de tarefas periódicas ou esporádicas é geralmente mais fácil de ter sua correção avaliada

Dependendo da criticidade de suas tarefas, os STR podem ser classificados em críticos (*Hard Real Time Systems*) e não-críticos (*Soft Real Time Systems*). Um STR crítico precisa ter as respostas de suas tarefas fornecidas impreterivelmente até seus respectivos *deadlines*. Caso contrário, as respostas são inúteis e podem até prejudicar o sistema como um todo. Uma falha num STR crítico poderia ter como resultados graves acidentes ou perdas catastróficas. Exemplos de tais sistemas podem ser encontrados em controle industrial e de aviação. Perder o *deadline* da verificação de altitude de um avião, por exemplo, pode provocar uma catástrofe caso o avião esteja aterrissando. No entanto, o não cumprimento ocasional de *deadlines* é permitido para os sistemas de tempo real não-críticos, apesar de não ser desejável. As respostas recebidas após este prazo podem ser parcialmente aproveitadas e as consequências deste atraso, geralmente, não são tão graves. Por exemplo, num sistema multimídia, deixar de exibir alguns quadros de imagem é tolerável.

Um dos equívocos mais comuns sobre os STR é a idéia que um computador suficientemente rápido é possível satisfazer os requisitos temporais, consequentemente, computação em tempo real é igual à computação rápida (STANKOVIC et al., 1998). O ponto é que a velocidade ajuda um STR a alcançar a necessária responsividade (*responsiveness*), mas em geral não dá suporte ao principal objetivo dos sistemas de tempo real que é a previsibilidade (STANKOVIC et al., 1998). É interessante ressaltar que não há vantagem em antecipar a conclusão de uma tarefa de tempo real crítica. Desde que esta tarefa complete sua execução num instante anterior ou igual ao seu *deadline*, seu tempo de resposta não é importante.

1.1.1 Mecanismo de Escalonamento

Como mencionado, o escalonador é um importante mecanismo, responsável por garantir a correção temporal num STR, seja ele crítico ou não. De maneira simples, o escalonador deve escolher, baseado em alguma política de escalonamento, qual tarefa deve ser executada em função do tempo, ou seja, o escalonador é o componente do sistema responsável, em tempo de execução, pela gestão do processador (FARINES; FRAGA; OLIVEIRA, 2000).

O termo escalonamento identifica o procedimento de ordenar tarefas na fila de tarefas prontas para executar (fila de prontos ou *ready queue*) (FARINES; FRAGA; OLIVEIRA, 2000). A regra de ordenação destas tarefas é definida pela política de escalonamento utilizada. Os escalonadores utilizam então estas políticas para produzirem uma escala de execução, a qual é uma função que associa uma tarefa em Γ a um determinado processador ao longo do tempo. Uma escala é dita válida se em qualquer instante de tempo um processador executa no máximo uma instância de uma tarefa e uma mesma instância não executa em mais do que um processador concomitantemente sendo que nenhuma instância é executada mais de uma vez e antes de ser liberada. Uma escala válida é correta se todas as tarefas do conjunto Γ são escalonadas de tal forma que não ocorra perda de *deadline*. Caso exista tal escala para um determinado sistema este é dito escalonável.

Escalonamento de tarefas em STR é um problema de otimização, onde se deseja minimizar o atraso das tarefas sujeito às restrições impostas pelo sistema, tais como relações de precedência entre tarefas, disponibilidade de recursos computacionais, etc. Entende-se por atraso a finalização de alguma instância de uma tarefa após seu *deadline* absoluto. Resolver o problema de escalonamento em sua forma geral não é viável, visto que o mesmo é NP-difícil (GAREY; JOHNSON, 1979). No entanto, há heurísticas eficientes para versões restritas do problema (BUTTAZZO, 1997; BARUAH; GOOSSENS, 2004).

Um exemplo clássico de política de escalonamento, na literatura de tempo real, é o *Earliest Deadline First* (EDF) (LIU; LAYLAND, 1973), na qual a tarefa mais prioritária é aquela que possui o *deadline* absoluto mais próximo a expirar. Uma das razões para

sua popularidade é que o EDF é uma política ótima para sistemas monoprocessados (LIU; LAYLAND, 1973; BARUAH; MOK; ROSIER, 1990), ou seja, caso haja alguma ordem de execução em que não haja violação de *deadlines*, a ordem gerada pelo EDF também cumprirá todos os *deadlines*. É importante notar que, como o problema de escalonamento é NP-difícil, ser ótimo está relacionado apenas a versões restritas do problema, por exemplo, o EDF é ótimo em sistemas monoprocessados com certas restrições (LIU; LAYLAND, 1973; SHA et al., 2004).

O problema de garantir que os requisitos temporais sejam satisfeitos tem sido amplamente estudado; políticas de escalonamento eficientes que consideram ambientes computacionais realistas são bem compreendidos nos sistemas de tempo real monoprocessados. No entanto, poucos dos resultados obtidos podem ser generalizados diretamente para o caso de sistemas multiprocessados. Introduzir unidades de processamento adicionais aumenta a complexidade do problema de escalonamento (DAVIS; BURNS, 2011).

Em sistemas multiprocessados, mesmo supondo que as tarefas podem executar em qualquer um dos processadores, existe uma restrição adicional. Em qualquer instante de tempo uma mesma tarefa não pode executar em mais que um processador. No entanto, uma tarefa pode iniciar sua execução numa unidade de processamento e migrar para outras durante sua execução. Políticas tradicionais de escalonamento, tal como o EDF, não aproveitam de modo adequado os recursos destes tipos de sistemas, devido à existência de anomalias observadas para estas políticas quando aplicadas em sistemas com arquitetura multiprocessada (DHALL; LIU, 1978; ANDERSSON; JONSSON, 2002).

1.1.2 Teste de Escalonabilidade

Um aspecto importante para o funcionamento adequado dos STR é verificar se a política de escalonamento empregada para um conjunto de tarefas Γ produz um escalonamento correto. Um meio para se constatar esta característica é o teste de escalonabilidade. Geralmente, usam-se funções matemáticas que exprimem o comportamento temporal do sistema no pior caso e que se baseiam no conhecimento sobre os parâmetros

das tarefas que compõem o sistema (MACÊDO et al., 2004). Estes testes normalmente são realizados em tempo de projeto e podem ser classificados como (FARINES; FRAGA; OLIVEIRA, 2000):

- Exatos: são testes que não descartam conjuntos de tarefas que apresentam escalas corretas e são precisos em identificar conjuntos não escalonáveis.
- Suficientes: são testes geralmente mais restritivos onde conjuntos de tarefas aceitos nesses testes certamente possuem escalonamento correto, mas entre os descartados podem existir conjuntos escalonáveis.
- Necessários: a única garantia que esse tipo de teste oferece é que os conjuntos de tarefas descartados certamente não são escalonáveis.

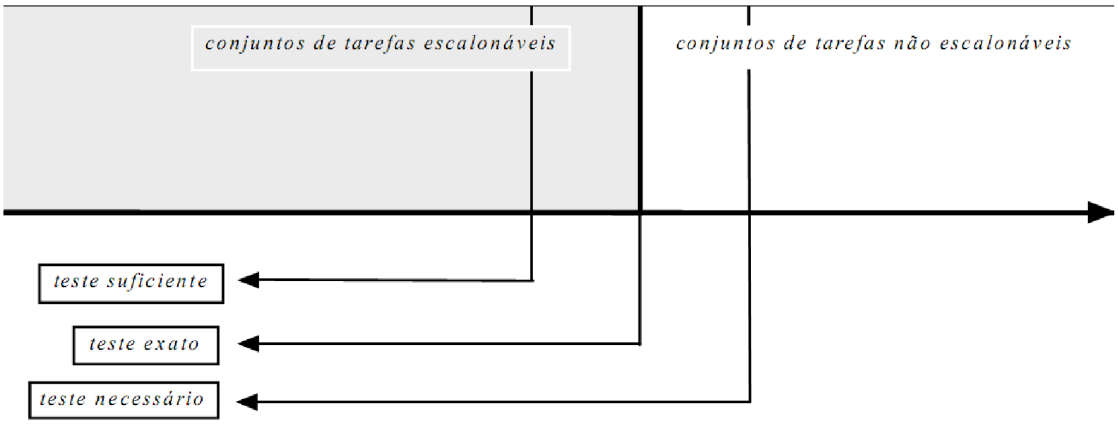


Figura 1.1. Tipos de testes de escalonabilidade (FARINES; FRAGA; OLIVEIRA, 2000)

A Figura 1.1 ilustra os tipos de testes descritos acima. É importante ressaltar que métodos baseados em simulação não podem responder de forma satisfatória se para a política de escalonamento empregada o conjunto Γ é escalonável devido ao alto número de possibilidades de escalonamento (MACÊDO et al., 2004). Embora os testes de escalonabilidade possam verificar se para uma determinada política de escalonamento as instâncias de um conjunto de tarefas cumprem seus *deadlines*, é útil determinar uma condição de escalonabilidade que expresse no pior caso o comportamento de uma determinada política.

Esta questão é geralmente tratada como o menor limite superior (*least upper bound*) de utilização, o qual informa que um conjunto de tarefas cumpre seus requisitos temporais sempre que a utilização do sistema não seja superior a um determinado valor de utilização, por exemplo, um conjunto de tarefas é escalonável por EDF, num sistema monoprocessado com certas restrições, se sua utilização não for superior a 1 (LIU; LAYLAND, 1973).

1.2 MOTIVAÇÃO E PROPOSTA

Plataformas *multicore* e multiprocessadas tornaram-se bastante populares. Processadores equipados com múltiplos núcleos de processamento são comumente encontrados no mercado. Protótipos desenvolvidos pela Intel com 80 núcleos (HELD; BAUTISTA; KOEHL, 2006) e, mais recentemente, uma versão comercial com 50 núcleos já foi anunciada pela Intel (INTEL, 2011). Chips como TILE-Gx100 da Tilera, contendo 100 núcleos (TILERA, 2011), são agora uma realidade. Provavelmente esta tendência deve continuar durante algum tempo e de acordo com alguns especialistas até 2017 haverá chips que possuam até 4096 núcleos de processamento (MERRITT, 2008). Neste contexto, vários sistemas de suporte a aplicações, como compiladores, linguagens de programação e sistemas operacionais, tem que ser reestruturados e reprojetados para lidar com esse maciço grau de paralelismo.

Em particular, quando se trata de sistemas de tempo real, o problema de escalonamento de tarefas deve ser avaliado tendo em consideração esta tendência de paralelismo. De fato, resolver adequadamente este problema é um passo necessário para ampliar o domínio dos sistemas de tempo real. Entre os aspectos que foram abordados pela comunidade de pesquisa nesta área, é possível destacar três metas principais de pesquisa: (a) fornecer soluções de escalonamento que permitam uma elevada utilização do sistema; (b) simplificar a implementação do escalonador de forma a minimizar a reestruturação dos serviços no nível do sistema operacional; e (c) reduzir tanto quanto possível a sobrecarga no tempo de execução do sistema. Neste trabalho é descrito um mecanismo de escalo-

namento denominado HIME (*HIghest-priority Migration managed by EDF*), o qual leva estes três objetivos em consideração.

Considerando somente as metas (b) e (c), uma escolha natural para implementar escalonadores de tempo real em sistemas multiprocessadores é comumente conhecida como *abordagem de escalonamento particionada* (DAVIS; BURNS, 2011). O conjunto de tarefas de tempo real no sistema é particionado em subconjuntos disjuntos que são alocados a processadores dedicados. Este tipo de solução é bem atrativo, já que um vasto arcabouço teórico têm sido desenvolvido ao longo das décadas para o escalonamento em sistemas monoprocessados. Tal solução tem sido empregada em vários sistemas, principalmente nos fracamente acoplados (sistemas distribuídos) (TINDELL; BURNS; WELLINGS, 1995; TINDELL; CLARK, 1994; PALENCIA; HARBOUR, 1999). No entanto, nem sempre é possível particionar o sistema adequadamente caso se exiga mais do que 50% dos recursos de processamento (CARPENTER et al., 2004). Em outras palavras, a abordagem particionada negligencia a meta (a). Além disso, já que os custos de migração de tarefas em arquiteturas *multicore* diminuíram consideravelmente nos últimos anos, não permitir a migração de tarefas geralmente é uma decisão de projeto difícil de ser justificada, pelo menos em termos de pesquisa científica.

No outro lado do espectro do escalonamento estão as *abordagens de escalonamento globais*, de acordo com as quais as tarefas são enfileiradas em uma fila global e selecionadas para a execução em qualquer unidade de processamento do sistema. Como as tarefas podem migrar entre os processadores, o sistema pode ser melhor utilizado quando comparado com as abordagens particionadas. De fato, obter em teoria 100% de utilização do sistema é possível para algumas estrategias (BARUAH, 1995; FUNK; NADADUR, 2009; LEVIN et al., 2010; MCNAUGHTON, 1959; REGNIER et al., 2011). No entanto, gerenciar a fila global pode restringir a escalabilidade deste tipo de solução (DAVIS; BURNS, 2011). Além disso, abordagens de escalonamento globais impõem elevados custos em tempo de execução, devido ao número de preempções necessárias (BARUAH, 1995; FUNK; NADADUR, 2009; LEVIN et al., 2010). Embora utilizar 100% do sistema com baixa sobrecarga tem se mostrado possível, algumas dessas soluções restringem severamente o modelo do

sistema (MCNAUGHTON, 1959) enquanto outras exigem o projeto de estruturas de dados especiais para serem implementadas (REGNIER et al., 2011), as quais não podem atender projetos com escalonadores legados. Além disso, quando heurísticas de escalonamento globais mais simples são aplicadas (ANDERSSON; BARUAH; JONSSON, 2001; FUNK; GOOSSENS; BARUAH, 2001; CIRINEI; BAKER, 2007), pode ocorrer uma acentuada subutilização do sistema. Em todo caso, algumas das três metas mencionadas acima são negligenciadas pelas abordagens globais.

Mais recentemente, *abordagens de escalonamento híbridas* têm sido descritas. Estas estratégias alocam estaticamente subconjuntos de tarefas em alguns processadores impedindo-as de migrar. Algumas tarefas são permitidas migrar, mas de forma controlada, isto é, a abordagem híbrida pode ser considerada intermediária, entre o esquema de escalonamento particionado e global. Embora as abordagens híbridas pareçam atraentes em termos de atingir as metas (a) – (c), estas não necessariamente as cumprem. O modo de escalonar as tarefas migratórias e não migratórias pode requerer o projeto de mecanismos especiais de escalonamento (BLETSAS; ANDERSSON, 2011; ANDERSSON; BLETSAS, 2008; ANDERSSON; BLETSAS; BARUAH, 2008; MASSA; LIMA, 2010), que podem comprometer a meta (b). Algumas soluções exigem uma elevada sobrecarga de tempo de execução para alcançar uma alta utilização do sistema (BASTONI; BRANDENBURG; ANDERSON, 2011), que não está em consonância com a meta (c). Outras soluções não possuem garantias teóricas que atestam o cumprimento da meta (a) (KATO; YAMASAKI; ISHIKAWA, 2009; GEORGE; COURBIN; SOREL, 2011; BURNS et al., 2010).

HIME é uma abordagem de escalonamento híbrida. Um escalonador com a política EDF (LIU; LAYLAND, 1973) trata de atender as tarefas em cada processador, o que está em conformidade com a meta (b). Tarefas são alocadas aos processadores em tempo de projeto (*off-line*). Se nenhuma tarefa migratória é encontrada durante a fase de atribuição, o modelo de escalonamento do sistema se comporta exatamente como a abordagem particionada baseada no EDF. Caso contrário, no máximo uma tarefa migratória pode ser alocada por processador. Para cumprir a meta (c), a sobrecarga de tempo de execução da abordagem proposta é minimizada, restringindo tanto o número de migrações por tarefa

quanto o número de tarefas migratórias. Os custos associados à preempção também é compatível com os encontrados no EDF particionado. Além disso, a gestão das tarefas migratórias pode ser implementada de forma razoavelmente simples. Como as tarefas migratórias são requeridas a executar em nível de prioridade mais alto e há no máximo uma destas tarefas por processador, é o suficiente designar as suas instâncias o menor *deadline* absoluto. Quanto à meta (a), é mostrado que HIME pode alcançar uma elevada utilização do sistema apesar de seu baixo custo associado às preempções e às migrações.

1.3 TRABALHOS RELACIONADOS E CONTRIBUIÇÕES

Abordagens híbridas para escalonamento de sistemas de tempo real multiprocessados têm chamado recentemente a atenção da comunidade de pesquisa. Estas abordagens podem ser descritas por dois componentes: o *algoritmo de particionamento*, o qual determina como dividir e atribuir cada tarefa (ou parte dela) para um processador permanente; e o *algoritmo de escalonamento*, que determina como escalonar as tarefas em cada processador. A migração das tarefas também é controlada pelo algoritmo de escalonamento para impedir que uma mesma tarefa seja atendida simultaneamente em mais do que um processador. A família de algoritmos de escalonamento que utilizam este tipo de estratégia são geralmente referidas como *divisão de tarefas* (*task-splitting*) ou *semi-particionamento* (*semi-partitioning*) (ANDERSSON; TOVAR; SOUSA, 2010).

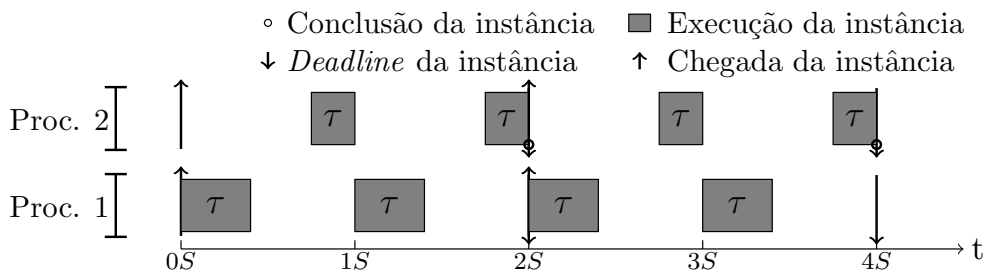


Figura 1.2. Gestão da tarefa τ em dois processadores usando execução de tarefa migratória baseada em janela de tempo, no qual a janela de tempo possui tamanho S e o *deadline* relativo de τ é $2S$

Duas estratégias de controle de migração são usualmente adotadas (ANDERSSON; PI-

NHO, 2010; ANDERSSON; TOVAR; SOUSA, 2010): *execução de tarefa migratória baseada em janela de tempo* (*slot-based split-task dispatching*) e *execução de tarefa migratória baseada na instância da tarefa* (*job-based split-task dispatching*). De acordo com a primeira (ver Figura 1.2), a escala de execução é organizada em intervalos de tempo de tamanhos iguais, ou seja, a escala de execução é dividida em janelas de tempos, as quais são sincronizadas em todos os processadores. A tarefa migratória é dividida de forma a executar em um processador no início da janela de tempo e em outro processador próximo do término. Segundo este esquema, um menor tamanho da janela de tempo conduz a um maior custo de preempção e de migração. Por outro lado, como a escalonabilidade do sistema é assegurada pelo limites da janela de tempo, reduzi-la também significa maior limite de escalonabilidade. Em outras palavras, há uma relação de compromisso entre os custos devido à preempção/migração e a utilização alcançada pelo sistema. Entre as diversas abordagens baseadas nesta estratégia descritas até o momento (BLETSAS; ANDERSSON, 2011; ANDERSSON; BLETSAS, 2008; ANDERSSON; BLETSAS; BARUAH, 2008; MASSA; LIMA, 2010), o esquema que possibilita a maior utilização do sistema é o NPS-F (*Notional Processor Scheduling-Fractional capacity*) (BLETSAS; ANDERSSON, 2011). Seus autores afirmam que se pode obter um menor limite superior de utilização entre 75% e 90% com custos aceitáveis de preempção e de migração. No entanto, um trabalho recente expôs que quando os custos em tempo de execução são considerados os valores reais de utilização obtidos para os conjuntos de tarefas serem escalonáveis é significativamente menor do que os limites teóricos (BASTONI; BRANDENBURG; ANDERSON, 2011). Além disso, este tipo de abordagem tem uma elevada complexidade de implementação (ANDERSSON; TOVAR; SOUSA, 2010).

Não há janela de tempo na abordagem de execução de tarefa migratória baseada na instância da tarefa (ver Figura 1.3). Em vez disso, quando uma instância de uma tarefa é liberada certa condição é configurada para especificar quando esta deve migrar para outro processador. Normalmente, essa condição é um limite associado ao tempo de computação que a instância deve executar em cada processador (KATO; YAMASAKI; ISHIKAWA, 2009; GUAN et al., 2010; GEORGE; COURBIN; SOREL, 2011; BURNS et al., 2010). O

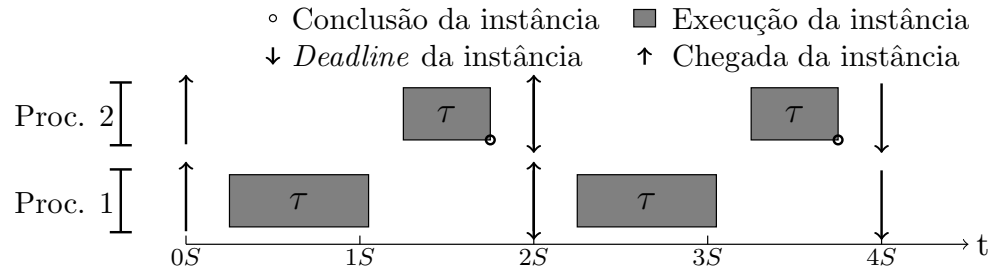


Figura 1.3. Gestão da mesma tarefa da Figura 1.2 em dois processadores usando execução de tarefa migratória baseada na instância da tarefa

SPA2 (*Semi-Partition Algorithm* 2) (GUAN et al., 2010) é um dos algoritmos de escalonamento desta classe, o qual utiliza como base a política de escalonamento *Rate-Monotonic* (RM) (LIU; LAYLAND, 1973) e reserva o nível de prioridade mais alta para gerenciar a tarefa migratória. De acordo com as suas regras de escalonamento, qualquer sistema cuja utilização não seja superior a $100 \ln 2\% \approx 69{,}3\%$ é corretamente escalonado. É importante mencionar que este é um dos melhores limite de utilização identificados para as estratégias que utilizam a execução de tarefa migratória baseada na instância da tarefa.

HIME também segue o esquema de execução de tarefa migratória baseada na instância da tarefa. Existe no máximo uma única tarefa migratória por processador e no máximo $\lfloor 0{,}5m \rfloor$ tarefas podem migrar em todo sistema. As tarefas não migratórias são escalonadas por EDF enquanto as migratórias são atendidas como as tarefas de mais alto nível de prioridade de forma similar ao SPA2. No entanto, HIME usa apenas um escalonador local com EDF para gerenciar tanto as tarefas migratórias quanto as não migratórias. Além disso, HIME garante que qualquer sistema utilizando até $72{,}\bar{2}\%$ de seus processadores seja corretamente escalonado, em oposição ao limite de 69,3% do SPA2. Pelo que foi averiguado, HIME fornece o maior menor limite superior de utilização do sistema entre as estratégias baseadas em execução de tarefa migratória baseada na instância da tarefa conhecidas. Além disso, experimentos indicam que, em média, HIME pode escalonar sistemas com utilização tão alta quanto as que são alcançadas pelo NPS-F.

Uma abordagem recente (BURNS et al., 2010), conhecida como esquema $C{=}D$, compartilha algumas características com HIME. Essa estratégia exige apenas um escalonador

local com EDF e possui baixo custo de implementação embora alguma computação adicional em tempo de execução (*on-line*) seja necessária para atualizar os *deadlines* das tarefas migratórias. Diferente de HIME, contudo, o esquema $C{=}D$ usa um teste de escalonabilidade pseudo-polinomial e pode haver até $m - 1$ tarefas migratórias uma vez que pode haver no máximo duas tarefas migrando por processador. Além disso, pelas informações obtidas o menor limite superior de utilização do sistema para o esquema $C{=}D$ ainda não foi derivado.

1.4 ESTRUTURA DA DISSERTAÇÃO

O restante deste trabalho é organizado como segue. A notação e o modelo computacional usados neste documento assim como os testes de escalonabilidade utilizados como base para o mecanismo proposto são abordadas no Capítulo 2. A estratégia deste mecanismo é descrita no Capítulo 3, enquanto que o Capítulo 4 aborda alguns aspectos teóricos e práticos do mesmo. Os resultados das simulações realizadas são apresentados no Capítulo 5 e as conclusões são tratadas no Capítulo 6.

CAPÍTULO 2

NOTAÇÃO E TESTE DE ESCALONABILIDADE

Este capítulo apresenta as bases do mecanismo proposto. A Seção 2.1 informa a notação e o modelo computacional adotados neste documento. Os testes de escalonabilidade para sistemas monoprocessados, adequados para a abordagem proposta, assim como experimentos inicias, são descritos na Seção 2.2.

2.1 NOTAÇÃO E MODELO DO SISTEMA

Neste trabalho considera-se um sistema composto por um conjunto de n tarefas esporádicas Γ a ser escalonado em m processadores idênticos. Tarefas são independentes uma das outras. Uma tarefa é representada por uma tupla (C, T), onde $C \leqslant T$ representa o seu tempo de computação e T é o intervalo de tempo mínimo entre a ativação de duas instâncias consecutivas. Neste documento, T também é chamado de período. A utilização de uma tarefa τ é representada por $U(\tau) = \frac{C}{T}$, enquanto que a utilização do conjunto de tarefas Γ é denotada por $U(\Gamma) = \sum_{\tau \in \Gamma} U(\tau)$.

Se $\tau = (C, T)$ é uma tarefa que chega no tempo t, o sistema deve escaloná-la para a execução de modo que τ não seja executada ao mesmo tempo em processadores distintos e C unidades de processamento seja destinada para τ dentro do intervalo $[t, t+T)$, ou seja, é assumido um modelo de tarefas com *deadline* implícito. Durante sua execução, as tarefas podem sofrer preempção e podem migrar entre processadores. Embora a abordagem de escalonamento descrita conduz a baixos custos de preempções e migrações, é adotada a suposição usual de que não há penalidades associadas com preempções ou migrações. Limites sobre preempções e migrações serão derivados na Seção 4.2.

2.1.1 Modelo de Escalonamento

Seja Γ uma conjunto de tarefas a ser escalonado em m processadores idênticos. O procedimento de alocação de tarefas apresentado neste trabalho é realizado em tempo de projeto (*off-line*) e responsável por definir os subconjuntos $\Gamma_j \subset \Gamma$, $j = 1, 2, \ldots, m$, cada um dos quais é designado a um processador. Denomina-se esses subconjuntos como *conjuntos de alocação*. As tarefas atribuídas a um único conjunto Γ_j são chamadas de *tarefas não migratórias*. A *tarefa migratória* é designada a mais de um processador e por isso é um elemento pertencente a mais do que um conjunto de alocação.

Define-se τ^i como a *i-ésima* porção de uma tarefa migratória τ alocada a algum processador. Sem perda de generalidade, assume-se que τ^i está pronta para execução logo após τ^{i-1} conclua sua execução. Além disso, denota-se $\Omega(\tau)$ como o conjunto de todas as tarefas atribuídas ao mesmo processador que τ e $\mho(\tau) = \Omega(\tau) \backslash \{\tau\}$. Por exemplo, se τ é uma tarefa alocada somente a Γ_j e Γ_k, $\Omega(\tau) = \Gamma_j \cup \Gamma_k$ e $\mho(\tau) = \Gamma_j \cup \Gamma_k \backslash \{\tau^1, \tau^2\}$.

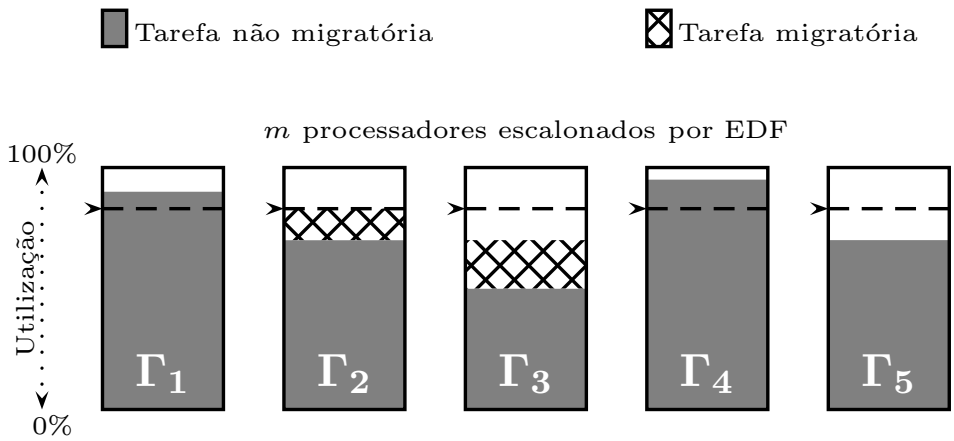


Figura 2.1. Modelo de escalonamento para um conjunto de tarefas alocadas em cinco processadores com no máximo uma tarefa migratória por processador

A Figura 2.1 ilustra o modelo de escalonamento assumido. Cada processador é equipado com um escalonador local EDF. Pode haver apenas tarefas não migratórias alocadas a alguns processadores, enquanto, algumas tarefas podem migrar entre alguns processadores. No primeiro caso, até 100% da capacidade do processador pode ser usada. No último caso, existe um limite máximo de capacidade do processador a ser utilizada, indicado na

figura por uma linha tracejada.

Se não houver nenhuma tarefa migratória após realizar o procedimento de alocação de tarefas, o esquema de escalonamento proposto reduz para uma abordagem particionada baseada no EDF. Na Figura 2.1 este é o caso para os processadores associados com Γ_1, Γ_4 e Γ_5. Caso exista uma tarefa migratória $\tau \in \Gamma_j$, é necessário reservar uma parte da capacidade de processamento do processador j a fim de executar τ. É imposta a restrição de que exista no máximo uma tarefa migratória por processador. Devido a esta restrição, alguns processadores podem ter sua capacidade de processamento subutilizada (abaixo do limite de escalonabilidade). Isto é ilustrado na Figura 2.1 para o processador associado a Γ_3.

Processadores distintos podem ter limites de escalonabilidade distintos. Isto é devido ao fato de os testes de escalonabilidade que são utilizados dependerem das características das tarefas atribuídas a cada processador. Isto será elucidado na Seção 2.2.

A tarefa migratória em cada processador é sempre escalonada com o mais alto nível de prioridade independentemente de seu *deadline* absoluto atual. Como esta abordagem de escalonamento pode ser implementada por uma simples estratégia de manipulação dos *deadlines* das tarefas, um escalonador local com EDF em cada processador é suficiente para lidar com tarefas migratórias e as não migratórias.

Antes de apresentar detalhes sobre a derivação da abordagem proposta, serão apresentadas na próxima seção as condições de escalonabilidade empregadas para a atribuição das tarefas migratórias.

2.2 TESTE DE ESCALONABILIDADE

Nesta seção, são tratadas as condições de escalonabilidade em sistemas monoprocessados. Portanto, assume-se que há uma tarefa τ para ser executada no mais alto nível de prioridade independentemente de seu *deadline* atual e τ é atribuído a apenas um conjunto de alocação. Tarefas em $\mho(\tau)$ são escalonadas de acordo com EDF.

Inicialmente (Seção 2.2.1) são derivados alguns resultados básicos para escalonar uma tarefa de alta prioridade juntamente com um conjunto de tarefas atendidas por EDF. Com base nesse resultado, são derivados novos testes de escalonabilidade na Seção 2.2.2. Os desempenhos destes testes são avaliados na Seção 2.2.3.

2.2.1 Resultado Básico

O Lema 2.1 estabelece que os testes de escalonabilidade comumente utilizados na política de escalonamento RM (LIU; LAYLAND, 1973) podem ser usados no contexto de HIME.

Lema 2.1 *Seja $\tau = (C, T)$ uma tarefa e considere o conjunto de tarefas $\mho(\tau)$ a ser escalonado em um processador de modo que T não é maior do que o período de qualquer tarefa em $\mho(\tau)$. Se τ é escalonada no mais alto nível de prioridade e todas as outras tarefas em $\mho(\tau)$ são escalonadas de acordo com a política EDF, não há perda de deadlines desde que*

$$\gamma(U(\tau), U(\mho(\tau))) \leqslant B \tag{2.1}$$

onde: $\gamma(u_1, u_2)$, u_1 e $u_2 \in [0, 1]$, é uma função monotonicamente crescente de u_1 e u_2 que fornece um teste de escalonabilidade suficiente, assegurando que não há perda de deadlines quando duas tarefas com utilizações u_1 e u_2 são escalonadas de acordo com RM desde que $\gamma(u_1, u_2) \leqslant B$, na qual B é uma constante pertencente ao intervalo $(0, 1]$.

Prova: Primeiramente nota-se que τ não pode perder seus *deadlines* uma vez que $U(\tau) = C/T \leqslant 1$ e a mesma executa com a máxima prioridade.

Assuma por contradição que a Equação (2.1) é respeitada e alguma tarefa em $\mho(\tau)$ perde seu *deadline* no tempo d. Chame esta de tarefa problema. Sem perda de generalidade, assuma que nenhuma tarefa perde seu *deadline* antes de d. Além disso, deixe $t < d$ ser a última vez em que o processador está ocioso. Se tal tempo não existe, assuma que $t = 0$. A demanda máxima de processamento devido às tarefas em $\mho(\tau)$ no intervalo

$[t, d)$ é dada pela Equação (2.2). É importante notar que nesta equação a função *chão* é necessária, pois as instâncias com *deadlines* superiores a d não são consideradas.

$$C' = \sum_{\tau_i \in \mho(\tau)} \left\lfloor \frac{d-t}{T_i} \right\rfloor C_i \tag{2.2}$$

Defina a tarefa $\tau' = (C', d)$ a ser escalonada em conjunto com τ de acordo com a política de escalonamento RM. Assuma que τ é liberada ao mesmo tempo em que τ'. Por construção, τ' também perde o seu *deadline* até o instante d. Note que $T < d$ e assim a escala de τ e τ' está em conformidade com a política RM. No entanto, $U(\tau') \leqslant U(\mho(\tau))$ e assim $\gamma(U(\tau), U(\tau')) \leqslant B$ é mantida, o que contradiz a suficiente do teste γ. ∎

O Lema 2.1 fornece uma base para o esquema de escalonamento proposto. Qualquer teste de escalonabilidade baseado em utilização para sistemas monoprocessados de prioridade fixa pode ser utilizado na Equação (2.1). Por exemplo, se o teste de escalonabilidade de Liu e Layland (LIU; LAYLAND, 1973) for considerado, a Equação (2.1) assume a forma de $U(\tau) + U(\mho(\tau)) \leqslant 2\left(\sqrt{2} - 1\right)$. Outra opção é o teste do limite hiperbólico (BINI; BUTTAZZO; BUTTAZZO, 2003), resultando em $(U(\tau) + 1)(U(\mho(\tau)) + 1) \leqslant 2$. Em vez de usar testes de escalonabilidade conhecidos, foram derivados novos testes que exploram o modelo de escalonamento de HIME obtendo melhores resultados.

2.2.2 Testes Desenvolvidos

Baseado no Lema 2.1, assume-se doravante o seguinte:

Hipótese 2.1 *O conjunto $\Omega(\tau)$ deve ser escalonado em um processador de tal forma que (a) $\mho(\tau)$ seja escalonado de acordo com a política de escalonamento EDF, (b) $\tau = (C, T)$ sempre executa no mais alto nível de prioridade e (c) $T \leqslant \min_{\tau' \in \mho(\tau)}(T')$.*

A seguir três novos testes de escalonabilidade suficientes são derivados, declarados nos Lemas 2.2, 2.3 e 2.4. Cada um destes testes explora algumas características particulares

do conjunto de tarefas considerado.

Lema 2.2 (Teste 1) *Nenhuma tarefa em $\Omega(\tau)$ perde seu deadline desde que a Hipótese 2.1 e a Equação (2.3) sejam respeitadas.*

$$\left(\frac{T}{\min_{\tau_i \in \mho(\tau)}(T_i)} + 1\right) U(\tau) + U(\mho(\tau)) \leqslant 1 \tag{2.3}$$

Prova: A tarefa de maior prioridade τ não perde seus *deadlines* uma vez que não sofre interferência de qualquer outra tarefa e $U(\tau) \leqslant 1$. Considere que alguma tarefa $\tau_i \in \mho(\tau)$ perde seu *deadline* no instante de tempo d sendo que a mesma foi liberada no instante $r < d$. Além disso, considere $t < d$ a última vez em que o processador não está ocioso durante $[t, d)$, mas está ocioso antes de t. Se tal tempo não existe, assuma $t = 0$. Note que $t \leqslant r$ e no pior caso $d - r = T_i$. Para simplificar a notação, defina $\Delta = r - t$. Como τ_i perde seu *deadline* em d, a demanda de computação entre t e d deve ser maior que $d - t = T_i + \Delta$. Para as tarefas em $\mho(\tau)$ deve-se considerar a execução das instâncias das tarefas que têm *deadlines* inferiores ou iguais a d. Além disso, já que τ interfere na execução de qualquer tarefa em $\mho(\tau)$, sua ativação em $[t, d)$ deve ser contabilizada. Sabe-se que τ não é ativada mais do que $\left\lceil \frac{T_i+\Delta}{T} \right\rceil$ vezes durante $[t, d)$. Calculando a demanda total e considerando que τ_i perde seu *deadline* obtém-se que:

$$\begin{aligned}
& \left\lceil \frac{T_i + \Delta}{T} \right\rceil C + \sum_{\tau_j \in \mho(\tau)} \left\lfloor \frac{T_i + \Delta}{T_j} \right\rfloor C_j > T_i + \Delta \\
\Rightarrow \quad & \left\lceil \frac{T_i + \Delta}{T} \right\rceil C + U(\mho(\tau))(T_i + \Delta) > T_i + \Delta \\
\Leftrightarrow \quad & \left(\frac{T_i + \Delta}{T} + 1\right) \frac{C}{T_i + \Delta} + U(\mho(\tau)) > 1 \\
\Leftrightarrow \quad & U(\tau) + \frac{U(\tau)T}{T_i + \Delta} + U(\mho(\tau)) > 1
\end{aligned}$$

Como $\Delta \geqslant 0$, resulta que

$$\left(\frac{T}{T_i} + 1\right) U(\tau) + U(\mho(\tau)) > 1 \tag{2.4}$$

A Equação (2.4) deve valer para qualquer tarefa $\tau_i \in \mho(\tau)$ que perde seu *deadline*. Como o lado esquerdo desta condição é maximizado quando τ_i tem o período mínimo entre todas as tarefas em $\mho(\tau)$, a Equação (2.3) é uma condição suficiente de escalonabilidade, conforme afirmado. ∎

Lema 2.3 (Teste 2) *Nenhuma tarefa em $\Omega(\tau)$ perde seu deadline desde que a Hipótese 2.1 e a Equação* (2.5) *sejam respeitadas.*

$$U(\tau) + \sum_{\tau_i \in \mho(\tau)} \frac{T_i}{\left\lfloor \frac{T_i}{T} \right\rfloor T} U(\tau_i) \leqslant 1 \tag{2.5}$$

Prova: Assuma que o conjunto de tarefas $\mho(\tau)$ e a tarefa τ respeitem a Equação (2.5). Como $U(\tau) \leqslant 1$, a tarefa de mais alta prioridade τ não perde seu *deadline*. Para mostrar que as tarefas em $\mho(\tau)$ cumprem seus requisitos temporais, considere um sistema Γ' obtido a partir de $\mho(\tau)$ como se segue. Para cada tarefa $\tau_i = (C_i, T_i) \in \mho(\tau)$ existe uma tarefa $\tau_i' = (C_i', T)$ em Γ', onde $C_i' = \frac{C_i}{\left\lfloor \frac{T_i}{T} \right\rfloor}$. Agora considere o escalonamento de τ e Γ' com τ sendo a tarefa de maior prioridade e Γ' sendo escalonado pela política de escalonamento EDF. Como todas as tarefas de Γ' tem o mesmo período e

$$U(\Gamma') = \sum_{\tau_i \in \Gamma'} \frac{C_i'}{T} = \sum_{\tau_i \in \Gamma'} \frac{C_i}{\left\lfloor \frac{T_i}{T} \right\rfloor} \frac{1}{T} = \sum_{\tau_i \in \mho(\tau)} \frac{T_i}{\left\lfloor \frac{T_i}{T} \right\rfloor T} U(\tau_i),$$

a Equação (2.5) resulta que $U(\tau) + U(\Gamma') \leqslant 1 \Rightarrow C + \sum_{\tau_j \in \Gamma'} C_j' \leqslant T$, onde $C + \sum_{\tau_j \in \Gamma'} C_j'$ é o tempo de resposta no pior caso de qualquer tarefa em Γ'. Isto significa que a escalonabilidade de Γ' é assegurada pela a Equação (2.5).

Em qualquer intervalo de tempo $L \geqslant T$ a demanda de processamento de Γ' é dada pela Equação (2.6), enquanto a demanda de $\mho(\tau)$ é igual a $\sum_{\tau_j \in \mho(\tau)} \left\lfloor \frac{L}{T_j} \right\rfloor C_j$.

$$\sum_{\tau_j \in \Gamma'} \left\lfloor \frac{L}{T} \right\rfloor C_j'. \tag{2.6}$$

Como $T_j = \left\lfloor \frac{T_j}{T} \right\rfloor T + \epsilon_j$, onde $0 \leqslant \epsilon_j < T$. Reescrevendo, na Equação (2.6), os períodos das tarefas de Γ' em função de T_j resulta que

$$\sum_{\tau_j \in \Gamma'} \left\lfloor \frac{L}{\frac{T_j - \epsilon_j}{\left\lfloor \frac{T_j}{T} \right\rfloor}} \right\rfloor C'_j \geqslant \sum_{\tau_j \in \Gamma'} \left\lfloor \frac{L}{T_j - \epsilon_j} \right\rfloor \left\lfloor \frac{T_j}{T} \right\rfloor C'_j = \sum_{\tau_j \in \Gamma'} \left\lfloor \frac{L}{T_j - \epsilon_j} \right\rfloor C_j \geqslant \sum_{\tau_j \in \mho(\tau)} \left\lfloor \frac{L}{T_j} \right\rfloor C_j \quad (2.7)$$

A Equação (2.7) comprova que a demanda de processamento de $\mho(\tau)$ não é superior a demanda de Γ' e como ambos os conjuntos de tarefas são escalonados por EDF, a escalonabilidade de Γ' implica no cumprimento dos requisitos temporais das tarefas de $\mho(\tau)$. Dessa modo, a Equação (2.5) é uma condição suficiente de escalonabilidade. ∎

O último teste de escalonabilidade proposto é dado pelo Lema 2.4. Este teste é utilizado para se determinar o menor limite superior de utilização de HIME (ver Seção 4.1).

Lema 2.4 (Teste 3) *Nenhuma tarefa em $\Omega(\tau)$ perde seu deadline desde que a Hipótese 2.1 e a Equação* (2.8) *sejam respeitadas.*

$$\left(\frac{U(\mho(\tau))}{\left\lfloor \frac{\min_{\tau_i \in \mho(\tau)}(T_i)}{T} \right\rfloor} + 1 \right) U(\tau) + U(\mho(\tau)) \leqslant 1 \quad (2.8)$$

Prova: Seja t^{φ} o tempo disponível para executar as tarefas em $\mho(\tau)$ dentro do intervalo de tempo L. Sabe-se que se $\forall L \geqslant \min_{\tau_i \in \mho(\tau)}(T_i)$ a Equação (2.9) for respeitada, então as tarefas em $\mho(\tau)$ cumprem seus *deadlines* (BLETSAS; ANDERSSON, 2011, Teorema 8).

$$L \sum_{\tau_i \in \mho(\tau)} U(\tau_i) \leqslant t^{\varphi} \Rightarrow U(\mho(\tau)) \leqslant \frac{t^{\varphi}}{L} \quad (2.9)$$

O mínimo valor de t^{φ} ocorre quando τ é ativada periodicamente. Além disso, os valores de L que minimizam o lado direito da Equação (2.9) ocorrem quando o início e término do intervalo de tempo L coincidem com a ativação e conclusão de instâncias de τ, respectivamente. Isso ocorre porque se L é aumentado por ϵ unidades, $0 < \epsilon \leqslant T - C$,

o valor de t^{φ} também tem acréscimo de ϵ. Por sua vez, diminuir o valor de L por um valor positivo $\epsilon < C$ mantém t^{φ} constante. Em outras palavras, os valores de L a serem considerados são dados por $L = (k+j)T + C$, onde $k = \left\lfloor \frac{\min_{\tau_i \in \mho(\tau)}(T_i)}{T} \right\rfloor$ e $j \in Z_+$. Neste caso, para cada intervalo de tempo de tamanho T, existem $(T - C)$ unidades de tempo disponíveis para a execução de tarefas em $\mho(\tau)$, o que conduz a $t^{\varphi} = (k+j)(T-C)$. Reescrevendo a Equação (2.9),

$$\begin{aligned} U(\mho(\tau)) &\leqslant \frac{(k+j)(T-C)}{(k+j)T+C} \\ U(\mho(\tau)) &\leqslant \frac{(k+j)(T-U(\tau)T)}{(k+j)T+U(\tau)T} \\ U(\mho(\tau)) &\leqslant \frac{1-U(\tau)}{1+\dfrac{U(\tau)}{k+j}} \end{aligned} \tag{2.10}$$

O lado direito da Equação (2.10) é uma função crescente em j. Fazendo-se $j = 0$, a Equação (2.10) torna-se a Equação (2.8), conforme requirido. Como $U(\tau) \leqslant 1$, a tarefa de mais alta prioridade τ não perde seu *deadline* e isto completa a prova. ∎

Dado um conjunto de tarefas não migratórias $\mho(\tau)$ alocado a um processador, os Lemas 2.2, 2.3 e 2.4 estabelecem as condições para reserva do tempo de processamento para executar a tarefa migratória, tarefa de maior prioridade, τ no processador de modo que nenhum *deadline* seja perdido. Por uma questão de notação, é importante definir a máxima reserva de processador disponível para executar τ no nível de prioridade mais elevado. Isto é declarado mais formalmente pelo Teorema 2.1.

Teorema 2.1 *Considere um conjunto de tarefas Γ a ser escalonado em um processador. A máxima utilização do processador disponível para executar uma tarefa no mais elevado nível de prioridade com período T sobre a Hipótese 2.1 é*

$$\sigma(\Gamma, T) = \max\left(\sigma_1(\Gamma, T), \sigma_2(\Gamma, T), \sigma_3(\Gamma, T)\right) \tag{2.11}$$

onde

$$\sigma_1(\Gamma, T) = \frac{1 - U(\Gamma)}{1 + \frac{T}{\min_{\tau_i \in \Gamma}(T_i)}} \tag{2.12}$$

$$\sigma_2(\Gamma, T) = 1 - \sum_{\tau_i \in \Gamma} \frac{C_i}{\left\lfloor \frac{T_i}{T} \right\rfloor T} \tag{2.13}$$

$$\sigma_3(\Gamma, T) = \frac{1 - U(\Gamma)}{1 + \frac{U(\Gamma)}{\left\lfloor \frac{\min_{\tau_i \in \Gamma}(T_i)}{T} \right\rfloor}} \tag{2.14}$$

Prova: As Equações 2.12 – 2.14 provêm diretamente dos Lemas 2.2 – 2.4, como segue:

$$\text{Lema 2.2} \Rightarrow \left(\frac{T}{\min_{\tau_i \in \Gamma}(T_i)} + 1 \right) \sigma_1(\Gamma, T) + U(\Gamma) = 1 \Leftrightarrow \sigma_1(\Gamma, T) = \frac{1 - U(\Gamma)}{1 + \frac{T}{\min_{\tau_i \in \Gamma}(T_i)}}$$

$$\text{Lema 2.3} \Rightarrow \sigma_2(\Gamma, T) + \sum_{\tau_i \in \Gamma} \frac{T_i}{\left\lfloor \frac{T_i}{T} \right\rfloor T} U(\tau_i) = 1 \Leftrightarrow \sigma_2(\Gamma, T) = 1 - \sum_{\tau_i \in \Gamma} \frac{C_i}{\left\lfloor \frac{T_i}{T} \right\rfloor T}$$

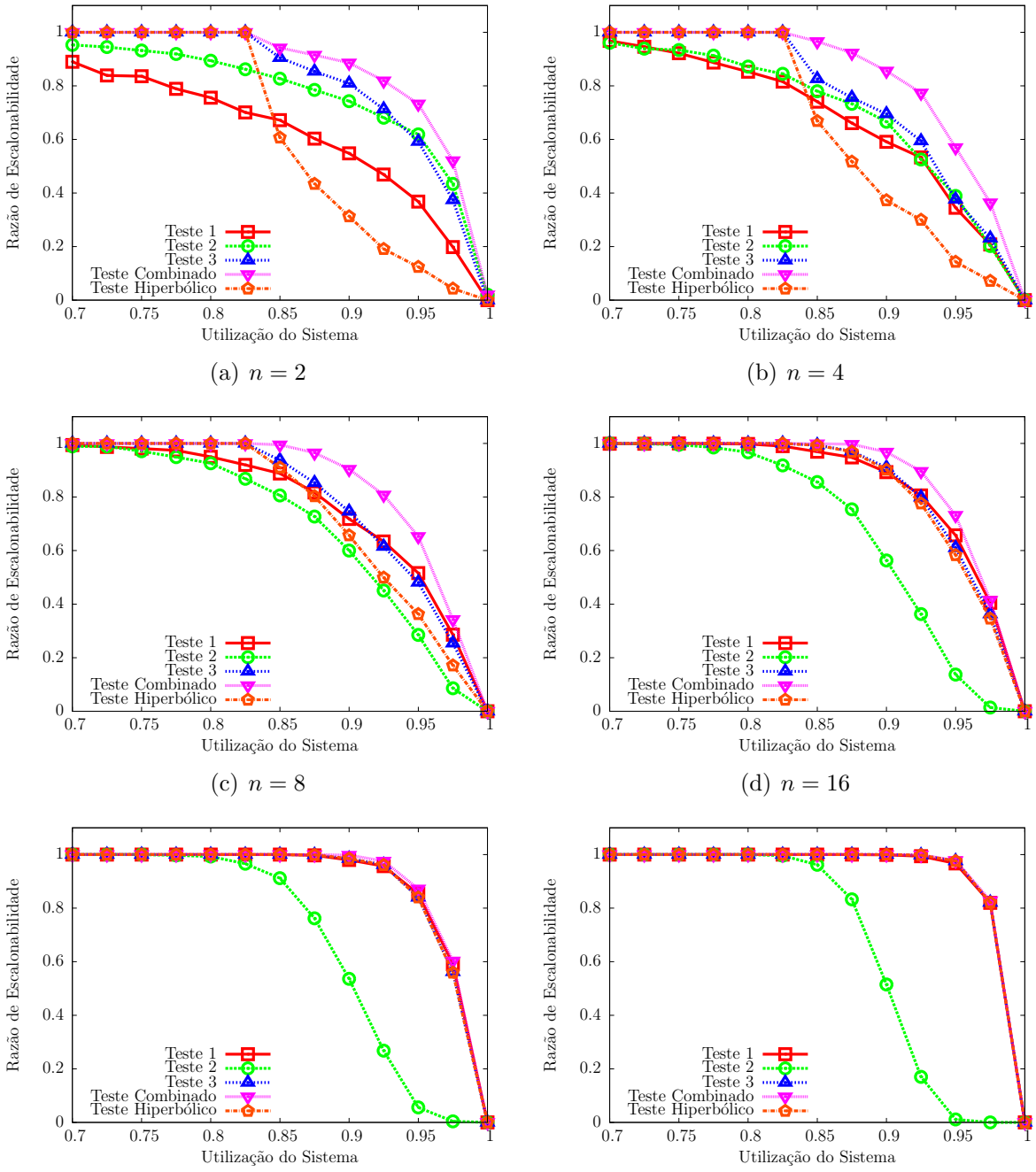
$$\text{Lema 2.4} \Rightarrow \left(\frac{U(\Gamma)}{\left\lfloor \frac{\min_{\tau_i \in \Gamma}(T_i)}{T} \right\rfloor} + 1 \right) \sigma_3(\Gamma, T) + U(\Gamma) = 1 \Leftrightarrow \sigma_3(\Gamma, T) = \frac{1 - U(\Gamma)}{1 + \frac{U(\Gamma)}{\left\lfloor \frac{\min_{\tau_i \in \Gamma}(T_i)}{T} \right\rfloor}}$$

Como estes lemas estabelecem testes de escalonabilidade suficientes, qualquer um deles pode ser utilizado individualmente. O melhor desempenho é dado selecionando o máximo destas equações. ∎

2.2.3 Avaliação dos Testes

A fim de comparar o desempenho dos testes de escalonabilidade derivados foram gerados 66.000 conjuntos de tarefas, com $n = 2, 4, 8, 16, 32, 64$ tarefas cada. As utilizações dos conjuntos de tarefas consideradas variaram de 70% a 100%. Para cada valor de utilização dos conjuntos de tarefas, 1.000 conjuntos foram gerados. Todos estes conjuntos sintéticos de tarefas foram gerados de acordo com um gerador de tarefas aleatórias descrito em outro trabalho (BINI; BUTTAZZO, 2005). Os períodos das tarefas foram gerados de acordo com uma distribuição log-uniforme de números inteiros no intervalo $[10, 1.000]$,

como recomendado por outros autores (BURNS et al., 2010).



(a) $n = 2$ (b) $n = 4$

(c) $n = 8$ (d) $n = 16$

(e) $n = 32$ (f) $n = 64$

Figura 2.2. Comparação entre os testes de escalonabilidade propostos e o teste baseado no limite hiperbólico

A Figura 2.2 retrata o comportamento dos testes de escalonabilidade propostos em termos da *razão de escalonabilidade*, ou seja, o percentual de conjuntos de tarefas aceitos

como escalonáveis. Para efeito de comparação, o desempenho do teste de escalonabilidade baseado no limite hiperbólico (Teste Hiperbólico), descrito por Bini *et al.* (BINI; BUTTAZZO; BUTTAZZO, 2003), também foi traçado. O teste de Liu e Layland (LIU; LAYLAND, 1973) não foi considerado uma vez que conduz ao limite de $2\left(\sqrt{2}-1\right) < 0{,}83$ acima do qual nenhum conjunto de tarefas é considerado escalonável.

Como pode ser observado na Figura 2.2, todos os testes de escalonabilidade tendem a ter um melhor desempenho para conjuntos com maior número de tarefas. Isto ocorre porque a utilização das tarefas geradas tende a ser menor para cada tarefa quando n cresce. Uma menor utilização da tarefa de maior prioridade resulta em menos interferência na execução das demais tarefas, conduzindo a melhores resultados.

Em média, o teste hiperbólico se comporta pior do que os testes propostos. Além disso, os testes 1 e 3 têm desempenho similar para $n \geqslant 8$. Note que estes testes utilizam uma relação entre os períodos de apenas duas tarefas enquanto o teste 2 infla a utilização do conjunto de tarefas considerando todas as tarefas. A Figura 2.2 também mostra o desempenho obtido utilizando a disjunção dos três testes propostos (Teste Combinado), ou seja, para que um conjunto de tarefas seja considerado escalonável basta que o mesmo seja considerado escalonável por ao menos um dos testes propostos.

2.3 SUMÁRIO

Neste capítulo foram descritos a notação e o modelo de escalonamento, que é voltado para sistemas compostos por tarefas esporádicas com *deadline* implícito. Neste modelo as tarefas não migratórias, a maioria das tarefas do sistema, são escalonadas por EDF (LIU; LAYLAND, 1973), enquanto que as tarefas migratórias são atendidas como o maior nível de prioridade e deve existir apenas uma destas tarefas por processador. Também foram apresentados os testes de escalonabilidade que servem como base para a abordagem proposta. Por fim, foi realizada uma comparação entre o teste baseado no limite hiperbólico (BINI; BUTTAZZO; BUTTAZZO, 2003) e os três testes desenvolvidos, os quais geralmente obtiveram melhores resultados.

Com base nas informações tratadas neste capítulo pode-se descrever a abordagem de escalonamento foco deste trabalho, que será detalhada no próximo capítulo.

CAPÍTULO 3

HIME

Este capítulo descreve o mecanismo de escalonamento desenvolvido neste trabalho. Inicialmente, na Seção 3.1, aborda-se o procedimento de alocação das tarefas. Então é demonstrado na Seção 3.2 que um conjunto de tarefas alocadas de acordo com este procedimento e escalonado por EDF cumpre os requisitos temporais de suas tarefas. Para ilustrar a estratégia proposta, um exemplo é apresentado na Seção 3.3.

3.1 PROCEDIMENTO DE ALOCAÇÃO DAS TAREFAS

O Algoritmo 1 descreve a estratégia de alocação usada em HIME, a qual é baseada em alguma heurística de *bin packing* (COFFMAN-JR.; GAREY; JOHNSON, 1997). Inicialmente, o conjunto de tarefas é ordenado de acordo com algum critério. Nesta dissertação, foram consideradas duas possíveis ordenações, ordem decrescente do período da tarefa (*Decreasing Period* - DP) e ordem decrescente da utilização da tarefa (*Decreasing Utilization* - DU). Os critérios DP e DU originam duas versões de HIME, HIME/DP e HIME/DU, respectivamente.

O *loop* principal do algoritmo de alocação termina quando todas as tarefas em Γ são devidamente alocadas ou não é possível alocar mais tarefas. No primeiro caso, o algoritmo retorna sucesso. Caso contrário, o algoritmo falha e o sistema é considerado não escalonável por HIME.

O procedimento de alocação tenta evitar a criação de tarefas migratórias (Seção 3.1.1). Contudo, se alguma tarefa migratória for definida, deve haver critérios para selecionar quais das tarefas serão migratórias (Seção 3.1.2) e para quais processadores estas serão designadas (Seção 3.1.3).

Algoritmo 1: Procedimento de alocação das tarefas

Entrada: Critério de ordenação `ord` $\in \{\text{DP}, \text{DU}\}$;
Entrada: Conjunto de n tarefas Γ;
Entrada: Número de processadores m;
Saída: Conjuntos alocados $\Gamma_1, \Gamma_2, \ldots, \Gamma_m$;

```
1  Ordene as tarefas em Γ de acordo com o critério de ordenação ord;
2  Γ_j ← {}, j = 1, 2, ..., m;
3  enquanto houver tarefa a ser alocada faça
4      Preencha os conjuntos Γ_j, j = 1, 2, ..., m, de acordo com alguma heurística de
       bin-packing até que todas as tarefas sejam alocadas ou seja encontrada uma tarefa
       τ_s ∈ Γ que não possa ser alocada;
5      se ∃τ_s então
6          caso ∃Γ_i, U(τ_s) + U(Γ_i) ⩽ 1
7              Γ_i ← Γ_i ∪ {τ_s};
8          caso ∃Γ_i, Γ_j, U(Γ_i) + U(Γ_j) ⩽ 1
9              Γ_i ← Γ_i ∪ Γ_j;
10             Γ_j ← {τ_s};
11         senão  // alocação de tarefa migratória
12             Seja Π ⊆ {Γ_1, Γ_2, ..., Γ_m} os conjuntos de alocação que não possuem tarefa
               migratória e que para cada Γ_j ∈ Π, U(Γ_j) < 1;
13             se ord = DP então
14                 k ← migTaskAlloc(τ_s, Π);
15             senão
16                 k ← migTaskAllocDU(τ_s, Π);
17             se k = 0 então
18                 retorna falha;
19 retorna sucesso;
```

3.1.1 Tarefas Não Migratórias

As tarefas não migratórias são alocados como descrito nas Linhas 4 – 10 do Algoritmo 1. A Linha 4 é responsável por alocar as tarefas de acordo com alguma heurística de *bin-packing*. Não há restrição com relação a heurística utilizada. No entanto, é exigido que o procedimento de alocação seja interrompido logo que alguma tarefa τ_s não possa ser inteiramente atribuída a algum processador segundo a heurística de *bin-packing* escolhida. Isto é requerido para preservar a ordem de alocação de tarefas (DP ou DU) sem violar a Hipótese 2.1. Como pode ser observado, quando uma tarefa τ_s não pode ser alocada a um conjunto Γ_j: ou (a) $U(\Gamma_j) + U(\tau_s) > 1$, se não houver tarefa migratória em Γ_j; ou (b)

$\sigma(\mho(\tau^j) \cup \{\tau_s\}, T) < U(\tau^j)$, onde $\tau^j = (C^j, T)$ é uma parte da tarefa migratória alocada a Γ_j.

Quando existe uma tarefa τ_s que não pode ser alocada de acordo com a Linha 4 do Algoritmo 1, as Linhas 6 – 10 verificam se é possível fazer τ_s uma tarefa não migratória:

- A Linha 6 verifica se há algum processador com capacidade disponível. Caso exista tal processador, τ_s é alocada ao mesmo. Para alguma heurísticas de *bin-packing* isso pode ser necessário. Por exemplo, considere um sistema com dois processadores e um conjunto de quatro tarefas tal que $U(\tau_1) = 0{,}8$, $U(\tau_2) = 0{,}7$, $U(\tau_3) = 0{,}3$ e $U(\tau_4) = 0{,}2$. A heurística de *bin-packing Next-Fit* sobre o critério de ordenação DU resulta em $\Gamma_1 = \{\tau_1\}$ e $\Gamma_2 = \{\tau_2, \tau_3\}$. A tarefa $\tau_s = \tau_4$ não pode ser alocada de acordo com a Linha 4. As Linhas 6 – 7 asseguram que $\Gamma_1 = \{\tau_1, \tau_4\}$ para este exemplo. Para heurísticas como *First-Fit* esta parte do algoritmo não é necessária.

- Nas Linhas 8 – 10 as tarefas previamente alocadas são rearranjadas para dar espaço para τ_s. Por exemplo, considere três tarefas para serem alocadas a dois processadores. Suponha que os seus períodos são tais que $T_1 \geqslant T_2 \geqslant T_3$ e que suas utilizações sejam $U(\tau_1) = 0{,}4$, $U(\tau_2) = 0{,}3$ e $U(\tau_3) = 0{,}8$. Usando o critério de ordenação DP, a heurística de *bin-packing Worst-Fit* alocaria estas tarefas de forma que $\Gamma_1 = \{\tau_1\}$ e $\Gamma_2 = \{\tau_2\}$, falhando ao alocar $\tau_s = \tau_3$ na Linha 4. Como pode ser observado, após a execução das Linhas 8 – 10 para este exemplo, $\Gamma_1 = \{\tau_1, \tau_2\}$ e $\Gamma_2 = \{\tau_3\}$, conduz a um sistema particionado. Novamente, para algumas heurísticas de *bin-packing*, como *First-Fit*, não há necessidade das Linhas 8 – 10.

É importante ressaltar que, caso não seja gerado nenhuma tarefa migratória, o sistema torna-se escalonado por EDF particionado. Lidar com os casos explicados acima é uma tentativa de fazer o procedimento de alocação o mais geral possível assim como reduzir o número de tarefas migratórias.

3.1.2 Seleção das Tarefas Migratórias

As tarefas migratórias são alocadas nas Linhas 11 – 16. A forma como este processo de alocação é realizado depende do critério de ordenação de tarefas utilizado. Em todo caso, o procedimento de alocação de tarefas migratórias recebe tanto a tarefa τ_s, que não pode ser alocada nas Linhas 4 – 10, como os conjuntos de alocação que não contêm tarefas migratórias. Após a tarefa migratória ser alocada, os conjuntos que atenderão τ_s são alterados para incluí-la.

Como as tarefas estão em ordem decrescente de seus períodos pelo critério DP, o período de τ_s não é maior do que o período de qualquer outra tarefa previamente alocada a Γ_j, $j = 1, 2, \ldots, m$; lembre que a Hipótese 2.1 item (c) é necessário para o Teorema 2.1. Portanto, τ_s é uma escolha natural para ser a tarefa migratória τ. Com o critério de ordenação DU, o período de τ_s pode não respeitar a Hipótese 2.1 item (c). Neste caso, outra tarefa τ' com período mínimo é selecionado como a tarefa migratória. Desde que τ' foi alocada como uma tarefa não migratória nas etapas anteriores do Algoritmo 1 e $U(\tau') \geqslant U(\tau_s)$, a função `migTaskAllocDU` substitui τ' por τ_s. Esta função é descrita abaixo.

Função migTaskAllocDU(τ_s, Π)

```
1 Selecione τ = (C, T) de forma que T = min{T_i, τ_i ∈ {τ_s} ∪ Π};
2 se τ ≠ τ_s então
3 |   Seja Γ_j o conjunto Ω(τ);
4 |   Γ_j ← Γ_j\{τ} ∪ {τ_s};
5 retorna migTaskAlloc(τ,Π);
```

3.1.3 Seleção dos Processadores para as Tarefas Migratórias

Um objetivo natural do escalonamento em sistemas multiprocessados é minimizar o número de migrações. Além disso, como HIME não pode usar a máxima capacidade de processamento de um processador quando há uma tarefa migratória alocada (ver Figura 2.1), é importante reduzir o desperdício desta capacidade de processamento no sistema.

Estes dois objetivos são alcançados de uma forma simples durante a execução da função `migTaskAlloc`, a qual é descrita abaixo. Seja $\tau = (C, T)$ uma tarefa migratória para a qual k processadores devem ser designados. Inicialmente, os conjuntos alocados são ordenados em ordem decrescente da capacidade de processamento disponível para a execução da tarefa de maior prioridade de período T. Então, $k-1$ conjuntos de alocação são selecionados segundo esta ordem (Linha 4 – 9). Ao alocar a última porção de τ, τ^k, o conjunto de alocação escolhido deve ser aquele com a máxima utilização e que possui $U(\tau^k) \leqslant \sigma(\mho(\tau^k), T)$ (Linha 10). Em outras palavras, ao escolher os k processadores que atendem τ, seleciona-se: (a) $k-1$ processadores em ordem decrescente da capacidade de processamento disponível para a tarefa migratória e (b) o último processador é selecionado de modo que $U(\Omega(\tau^k))$ seja maximizada sem comprometer a escalonabilidade de $\mho(\tau)$. A função retorna $k > 1$ (número de processadores em que τ foi alocada) ou 0 se não conseguir alocar τ.

Função migTaskAlloc(τ, Π)

```
1  Ordene Π de modo que ∀Γ_i, Γ_j ∈ Π, (i < j) ⇒ σ(Γ_i, T) ⩾ σ(Γ_j, T);
2  k ← 1;
3  C^k ← σ(Γ_k, T)T;
4  enquanto C > C^k e Π ≠ {} faça
5      Γ_k ← Γ_k ∪ {τ^k = (C^k, T)};
6      C ← C − C^k;
7      Π ← Π\Γ_k;
8      k ← k + 1;
9      C^k ← σ(Γ_k, T)T;
10 Selecione Γ_j ∈ Π de forma que σ(Γ_j, T) − C/T ⩾ 0 e U(Γ_j) seja máximo;
11 se ∃Γ_j então
12     Γ_j ← Γ_j ∪ {τ^k = (C, T)};
13     retorna k;
14 retorna 0;
```

Como pode ser observado as funções `migTaskAlloc` e `migTaskAllocDU` podem ser implementadas com complexidade $O(n \log n)$. Primeiro, os valores de σ dos conjuntos alocados são calculados (complexidade $O(n)$) e então este valores são utilizados na ordenação. Isto e o fato do *loop* principal do Algoritmo 1 não realizar mais do que $O(n)$ passos resulta que o mesmo possui complexidade $O(\beta + n^2 \log n)$, onde β é a complexi-

dade associada a heurística de *bin-packing*. Portanto, o Algoritmo 1 possui complexidade polinomial, caso a heurística de *bin-packing* utilizada execute em tempo polinomial.

3.2 ESCALONABILIDADE

Uma vez que todas as tarefas no conjunto Γ sejam alocadas pelo Algoritmo 1, todos os requisitos temporais serão cumpridos durante a execução do sistema, como é demonstrado a seguir.

Teorema 3.1 *Se o Algoritmo 1 retorna sucesso para um determinado conjunto de tarefas* Γ*, nenhuma tarefa perde seu deadline quando escalonadas respeitando a Hipótese 2.1.*

Prova: O Algoritmo 1 retorna sucesso apenas quando todas as tarefas são alocadas. Sejam $\Gamma_1, \Gamma_2, \ldots, \Gamma_m$ conjuntos de tarefas definidos por este algoritmo. Se não houver nenhuma tarefa migratória em algum conjunto Γ_j, então todas as tarefas não migratórias cumprem seus requisitos temporais uma vez que estas são escalonadas por escalonadores locais com a política de escalonamento EDF e o procedimento de alocação garante que $U(\Gamma_j) \leqslant 1$, $j = 1, 2, \ldots, m$. Assim, considere que existe alguma tarefa migratória $\tau = (C, T)$ que é atendida em k processadores.

Pelo modelo do sistema, τ não sofre interferência de qualquer outra tarefa alocada nos seus k processadores. Portanto, o tempo de início de τ^j depende apenas da finalização de τ^{j-1}, $j > 1$. Dessa forma, as instâncias de τ atendem seus *deadlines*, pois

$$C = \sum_{j=1}^{k} C^j \leqslant T.$$

Agora, considere as tarefas não migratórias alocadas aos mesmos k processadores de τ. Do ponto de vista de qualquer tarefa em $\mho(\tau^j)$, τ^j comporta-se como uma tarefa esporádica com período T. Como o Algoritmo 1 assegura que T não é maior do que o período de qualquer outra tarefa em $\mho(\tau^j)$ (Seção 3.1.2) e $U(\tau^j) \leqslant \sigma(\mho(\tau^j), T)$ (pela função `migTaskAlloc`), não há perda de *deadlines* pelo Teorema 2.1. ∎

3.3 EXEMPLO

Para ilustrar o comportamento da estratégia proposta, considere a Figura 3.1 que mostra uma possível escala de execução para quatro tarefas ativadas no instante zero utilizando HIME/DU. O sistema é composto por dois processadores. Os parâmetros e a utilização das tarefas são informados na Tabela 3.1. Inicialmente, o procedimento de alocação (Algoritmo 1) atribui τ_1 a Γ_1 e τ_2 a Γ_2. Como τ_3 não pode ser designada como uma tarefa não migratória e existem tarefas previamente alocadas com menor período, a função `migTaskAllocDU` troca τ_3 por τ_1. Portanto, τ_1 será uma tarefa migratória atendida de tal forma que $\tau_1^1 = (1, 2)$ e $\tau_1^2 = (0{,}25, 2)$. Além disso, como a inserção de τ_4 em Γ_2 preserva a escalonabilidade tem-se que $\Gamma_1 = \{\tau_3, \tau_1^1\}$ e $\Gamma_2 = \{\tau_2, \tau_1^2, \tau_4\}$.

Tarefa	C	T	U
τ_1	1,25	2	0,625
τ_2	1,65	3	0,550
τ_3	3,00	6	0,500
τ_4	1,20	6	0,200

Tabela 3.1. Informações das tarefas

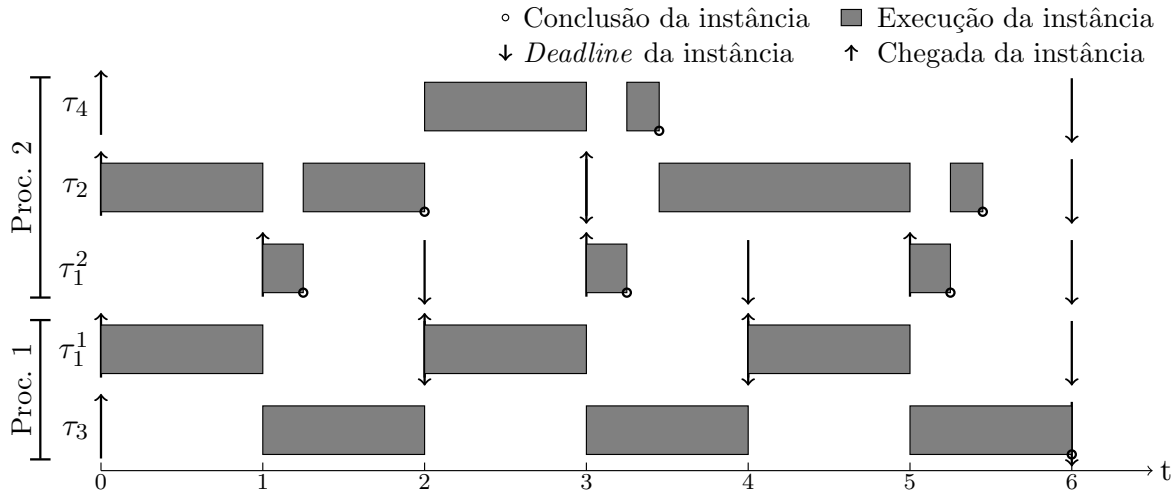


Figura 3.1. Escala de execução gerada por HIME/DU

No instante 0, a primeira instância de τ_2 e τ_1 (atendida como τ_1^1) começam a executar. No instante 1, τ_1^1 consome completamente sua reserva de processamento permitindo a execução da primeira instância de τ_3. Além disso, neste instante, a primeira instância

de τ_1 migra para o segundo processador causando a preempção de τ_2, sendo atendida como τ_1^2 até o instante 1,25, quando completa a sua execução, finalizando a primeira instância de τ_1, permitindo τ_2 continuar a executar. A primeira instância de τ_2 termina sua execução no instante 2, permitindo a execução de τ_4, que conclui sua computação no instante 3,45. No instante 2 também ocorre a ativação de uma nova instância de τ_1, que começa a ser atendida como τ_1^1, interrompendo a execução de τ_3 até que τ_1^1 termine sua reserva no instante 3. Esta sequência de preempções de τ_3 pelas instâncias de τ_1^1 se repete até sua conclusão no instante 6. Ainda no instante 3, surge uma nova instância de τ_2, e τ_1 migra novamente para o segundo processador causando a preempção de τ_4, que executa sem sofrer preempção entre 3,45 a 5,25, concluindo sua execução no instante 5,45.

3.4 SUMÁRIO

Neste capítulo foi descrito HIME (*HIghest-priority Migration managed by EDF*), uma abordagem de escalonamento híbrida baseada no EDF (LIU; LAYLAND, 1973). Uma das principais características desta abordagem é que a mesma tenta evitar a criação de tarefas migratórias e caso não haja este tipo de tarefa, o sistema torna-se escalonado por EDF particionado. O algoritmo que descreve a estratégia de alocação usada em HIME foi descrito e demonstrou-se que o mesmo serve como teste de escalonabilidade. Este algoritmo não impõem a utilização de uma heurística de *bin packing* (COFFMAN-JR.; GAREY; JOHNSON, 1997) específica e permite a utilização de dois critérios de ordenação das tarefas, DP (ordem decrescente do período) e DU (ordem decrescente da utilização). Os critérios DP e DU originam duas versões de HIME, HIME/DP e HIME/DU, respectivamente.

Com a abordagem proposta devidamente descrita é interessante deduzir propriedades da mesma que retratem tanto alguns aspectos teóricos quanto práticos, o que será tratado no próximo capítulo.

CAPÍTULO 4

ASPECTOS TEÓRICOS E PRÁTICOS DE HIME

Questões relacionadas ao menor limite superior de utilização de HIME são consideradas na Seção 4.1, a qual demonstra que o Algoritmo 1 retorna sucesso para conjuntos de tarefas sempre que a utilização do sistema não seja superior a $0{,}7\bar{2}m$ (Seção 4.1.1). Além disso, a Seção 4.1.2 mostra que quando o critério de ordenação DU é utilizado este limite pode ser melhorado para $\left(\frac{\sqrt{17}}{3} - 1\right) 2m \approx 0{,}75m$, contanto que algumas modificações no procedimento de alocação sejam feitas. Algumas questões relacionadas a custo de preempção e migração são discutidas na Seção 4.2, enquanto a Seção 4.3 tece alguns comentários sobre aspectos de implementação.

4.1 LIMITE DE UTILIZAÇÃO DO SISTEMA

Os limites de utilização derivados aqui são encontrados usando o teste de escalonabilidade estabelecido pelo Lema 2.4. Assim, assume-se doravante que quando uma tarefa migratória é alocada a um processador j, $\sigma(\Gamma_j, T) = \sigma_3(\Gamma_j, T)$. De fato, é utilizada nesta seção uma versão conservadora de σ_3, como declarado a seguir:

Hipótese 4.1 *Para que qualquer tarefa migratória $\tau = (C, T)$ seja alocada em um conjunto Γ_j, assume-se que a Equação (2.14) é o único meio para determinar a máxima capacidade de processamento a ser destinada a uma tarefa migratória τ. Como o valor mínimo desta capacidade ocorre quando $T = \min_{\tau_i \in \Gamma_j}(T_i)$, adota-se*

$$\sigma(\Gamma_j, T) = \sigma_3(\Gamma_j, T) = \frac{1 - U(\Gamma_j)}{1 + U(\Gamma_j)} \tag{4.1}$$

É importante mencionar que a Equação (4.1) é equivalente ao teste do limite hiperbólico (BINI; BUTTAZZO; BUTTAZZO, 2003), o qual obteve pior desempenho do que o

teste relacionado à Equação (2.14), como visto na Seção 2.2.3. Além disso, não considerar os outros dois testes de escalonabilidade derivados na Seção 2.2.3 significa que as possíveis melhorias na escalonabilidade do sistema não são consideradas. De fato, como será visto no Capítulo 5, o comportamento médio do Algoritmo 1 está bem acima dos limites derivados neste capítulo.

4.1.1 Demostração do Limite de Utilização

Inicialmente, são informadas algumas propriedades que são utilizadas nesta seção:

Observação 4.1 *Seja Γ um conjunto de tarefas a serem escalonadas em m processadores. Se $\tau \in \Gamma$ é uma tarefa migratória alocada para $k > 1$ destes processadores pelo Algoritmo 1, sob a Hipótese 4.1, as seguintes relações são mantidas.*

$$U(\tau) + U(\mho(\tau^j)) > 1, \quad j = 1, 2, \ldots, k \tag{4.2}$$

$$U(\mho(\tau^i)) + U(\mho(\tau^j)) > 1, \quad i, j = 1, 2, \ldots, k,\ i \neq j \tag{4.3}$$

$$\sum_{j=1}^{k-1} \frac{1 - U(\mho(\tau^j))}{1 + U(\mho(\tau^j))} < U(\tau) \tag{4.4}$$

Prova: Como τ é uma tarefa migratória, as Linhas 6 e 8 do Algoritmo 1 garantem as equações 4.2 e 4.3, respectivamente. Além disso, como τ requer k processadores, a Equação (4.4) provém diretamente da Hipótese 4.1. ∎

A partir de considerações associadas à Observação 4.1, foi determinado um limite de utilização do sistema. Qualquer sistema, cuja utilização seja inferior a este limite, é considerado escalonável por HIME. Para identificar o limite de utilização, primeiro derivou-se a mínima utilização associada a k processadores que atendem uma mesma tarefa migratória. Então este resultado foi estendido para m processadores. Além disso, também foi identificado que, caso o critério DU seja adotado, o limite de escalonabilidade vinculado a esta abordagem pode ser melhorado.

Para se determinar o limite de utilização em questão, a seguinte propriedade foi utilizada:

Observação 4.2 *Sejam* $x_1, x_2, \ldots, x_k$ *valores tais que* $0 \leqslant x_j \leqslant 1$, $j = 1, 2, \ldots, k$. *Se* $\bar{x} = \frac{1}{k}\sum_{j=1}^{k} x_j$, *então*

$$k\frac{1-\bar{x}}{1+\bar{x}} \leqslant \sum_{j=1}^{k} \frac{1-x_j}{1+x_j} \tag{4.5}$$

Prova: Seja $f(x) = \frac{1-x}{1+x}, 0 \leqslant x \leqslant 1$. Pela convexidade desta função no intervalo $[0, 1]$, sabe-se que, pela desigualdade de Jensen (JENSEN, 1906),

$$f(\bar{x}) = f\left(\frac{1}{k}\sum_{i=1}^{k} x_j\right) \leqslant \frac{1}{k}\sum_{i=1}^{k} f\left(x_j\right)$$

e assim a observação segue. ∎

Agora será considerado um caso particular em que a tarefa migratória τ é alocada a $k = 2$ processadores.

Lema 4.1 *Se uma tarefa migratória* τ *foi alocada a dois processadores pelo Algoritmo 1 sob a Hipótese 4.1, a utilização destes processadores é superior a* 1,5.

Prova: Sabe-se pela Observação 4.1 que

$$\begin{aligned} U(\mho(\tau^1)) + U(\mho(\tau^2)) &> 1 \\ U(\mho(\tau^1)) + U(\tau) &> 1 \\ U(\mho(\tau^2)) + U(\tau) &> 1 \end{aligned}$$

Resumindo, estas equações conduzem a $U(\Omega(\tau)) = U(\mho(\tau^1)) + U(\mho(\tau^2)) + U(\tau) > \dfrac{3}{2}$, conforme requerido. ∎

Quando $k \geqslant 3$, a mínima utilização do processador que pode ser alcançada depende do número de processadores considerados:

Lema 4.2 *Se τ é uma tarefa migratória alocada a um conjunto de $k \geqslant 3$ processadores pelo Algoritmo 1 e respeitando a Hipótese 4.1, então*

$$U(\Omega(\tau)) > k\frac{1+\bar{u}^2}{1+\bar{u}} - \frac{1-\bar{u}}{1+\bar{u}} \tag{4.6}$$

onde

$$\bar{u} = \frac{1}{k-1}\sum_{j=1}^{k-1} U(\mho(\tau^j))$$

Prova: De acordo com as Observações 4.1 e 4.2,

$$U(\tau) > \sum_{j=1}^{k-1} \frac{1-U(\mho(\tau^j))}{1+U(\mho(\tau^j))} \geqslant (k-1)\frac{1-\bar{u}}{1+\bar{u}}$$

A função `migTaskAlloc` (Seção 3.1.3) seleciona os processadores em ordem decrescente da capacidade de processamento destinado a atender a tarefa migratória. Além disso, pela Hipótese 4.1, esta capacidade de processamento segue pela Equação (4.1), uma função contínua e decrescente. Dessa forma, a escolha dos processadores na função `migTaskAlloc` equivale a uma seleção em ordem crescente de utilização dos processadores. Estas observações implicam que $U(\mho(\tau^k)) \geqslant \bar{u}$. Portanto,

$$\begin{aligned}
U(\Omega(\tau)) &= U(\mho(\tau)) + U(\tau) \\
U(\Omega(\tau)) &> U(\mho(\tau^k)) + \sum_{j=1}^{k-1} U(\mho(\tau^j)) + (k-1)\frac{1-\bar{u}}{1+\bar{u}} \\
U(\Omega(\tau)) &> \bar{u} + (k-1)\bar{u} + (k-1)\frac{1-\bar{u}}{1+\bar{u}} \\
U(\Omega(\tau)) &> k\frac{1+\bar{u}^2}{1+\bar{u}} - \frac{1-\bar{u}}{1+\bar{u}}
\end{aligned}$$

∎

Com base nas informações obtidas pelos Lemas 4.1 e 4.2, pode-se determinar um limite de utilização.

Teorema 4.1 *Um conjunto de tarefas* Γ *com a utilização* $U(\Gamma)$ *é escalonável em* m *processadores por* HIME *desde que* $U(\Gamma) \leqslant 0{,}7\bar{2}m$.

Prova: Assuma que existam tarefas migratórias no sistema uma vez que os processadores com este tipo de tarefas podem não ser totalmente utilizados. Os Lemas 4.1 e 4.2 informam um valor inferior à mínima utilização do processador para $\Omega(\tau)$ alocado em $k > 1$ processadores. Dividindo este valor pelo número de processadores que τ foi alocada resulta na utilização média dos processadores de 0,75 (para dois processadores) e

$$\frac{1+\bar{u}^2}{1+\bar{u}} - \frac{1-\bar{u}}{k(1+\bar{u})} \quad \text{(para } k \geqslant 3 \text{ processadores)}. \tag{4.7}$$

O valor mínimo dado pela Equação (4.7) ocorre quando $k = 3$. Além disso, de acordo com a Observação 4.1, Equação (4.3), a utilização média dos conjuntos alocados em um sistema que necessita de pelo menos uma tarefa migratória é maior do que 0,5. Portanto, a utilização média para k processadores que executam a tarefa migratória τ não pode ser inferior a

$$\frac{1+0{,}5^2}{1+0{,}5} - \frac{1-0{,}5}{3(1+0{,}5)} = \frac{13}{18} = \underbrace{0{,}7\bar{2}}_{k=3} < \underbrace{0{,}75}_{k=2}.$$

Como a máxima capacidade de processamento desperdiçada ocorre quando a tarefa migratória τ é alocada a 3 processadores, conservadoramente assume-se que este tipo de alocação ocorre para todas as tarefas migratórias do sistema o que conduz ao limite declarado. ∎

4.1.2 Melhoria Considerando o Critério DU

Se o critério DU é considerado, uma melhoria em relação ao limite anteriormente derivado pode ser obtida. Para efeito de intuição, considere a Equação (4.6) e um cenário de pior caso onde $\bar{u}$ é um pouco superior a 0,5. Isto significa que quando a tarefa migratória τ for atribuída em k processadores, estes processadores estão em média pelo menos mais da metade de suas capacidades de processamento utilizadas. Assim, a Equação (4.6)

assume a forma

$$\frac{U(\Omega(\tau))}{k} > \frac{5}{6} - \frac{1}{3k} \quad \text{para } k \geqslant 3$$

indicando um limite inferior para a utilização média de $\Omega(\tau)$. É importante notar que esta função é crescente em k e para $k = 4$, $\frac{U(\Omega(\tau))}{k} > 0{,}75$, o que é a mesma média obtida quando $k = 2$ pelo Lema 4.1. Desse modo, o único valor de k que possibilita uma menor utilização em k processadores é quando $k = 3$. Portanto, só é necessário assegurar uma melhor utilização média para este valor de k. Nesta seção, vamos lidar com este caso. Primeiro, será derivado no Teorema 4.2 o novo limite de escalonabilidade considerando-se que as tarefas são alocadas de acordo com o critério de ordenação DU e isto não viola a Hipótese 2.1. Então, propõem-se uma modificação na função `migTaskAllocDU` para que o resultado deste teorema possa ser aplicado.

Teorema 4.2 *Um conjunto de tarefas Γ com utilização $U(\Gamma)$ é escalonável em m processadores por* HIME*/DU desde que $U(\Gamma) \leqslant \left(\frac{\sqrt{17}}{3} - 1\right) 2m$, caso o período e a utilização de qualquer tarefa migratória $\tau \in \Gamma$ não sejam maiores do que o período e a utilização de qualquer outra tarefa em $\mho(\tau)$, respectivamente.*

Prova: Seja $\bar{u} = \frac{1}{k-1} \sum_{j=1}^{k-1} \mho(\tau^j)$. Como τ é alocado a k processadores, a partir das Observações 4.1 e 4.2 sabe-se que

$$U(\tau) > \sum_{j=1}^{k-1} \frac{1 - \mho(\tau^j)}{1 - \mho(\tau^j)} \geqslant (k-1)\frac{1-\bar{u}}{1+\bar{u}} \tag{4.8}$$

As tarefas são alocadas aos processadores em ordem decrescente da sua utilização, e por hipótese, $U(\tau)$ não é maior do que a utilização de qualquer outra tarefa alocada antes de τ. Assim, $k'U(\tau) \leqslant \sum_{i=1}^{k'} U(\mho(\tau^j))$ para algum $k' = 1, 2, \ldots, k$. Em particular, para $k' = k - 1$, $U(\tau) \leqslant \bar{u}$. Combinando esta observação com a Equação (4.8),

$$\bar{u} > (k-1)\frac{1-\bar{u}}{1+\bar{u}} \Rightarrow \bar{u}^2 + k\bar{u} - (k-1) > 0$$

Resolvendo esta desigualdade, tem-se que:

$$\bar{u} > \frac{\sqrt{k^2 + 4(k-1)} - k}{2}$$

Como o limite mínimo para $\bar{u}$ ocorre quando $k = 3$, tem-se que $\bar{u} > 0{,}5(\sqrt{17} - 3)$. Aplicando o resultado do Lema 4.2, este valor pode ser usado na Equação (4.6), resultando em

$$\frac{U(\Omega(\tau))}{k} > \frac{2\left(\sqrt{17} - 3\right)}{3}.$$

Conservadoramente, assumindo que o cenário que leva ao máximo desperdício de processador ocorre para os m processadores, o teorema segue. ∎

É importante ressaltar que o teorema acima é baseado no fato de que a tarefa migratória τ possui o mínimo período entre todas as tarefas em $\mho(\tau)$ mesmo se a ordem DU for aplicada. Lembre-se que ao escolher uma tarefa migratória a função `migTaskAllocDU` (Linha 2) não garante esta propriedade.

Uma abordagem simples para superar este efeito é o de modificar os parâmetros de τ para que tanto a escalonabilidade do sistema seja preservada quanto o Teorema 4.2 possa ser aplicado. Por exemplo, considere $\tau = (C, T)$ e adote que $T_{\min} = \min_{\tau_i \in \mho(\tau)}(T_i)$. Define-se $\tau' = (C', T')$ de modo que $T' = \frac{T}{\delta}$ e $C' = T'U(\tau)$, no qual $\delta = \left\lceil \frac{T}{T_{\min}} \right\rceil$. Uma vez que T é um múltiplo de T' e τ' tem a mesma utilização de τ, não é difícil verificar que a escalonabilidade do sistema ainda será preservada se τ' for usada para executar τ. Claramente, esta abordagem aumenta por um fator de δ tanto o número de preempções que ocorrem em $\Omega(\tau)$ quanto o número de migrações de τ.

No entanto, é importante notar que provavelmente esse custo não seja necessário. Como será visto no Capítulo 5, as simulações indicam que sistemas com elevada utilização podem ser efetivamente escalonados por HIME sem reduzir os períodos das tarefas migratórias. Com base nesta observação, não se considerou o aumento dos custos de migração/preempção para favorecer uma melhora no pior caso do limite de escalonabilidade.

4.2 NÚMERO DE PREEMPÇÕES E DE MIGRAÇÕES

De acordo com a estratégia proposta, qualquer tarefa migratória τ pode ser alocada a no máximo m processadores. Além disso, pelo modelo assumido, existe no máximo uma tarefa migratória por processador. Estas observações levam a uma propriedade interessante sobre o número máximo de migrações:

Lema 4.3 *Se o Algoritmo 1 é aplicado com sucesso a um conjunto de tarefas a serem escalonadas em m processadores, há no máximo $\lfloor 0{,}5m \rfloor$ tarefas migratórias. Além disso, se $k > 0$ tarefas migratórias são alocadas no sistema, nenhuma instância de uma tarefa migra mais do que $m - 2k + 1$ vezes durante sua execução.*

Prova: O número máximo de tarefas migratórias resulta diretamente do modelo de escalonamento. Como uma tarefa migratória é alocada a pelo menos dois processadores, o número máximo de processadores para o qual uma tarefa migratória pode ser alocada é $m - 2(k-1)$. Isso resulta em no máximo $m - 2(k-1) - 1$ migrações para uma instância de uma tarefa. ∎

O número máximo de migrações e preempções que ocorrem durante um intervalo de tempo também pode ser limitado:

Lema 4.4 *Seja Γ um conjunto de tarefas e suponha que após executar com êxito o Algoritmo 1 para Γ, $k > 0$ tarefas migratórias são definidas para serem executadas em $p > k$ processadores. Além disso, assuma que $P_{EDF}(\Delta)$ é o número de preempções devido ao escalonamento de $n - k$ tarefas não migratórias por EDF particionado em um intervalo de tempo Δ. Em qualquer intervalo de tempo Δ,* HIME *não gera mais do que $\left\lceil \frac{\Delta}{\min_{\tau \in \Gamma}(T)} \right\rceil (m - k)$ migrações e o número de preempções não é superior a $P_{EDF}(\Delta) + \left\lceil \frac{\Delta}{\min_{\tau \in \Gamma}(T)} \right\rceil p$.*

Prova: Sem perda de generalidade, assuma que $\tau_1, \tau_2, \ldots, \tau_k$ são as tarefas migratórias definidas pelo Algoritmo 1. Considere que cada tarefa migratória $\tau_i = (C_i, T_i)$ é alocada

em m_i processadores, $i = 1, 2, \ldots, k$. Pelo modelo de escalonamento adotado, T_i é o mínimo período entre as tarefas em $\Omega(\tau_i)$. Além disso, cada instância de τ_i migra $m_i - 1$ vezes durante sua execução e existem no máximo $\left\lceil \frac{\Delta}{T_i} \right\rceil$ instância de τ_i. Portanto, o número total de migrações em Δ não é superior a

$$\sum_{i=1}^{k}(m_i - 1)\left\lceil \frac{\Delta}{T_i} \right\rceil \leqslant (m-k)\left\lceil \frac{\Delta}{\min_{i=1}^{k}(T_i)} \right\rceil \leqslant (m-k)\left\lceil \frac{\Delta}{\min_{\tau_i \in \Gamma}(T_i)} \right\rceil .$$

Além disso, cada instância de uma tarefa migratória τ_i provoca um ponto de preempção em cada processador que τ_i executa. Isto e o fato de que o número de instâncias de uma tarefa migratória que ocorrem em Δ não ser maior do que $\left\lceil \frac{\Delta}{\min_{\tau \in \Gamma}(T)} \right\rceil$ resulta no limite desejado. ∎

Note que pela propriedade acima o número máximo de preempções causadas por HIME é compatível (ligeiramente superior) aos observados no EDF particionado. Isto é devido ao fato de que as tarefas migratórias, executadas no mais alto nível de prioridade, podem ser consideradas como tarefas extras alocadas para os processadores que as executam.

4.3 QUESTÕES DE IMPLEMENTAÇÃO

A principal característica da abordagem proposta referente à implementação é que a mesma não requer quaisquer características especiais do sistema operacional de tempo real. Em relação ao escalonamento, embora HIME seja baseado em gerir tanto tarefas com prioridade fixa quanto tarefas com prioridade dinâmica, apenas escalonadores locais equipados com EDF são necessários. De fato, como há no máximo uma única tarefa migratória por processador, esta tarefa de maior prioridade pode ser escalonada através da modificação do seu *deadline* para ser o mínimo em cada processador. Isso pode ser obtido configurando o *deadline* absoluto das instâncias da tarefa migratória como zero e utilizando somente a tradicional versão do EDF em cada processador como um algoritmo de escalonamento local.

Quando se trata de atender tarefas migratórias e não migratórias, foi identificado que é necessário um serviço de afinidade de tarefas (*task affinity service*) disponibilizado pelo sistema operacional. Este serviço é relativamente simples e indica em qual processador uma dada tarefa deve executar. Por exemplo, considere uma tarefa $\tau = (C, T)$ liberada no instante de tempo t. Esta tarefa é enviada para um determinado processador através do serviço de afinidade. Um variável global associada a cada tarefa indica o processador atual de τ. Se τ é uma tarefa não migratória, seu processador não muda em tempo de execução. Caso contrário, o processador em que τ executa é modificado após o término de cada τ^j. Em outras palavras, não há necessidade de recursos de temporização especiais no nível do sistema operacional. Apenas um único temporizador (*timer*) por processador para medir o tempo real é necessário. Este temporizador é configurado quando τ^j inicia sua execução. Quando terminar o tempo reservado a τ^j, a afinidade da tarefa é alterada para indicar o próximo processador associado à τ.

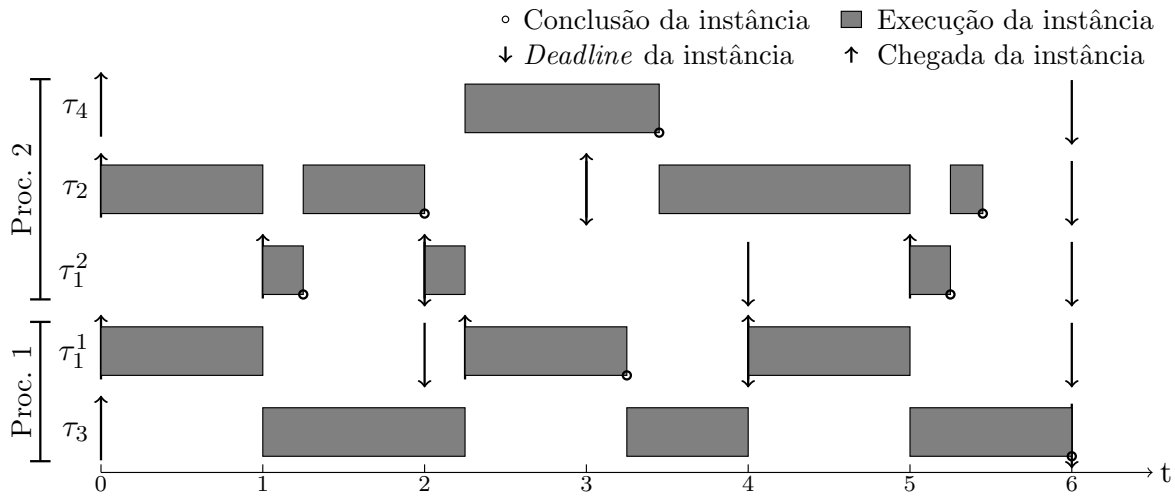


Figura 4.1. Escala de execução da Figura 3.1 com espelhamento de tarefa

Como uma tarefa migratória pode iniciar sua execução em qualquer processador a que foi alocada, é possível evitar o custo de migração entre diferentes instâncias da mesma tarefa migratória. Este serviço, conhecido como *espelhamento de tarefa*, pode ser implementado de forma simples em HIME por meio da manipulação da afinidade da tarefa, ou seja, atualizando a afinidade da tarefa somente quando a instância precisa migrar. Quando uma nova instância de tarefa migratória τ é ativada, esta pode começar sua

execução no último processador que a instância de τ que a precede executou. A Figura 4.1 ilustra a utilização desta estratégia no exemplo da Seção 3.3. Como pode ser observado, está técnica pode ser útil para trabalhar em conjunto com a otimização apresentada na Seção 4.1.2.

Outro efeito colateral positivo relacionado ao atendimento da tarefa migratória é que seu tempo de resposta no pior caso é igual ao tempo de computação mais o que é gasto pelo sistema para gerenciar esta tarefa. Em outras palavras, os custos de migração, quando considerado, pode ser diretamente incorporado ao custo de execução, o que geralmente simplifica a análise do sistema.

4.4 SUMÁRIO

Neste capítulo foram apresentadas algumas propriedades associadas a HIME, como o número máximo de preempções causadas pela mesma, que é próximo aos observados no EDF particionado, e a existência de no máximo $\lfloor 0{,}5m \rfloor$ tarefas migratórias no sistema, cujas instâncias podem migrar no máximo $m - 1$ vezes. Além disso, derivou-se o menor limite superior de utilização de HIME e uma otimização foi posposta para HIME/DU. Foram ainda abordados alguns aspectos relativos à simplicidade de implementação de HIME, pois o mesmo não requer mecanismos especiais no nível do sistema operacional.

Como este capítulo investigou as características de HIME no cenário de pior caso é importante realizar uma avaliação do mesmo sem este tipo de imposição, o que será abordado no próximo capítulo.

CAPÍTULO 5

AVALIAÇÃO

Foi realizado um extenso conjunto de simulações para avaliar a abordagem proposta. Os parâmetros da simulação são descrito na Seção 5.1. Este capítulo mostra alguns resultados desses experimentos, descrevendo o comportamento geral de HIME. Dois tipos de experimentos foram realizados. Primeiro (Seção 5.2), foi avaliado o efeito da utilização de diferentes heurísticas de *bin-packing* na Linha 4 do Algoritmo 1. No segundo experimento (Seção 5.3) HIME foi comparado com duas outras abordagens de escalonamento.

5.1 PARÂMETROS DE GERAÇÃO DO CONJUNTOS DE TAREFA

Cada conjunto sintético de tarefas foi gerado de acordo com o procedimento descrito por Emberson *et al.* (EMBERSON; STAFFORD; DAVIS, 2010) usando uma implementação disponível (EMBERSON, 2010). Os parâmetros para cada conjunto de tarefas gerado Γ foram semelhantes aos utilizados na Seção 2.2.3, isto é, a utilização média do sistema, $\frac{U(\Gamma)}{m}$, encontra-se no intervalo [0,7, 1,0] e os períodos das tarefas seguem uma distribuição log-uniforme de números inteiros pertencente ao intervalo $[10, 1.000]$.

Apesar dos diversos tamanhos de conjuntos de tarefas terem sido considerados durante os experimentos, é apresentado neste capítulo resultados para uma quantidade relativamente pequena de conjuntos. Conjuntos grandes de tarefas tendem a apresentar uma utilização pequena por tarefa e isto aumenta as chances dos sistemas serem puramente particionados. Devido a este fato, considera-se o fator $\alpha = \frac{n}{m}$ como um parâmetro para caracterizar o tipo de conjunto de tarefas gerado. Por exemplo, para $\alpha \leqslant 1$ é claro que cada tarefa pode ser alocada a um processador dedicado. Além disso, se $\alpha > 2{,}5$, foi observado nos experimentos que HIME geralmente gera apenas poucas tarefas migratórias. Isso ocorre porque a utilização de cada tarefa tende a ser baixa. A fim de evitar que

HIME possua vantagens devido a este benefício, considera-se valores de α no intervalo $(1,2{,}5]$.

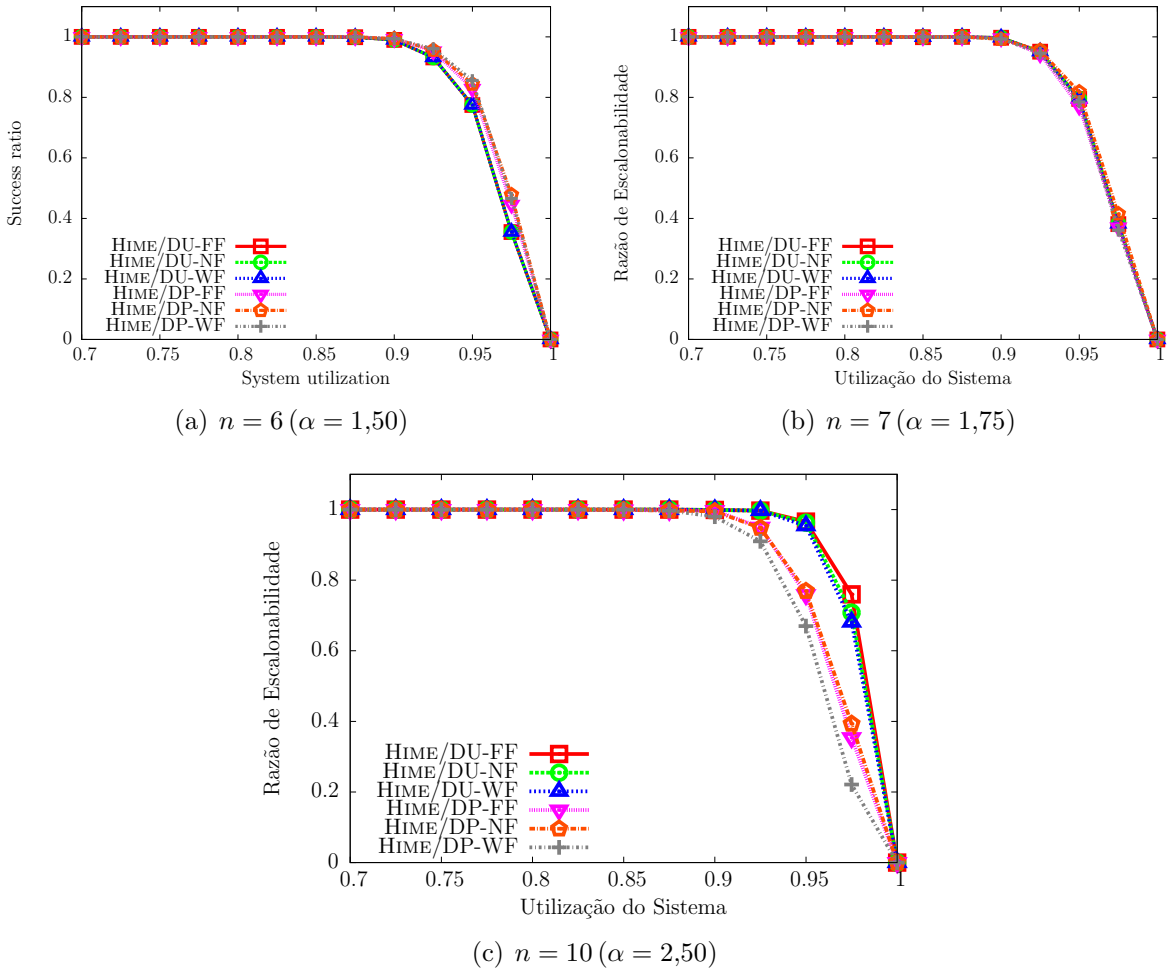
Nas próximas seções são mostrados alguns resultados dos experimentos realizados descrevendo o comportamento geral de HIME. Cada ponto nos gráficos representa uma média de 1.000 conjuntos de tarefas gerados a serem escalonados em $m = 4, 16, 32$ processadores.

5.2 COMPARANDO DIFERENTES VERSÕES DE HIME

Seis versões de HIME foram consideradas. Estas versões foram obtidas através da combinação do critério de ordenação DP e DU com três heurísticas de *bin-packing*, *First-Fit* (FF), *Next-Fit* (NF) e *Worst-Fit* (WF). Como se pode verificar nas Figuras 5.1 – 5.3, uma pequena diferença entre estas versões foi observada. Isto indica certa robustez no mecanismo proposto sobre a escolha da heurística de *bin-packing*, a qual é um resultado das Linhas 6 – 10 do Algoritmo 1. Lembre-se que estas linhas são uma tentativa de adiar a definição de tarefas migratórias. Antes de defini-las, as Linhas 6 – 10 tentam usar a possível capacidade de processamento deixada disponível quando alguma heurística de *bin-packing* é considerada.

5.3 COMPARANDO HIME COM OUTRAS ABORDAGENS DE ESCALONAMENTO

HIME foi comparado com algoritmos de escalonamento que utilizam teste polinomial, esquemas como $C{=}D$ (BURNS et al., 2010), que executam em tempo pseudo-polinomial, não foram considerados. Como NPS-FΩ é conhecido por obter os melhores resultados em termos de escalonabilidade quando os custo de tempo de execução não são considerados (BASTONI; BRANDENBURG; ANDERSON, 2011; BLETSAS; ANDERSSON, 2011), esta abordagem foi tomada como base de comparação. Além disso, embora SPA2 (GUAN et al., 2010) seja um esquema com teste polinomial e que possui baixo custo em tempo de execução, esta estratégia apresenta menor escalonabilidade vinculada, uma vez que um

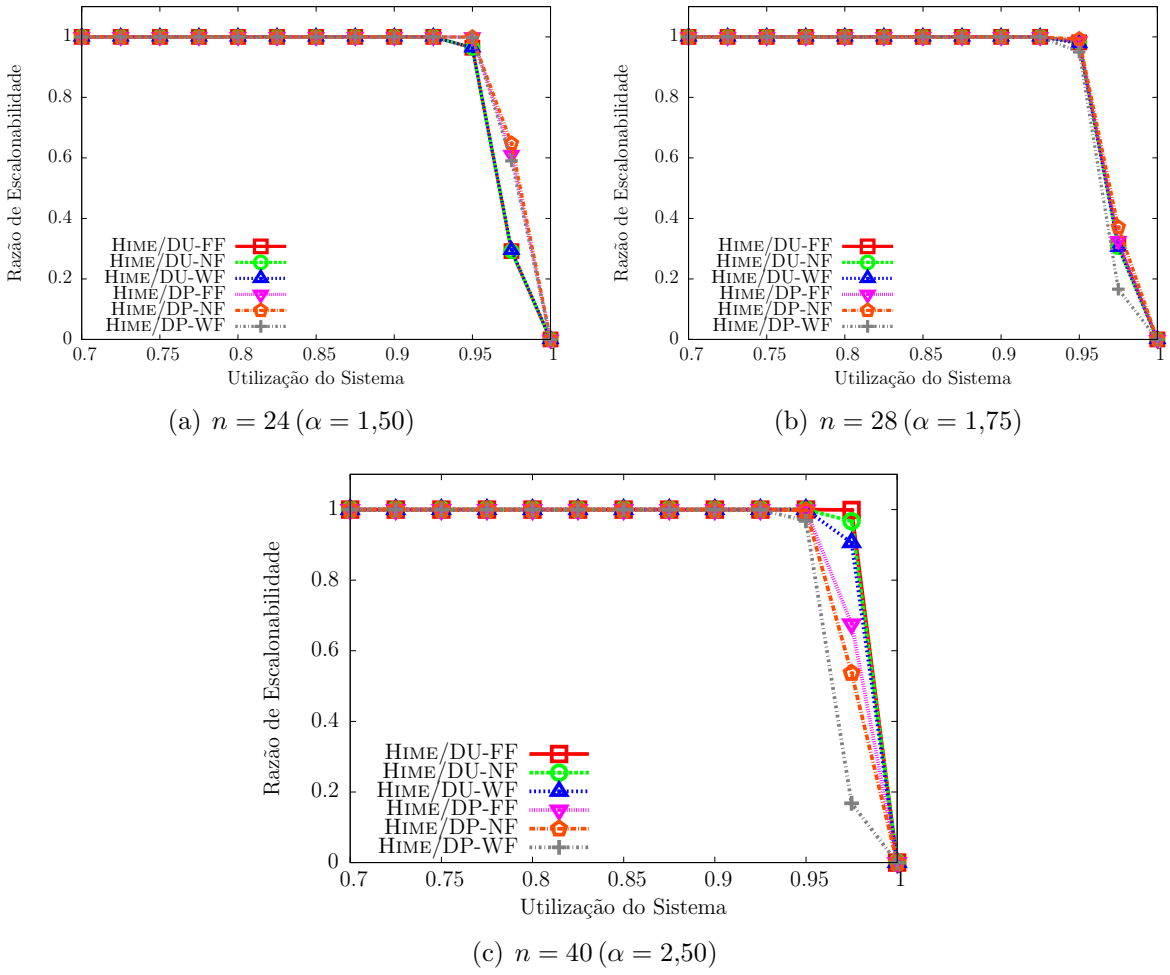


(a) $n = 6\,(\alpha = 1{,}50)$ (b) $n = 7\,(\alpha = 1{,}75)$

(c) $n = 10\,(\alpha = 2{,}50)$

Figura 5.1. Razão de escalonabilidade de seis versões de HIME para $m = 4$ processadores com $\alpha \in [1{,}5, 2{,}5]$

conjunto de tarefas Γ somente é considerado escalonável quando $\frac{U(\Gamma)}{m} \leqslant n(2^{\frac{1}{n}} - 1)$. O EDF particionado empregando a heurística de *bin-packing First-Fit* foi considerado para ilustrar até que ponto as abordagens de escalonamento híbridas melhoraram a escalonabilidade do sistema. Além disso, a estratégia utilizada pelo NPS-FΩ foi somente empregada em nossos experimentos quando o EDF particionado não encontrou uma partição para um determinado conjunto de tarefas, como recomendado por seus autores (BASTONI; BRANDENBURG; ANDERSON, 2011). A ordem de alocação das tarefas para estas duas abordagens segue o critério DU, pois este geralmente resulta em melhor desempenho para NPS-F (BLETSAS; ANDERSSON, 2011). Com base nos resultados do experimento da seção anterior, os quais indicam um comportamento semelhante devido à heurística



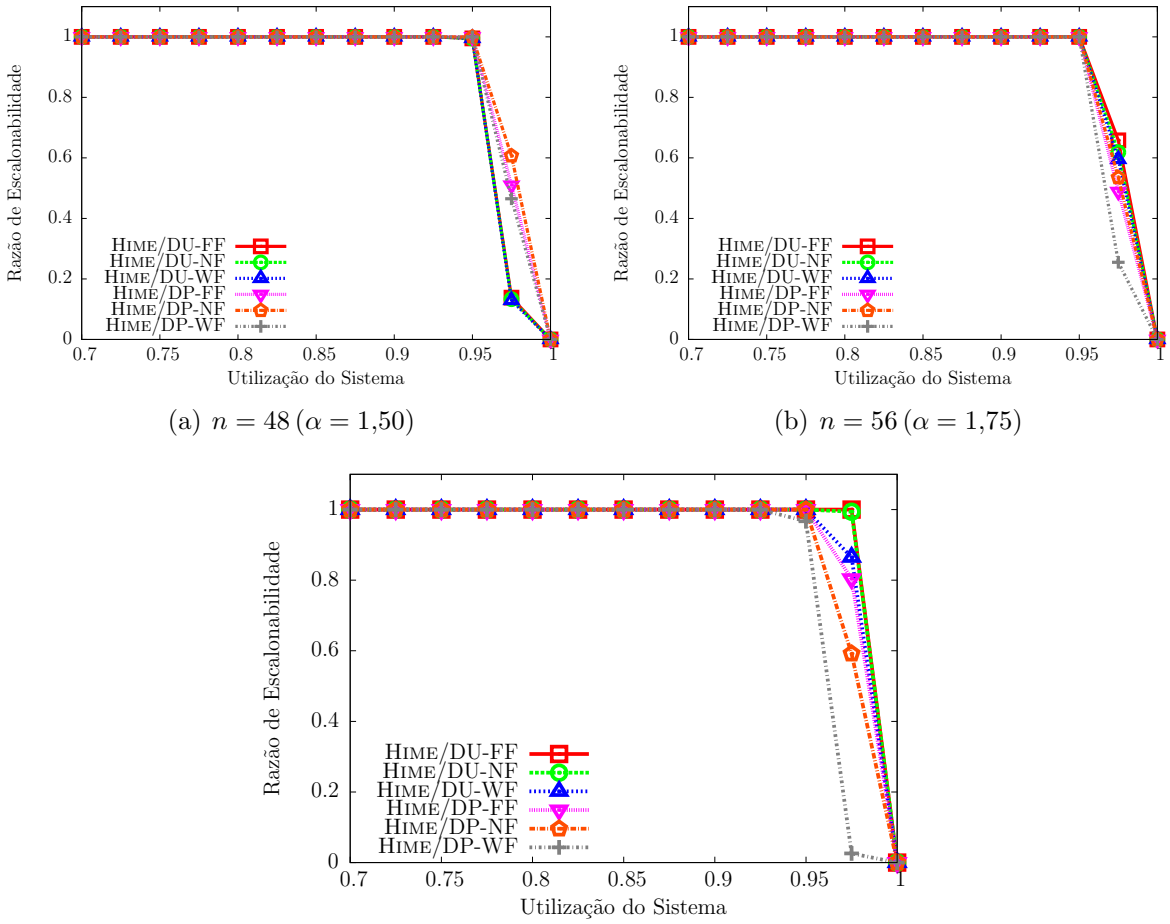
(a) $n = 24\,(\alpha = 1{,}50)$ (b) $n = 28\,(\alpha = 1{,}75)$

(c) $n = 40\,(\alpha = 2{,}50)$

Figura 5.2. Razão de escalonabilidade de seis versões de HIME para $m = 16$ processadores com $\alpha \in [1{,}5, 2{,}5]$

bin-packing para HIME, considera-se neste experimento a versão HIME/DU-FF.

Os custos de migração e preempção do NPS-F dependem do tamanho do intervalo de tempo escolhido para a janela de tempo, que não deve ser maior do que o período mínimo $T_{\min} = \min_{\tau_i \in \Gamma}(T_i)$. Este intervalo é definido como $\frac{T_{\min}}{\delta}$, onde δ é um inteiro positivo. Assim, uma instância de uma tarefa migratória com período T migra pelo menos $\lfloor \frac{T}{\delta} \rfloor$ vezes durante sua execução. Nos experimentos, os valores de δ ficaram entre 1 e 4, conduzindo ao menor limite superior de utilização do sistema de 75% a 90% de m. A grande maioria dos valores de δ não possuem uma boa relação custo-benefício, devido aos custos crescentes de preempção das tarefas. É importante ressaltar que HIME



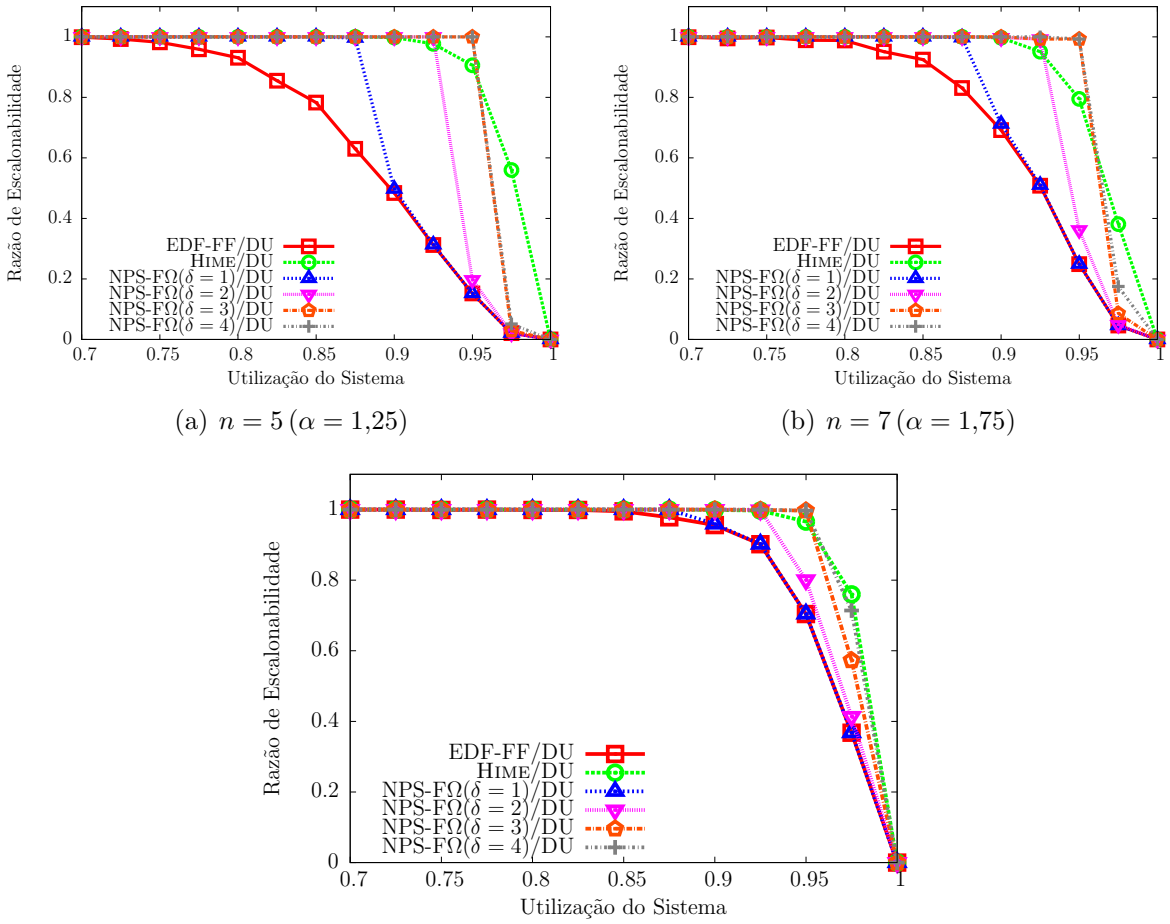
(a) $n = 48\,(\alpha = 1{,}50)$ (b) $n = 56\,(\alpha = 1{,}75)$

(c) $n = 80\,(\alpha = 2{,}50)$

Figura 5.3. Razão de escalonabilidade de seis versões de HIME para $m = 32$ processadores com $\alpha \in [1{,}5, 2{,}5]$

garante que uma instância de uma tarefa migre no máximo $m-1$ vezes e não exista mais do que $\lfloor 0{,}5m \rfloor$ tarefas migratórias. Além disso, o número máximo de preempções e de tarefas migratórias em HIME não são superiores aos encontrados para o NPS-F (BLETSAS; ANDERSSON, 2011). Em outras palavras, as migrações e preempções observadas na abordagem proposta devem ser significativamente inferiores aos encontrados para NPS-F. Este aspecto, contanto, não foi levado em consideração nos experimentos comparativos.

Três critérios para a geração dos conjuntos de tarefas foram considerados: $n = m+1$, $n = 2m-1$ e $n = 2m + \lfloor 0{,}5m \rfloor$. Para todos os critérios, os valores de α pertencem ao intervalo $(1, 2{,}5]$, conforme desejado. O primeiro critério expressa o mínimo número de



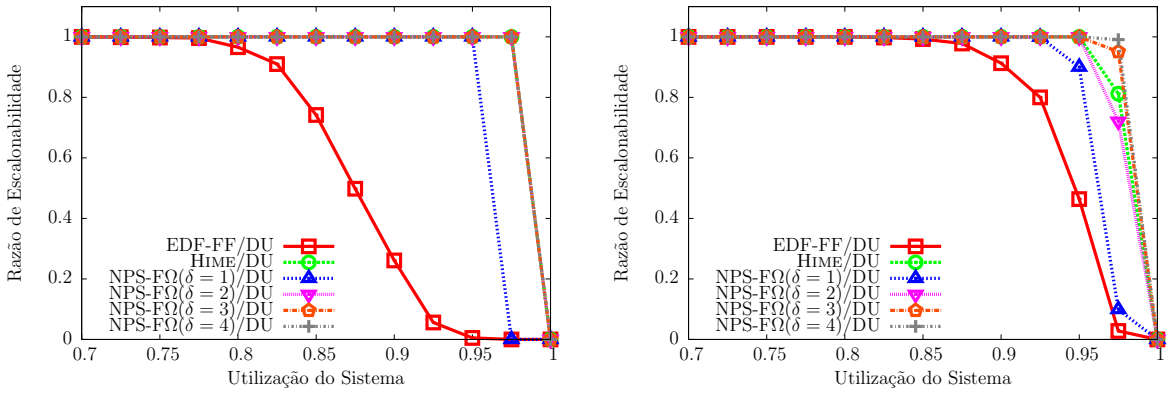
(a) $n = 5\,(\alpha = 1{,}25)$ (b) $n = 7\,(\alpha = 1{,}75)$

(c) $n = 10\,(\alpha = 2{,}50)$

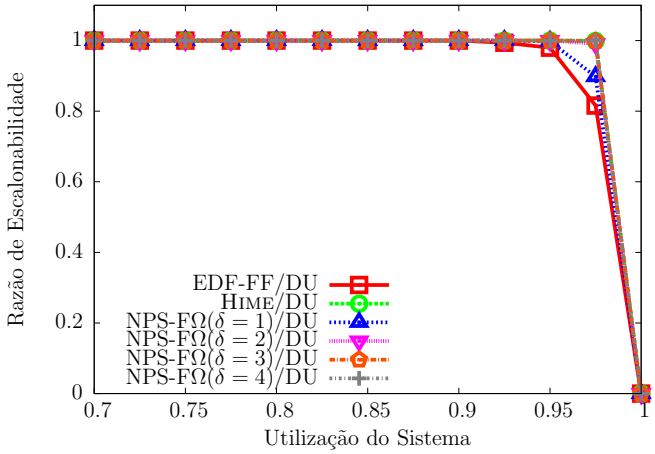
Figura 5.4. Razão de escalonabilidade de HIME, NFS-F e EDF particionado para $m = 4$ processadores com $\alpha \in (1, 2{,}5]$

tarefas que torna possível definir uma tarefa migratória. O segundo representa o mínimo número de tarefas que pode conduzir a existência de $m - 1$ tarefas migratórias, o qual é possível para o NPS-F. E o último critério abrange o cenário com o máximo valor considerado para α.

Alguns dos resultados encontrados nos experimentos são resumidos nos gráficos das Figuras 5.4 – 5.6. Como pode ser observado, HIME apresenta desempenho comparável com os obtidos pelo NPS-FΩ em termos de escalonabilidade. Como os custos em tempo de execução do NPS-F (não considerados neste experimento) são reconhecidamente elevados (BASTONI; BRANDENBURG; ANDERSON, 2011) e as estratégias que utilizam execução



(a) $n = 17\,(\alpha \approx 1{,}06)$ (b) $n = 31\,(\alpha \approx 1{,}94)$

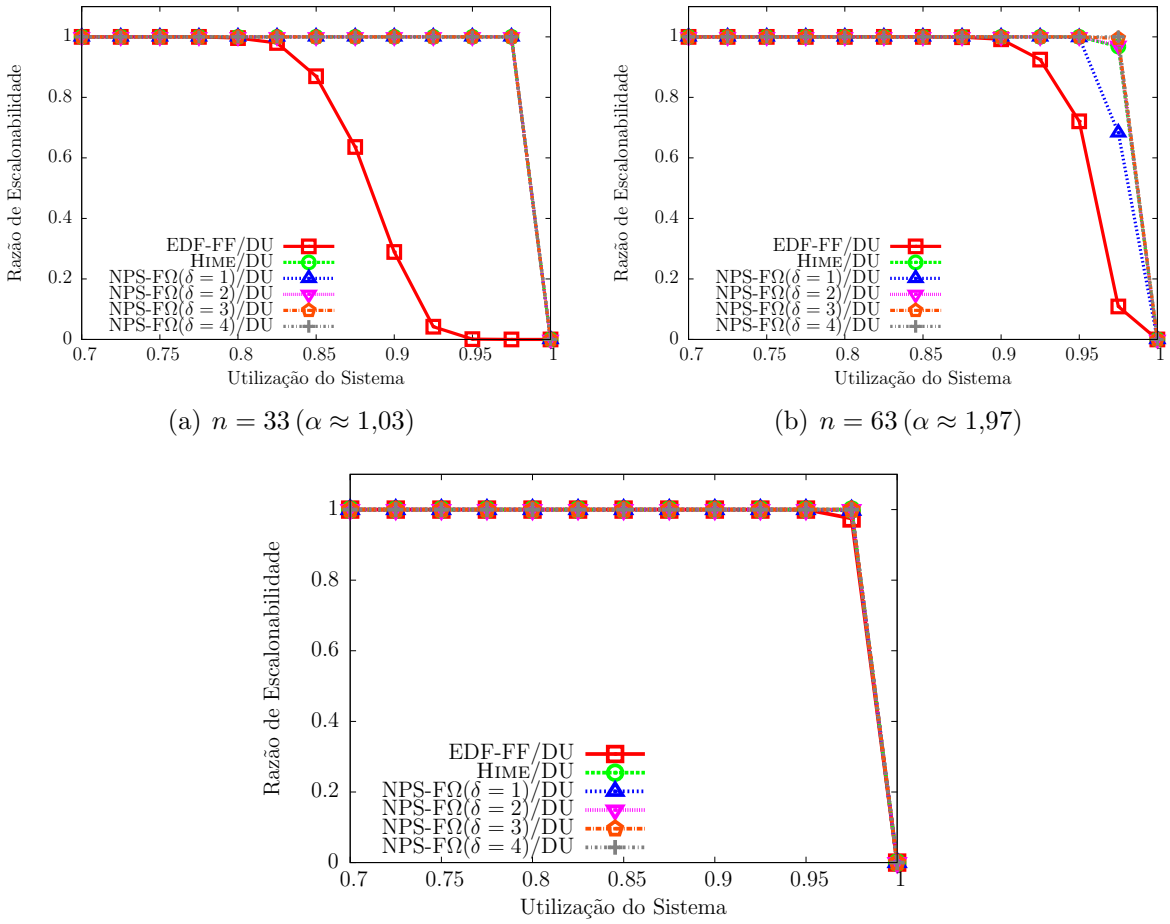


(c) $n = 40\,(\alpha = 2{,}50)$

Figura 5.5. Razão de escalonabilidade de HIME, NFS-F e EDF particionado para $m = 16$ processadores com $\alpha \in (1, 2{,}5]$

de tarefa migratória baseada na instância da tarefa geram menos preempções do que as abordagens que adotam execução de tarefa migratória baseada em janela de tempo (ANDERSSON; PINHO, 2010), esses resultados indicam que HIME obtém uma boa relação custo-benefício.

As Figuras 5.4 – 5.6 também indicam que o desempenho do EDF particionado em termos de escalonabilidade se aproxima dos observados nas outras abordagens para valores maiores de α. Este efeito é esperado, como mencionado anteriormente, pois o sistema tende a ser particionado quando α cresce. Efeito similar também pode ser observado quando m cresce, como as Figuras 5.4(c), 5.5(c) e 5.6(c) mostram. De fato, quanto maior



(a) $n = 33\,(\alpha \approx 1{,}03)$ (b) $n = 63\,(\alpha \approx 1{,}97)$

(c) $n = 80\,(\alpha = 2{,}50)$

Figura 5.6. Razão de escalonabilidade de HIME, NFS-F e EDF particionado para $m = 32$ processadores com $\alpha \in (1, 2{,}5]$

o valor de m, maiores conjuntos de tarefas são gerados nos experimentos. Nestes casos, é mais provável que uma parcela maior das tarefas seja definida como não migratória.

Foram selecionados alguns dados dos experimentos para ilustrar até que ponto HIME gera tarefas migratórias. Os resultados são apresentados nas Tabelas 5.1 – 5.3. Os valores indicados nas tabelas correspondem aos conjuntos de tarefas considerados escalonáveis por HIME para $m = 16$, que foram mostrados na Figura 5.5. A primeira coluna da tabela informa a utilização dos conjuntos de tarefas ($U(\Gamma)$), enquanto que a segunda (P) indica a porcentagem de processadores para os quais nenhuma tarefa migratória foi alocada. Essa métrica fornece uma indicação da parte do sistema que permanece particionada. A

			Processador por tarefas migratória (10^{-4})														
$U(\Gamma)$	P(%)	M(%)	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0,700	100,00	0,00	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0,725	99,99	0,10	0,6	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0,750	99,98	0,20	1,2	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0,775	99,94	0,40	1,2	1,2	–	–	–	–	–	–	–	–	–	–	–	–	–
0,800	99,54	3,40	16,5	3,5	–	–	–	–	–	–	–	–	–	–	–	–	–
0,825	98,67	9,00	33,5	19,4	–	–	–	–	–	–	–	–	–	–	–	–	–
0,850	96,01	25,80	80,0	71,2	0,6	–	–	–	–	–	–	–	–	–	–	–	–
0,875	91,66	50,20	112,9	170,6	11,8	–	–	–	–	–	–	–	–	–	–	–	–
0,900	86,23	73,90	87,7	272,4	70,0	4,1	0,6	–	–	–	–	–	–	–	–	–	–
0,925	79,13	94,40	42,9	241,8	205,3	60,0	5,3	–	–	–	–	–	–	–	–	–	–
0,950	70,14	99,50	6,5	54,7	171,2	211,8	105,3	31,2	2,9	1,8	–	–	–	–	–	–	–
0,975	51,48	100,00	–	–	6,5	32,4	81,2	161,2	134,7	93,5	45,3	20,0	8,2	2,9	1,8	–	0,6

Tabela 5.1. Distribuição das tarefas migratórias para HIME/DU-FF em $m = 16$ e $n = 17$ ($\alpha \approx 1{,}06$)

			Processador por tarefas migratória (10^{-4})								
$U(\Gamma)$	P(%)	M(%)	2	3	4	5	6	7	8	9	10
0,700	100,00	0,00	–	–	–	–	–	–	–	–	–
0,725	100,00	0,00	–	–	–	–	–	–	–	–	–
0,75	100,00	0,00	–	–	–	–	–	–	–	–	–
0,775	100,00	0,00	–	–	–	–	–	–	–	–	–
0,800	100,00	0,00	–	–	–	–	–	–	–	–	–
0,825	99,98	0,20	0,6	–	–	–	–	–	–	–	–
0,850	99,89	0,70	1,9	0,6	–	–	–	–	–	–	–
0,875	99,59	2,20	9,0	1,0	–	–	–	–	–	–	–
0,900	98,53	8,70	33,2	3,2	–	–	–	–	–	–	–
0,925	96,32	20,00	79,0	10,6	–	–	–	–	–	–	–
0,950	86,44	53,60	269,0	52,6	1,0	–	–	–	–	–	–
0,975	55,85	96,55	688,5	245,1	26,2	7,5	2,4	0,4	0,4	0,4	–

Tabela 5.2. Distribuição das tarefas migratórias para HIME/DU-FF em $m = 16$ e $n = 31$ ($\alpha \approx 1{,}94$)

			Processador por tarefas migratória (10^{-4})								
$U(\Gamma)$	P(%)	M(%)	2	3	4	5	6	7	8	9	10
0,700	100,00	0,00	–	–	–	–	–	–	–	–	–
0,725	100,00	0,00	–	–	–	–	–	–	–	–	–
0,750	100,00	0,00	–	–	–	–	–	–	–	–	–
0,775	100,00	0,00	–	–	–	–	–	–	–	–	–
0,800	100,00	0,00	–	–	–	–	–	–	–	–	–
0,825	100,00	0,00	–	–	–	–	–	–	–	–	–
0,850	100,00	0,00	–	–	–	–	–	–	–	–	–
0,875	100,00	0,00	–	–	–	–	–	–	–	–	–
0,900	100,00	0,00	–	–	–	–	–	–	–	–	–
0,925	99,93	0,60	3,5	–	–	–	–	–	–	–	–
0,950	99,64	2,00	15,9	0,6	–	–	–	–	–	–	–
0,975	96,32	18,32	105,4	41,8	1,8	0,6	–	–	–	–	–

Tabela 5.3. Distribuição das tarefas migratórias para HIME/DU-FF em $m = 16$ e $n = 40$ ($\alpha \approx 2{,}5$)

terceira coluna (M) representa a percentagem de conjuntos de tarefas com pelo menos uma tarefa migratória. As demais colunas informam o percentual de tarefas que executam em $k = 2, 3, \ldots, 16$ processadores.

Observa-se que nenhuma tarefa migratória foi gerada para conjuntos de tarefas com utilização inferior a 0,725. O número de tarefas que precisam migrar para tornar o conjunto de tarefas escalonável aumenta em função da sua utilização. A maioria das

tarefas migratórias não necessitam mais do que três processadores, mesmo considerando valores de utilização bastante elevados. Como pode ser visto nas tabelas, menos que 1% das tarefas migratórias foram alocadas para mais do que 9 processadores.

Outra observação interessante é que a maioria dos processadores executa somente tarefas não migratórias. De fato, observou-se nos experimentos que, em média, mais de 80% dos processadores com $U(\Gamma) \leqslant 0{,}90$ se comportam como EDF particionado. Além disso, observe que para $U(\Gamma) = 0{,}975$, em média, pelo menos uma tarefa migratória foi definida para 71,62% dos conjuntos de tarefas gerados. Mesmo para esta situação, o número médio de processadores que executam somente tarefas não migratórias é superior a 51%. Este comportamento evidencia uma característica importante de HIME: tarefas migratórias são definidas apenas quando necessário, de modo a beneficiar a escalonabilidade.

5.4 SUMÁRIO

Neste capítulo foram realizados experimentos que avaliaram algumas versões de HIME/DP e de HIME/DU. A comparação entre estas versões indicou que independente da heurística de *bin-packing* utilizada a grande maioria dos conjuntos de tarefas cumprem seus requisitos temporais, mesmo que estes tenham utilização elevada. Além disso, HIME/DU-FF foi comparado com o EDF-FF/DU e o NPS-FΩ/DU (BLETSAS; ANDERSSON, 2011). Este experimento mostrou que a abordagem proposta possui resultados equiparáveis aos encontrados a uma das abordagens híbridas de escalonamento para STR multiprocessados que possuem melhores resultados teóricos.

CAPÍTULO 6

CONCLUSÃO

Este trabalho descreveu HIME (*HIghest-priority Migration managed by EDF*), uma abordagem híbrida de escalonamento para sistemas de tempo real multiprocessados, capaz de garantir o escalonamento de STR crítico com alta utilização das unidades de processamento. Dependendo do esquema de alocação de tarefas usado, várias versões de HIME podem ser obtidas. Sistemas com utilização não superior a 72,$\bar{2}$% são garantidos para serem escalonados por todas essas versões, enquanto que a utilização vinculada às versões que utilizam o esquema de alocação que adota a ordem decrescente de utilização das tarefas (HIME/DU) podem ser elevada para cerca de 74,87%. Além disso, experimentos, usando uma grande variedade de conjuntos de tarefas gerados aleatoriamente, indicaram que o HIME pode lidar com sistemas com utilização maior que 95%.

Do ponto de vista de implementação, várias características tornam HIME bastante atraente. Nenhum serviço especial, no nível do sistema operacional, é necessário. Apenas o serviço padrão para a gestão de temporização (*timer*), escalonadores locais com EDF e um mecanismo simples de associação de tarefa ao processador (*task affinity service*) são necessários. Além disso, HIME produz um número reduzido de migração por tarefa e os custos devido a preempções em cada processador são compatíveis com as abordagens de escalonamento particionado.

Como continuação deste trabalho, pode-se indicar as seguintes possíveis extensões:

- Incluir outros modelos de sistema, por exemplo, adequar HIME para sistemas com processadores não necessariamente idênticos.
- Rever os testes de escalonabilidade utilizados para alocação das tarefas para considerar tarefas com *deadline* arbitrário.

- Como a infra-estrutura básica exigida por HIME é semelhante àquelas encontradas em STR monoprocessados, questões como o compartilhamento de recursos, relações de precedência, entre outras, possivelmente podem ser tratadas com estratégias similares às adotadas nesses sistemas.

- Uma avaliação experimental é necessária para verificar o desempenho em tempo de execução de HIME, quando comparado com outras abordagens híbridas baseadas no EDF.

Os resultados aqui apresentados devem servir de base para tais trabalhos e para o aprimoramento dos mecanismos de escalonamento para sistemas de tempo real multiprocessados.

REFERÊNCIAS

ANDERSSON, B.; BARUAH, S.; JONSSON, J. Static-Priority Scheduling on Multiprocessors. In: *Proc. 22nd IEEE Real-Time Systems Symposium (RTSS'01)*. [S.l.: s.n.], 2001. p. 193 – 202. ISBN 0-7695-1420-0.

ANDERSSON, B.; BLETSAS, K. Sporadic Multiprocessor Scheduling with Few Preemptions. In: *Proc. of the 20th IEEE Euromicro Conference on Real-Time Systems (ECRTS'08)*. [S.l.: s.n.], 2008. p. 243 – 252. ISSN 1068-3070.

ANDERSSON, B.; BLETSAS, K.; BARUAH, S. Scheduling Arbitrary-Deadline Sporadic Task Systems on Multiprocessors. In: *Proc. of the 29th IEEE Real-Time Systems Symposium (RTSS'08)*. [S.l.: s.n.], 2008. p. 385 – 394. ISSN 1052-8725.

ANDERSSON, B.; JONSSON, J. Preemptive Multiprocessor Scheduling Anomalies'. In: *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*. [S.l.: s.n.], 2002. p. 12 – 19.

ANDERSSON, B.; PINHO, L.-M. Implementing Multicore Real-Time Scheduling Algorithms Based on Task Splitting Using Ada 2012. In: *Reliable Software Technology - Ada-Europe 2010*. [S.l.]: Springer Berlin / Heidelberg, 2010. v. 6106, p. 54 – 67.

ANDERSSON, B.; TOVAR, E.; SOUSA, P. B. *Implementing Slot-Based Task-Splitting Multiprocessor Scheduling*. [S.l.], 2010. Disponível em: <`http://www.cister.isep.ipp.pt/docs/implementing%2Bslot%252Dbased%2Btask%252Dsplitting%2Bmultiprocessor%2Bscheduling/553/attach.pdf`>.

BARUAH, S.; GOOSSENS, J. "Real-Time Scheduling: Algorithms and Complexity". In: LEUNG, J. Y.-T. (Ed.). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. [S.l.]: Chapman & Hall/CRC, 2004. cap. 28.

BARUAH, S.; MOK, A.; ROSIER, L. Preemptively scheduling hard-real-time sporadic tasks on one processor. In: IEEE. *Proc. of the 11th Real-Time Systems Symposium (RTSS'90)*. [S.l.], 1990. p. 182 – 190.

BARUAH, S. K. Fairness in Periodic Real-Time Scheduling. In: *Proc. of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*. [S.l.: s.n.], 1995. p. 200 – 209. ISBN 0-8186-7337-0.

BASTONI, A.; BRANDENBURG, B.; ANDERSON, J. Is Semi-Partitioned Scheduling Practical? In: *Proc. of the 23rd IEEE Euromicro Conference on Real-Time Systems (ECRTS'11)*. [S.l.: s.n.], 2011. p. 125 – 135.

BERNSTEIN, P. A.; HADZILACOS, V.; GOODMAN, N. *Concurrency Control and Recovery in Database Systems*. [S.l.]: Addison-Wesley, 1987.

BINI, E.; BUTTAZZO, G. C. Measuring the Performance of Schedulability Tests. *Real-Time Systems*, Springer Netherlands, v. 30, p. 129 – 154, 2005. ISSN 0922-6443.

BINI, E.; BUTTAZZO, G. C.; BUTTAZZO, G. M. Rate Monotonic Analysis: the Hyperbolic Bound. *IEEE Transactions on Computers*, v. 52, n. 7, p. 933 – 942, 2003. ISSN 0018-9340.

BLETSAS, K.; ANDERSSON, B. Preemption-Light Multiprocessor Scheduling of Sporadic Tasks with High Utilisation Bound. *Real-Time Systems*, Springer Netherlands, v. 47, p. 319 – 355, 2011. ISSN 0922-6443.

BURNS, A. et al. Partitioned EDF Scheduling for Multiprocessors Using a C=D Scheme. In: *Proc. of the 18th IEEE Conference on Real-Time and Network Systems (RTNS'10)*. [S.l.: s.n.], 2010. p. 169 – 178.

BUTTAZZO, G. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. [S.l.]: Kluwer Academic Publishers, 1997.

CARPENTER, J. et al. A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms. In: *Handbook on Scheduling Algorithms, Methods, and Models*. [S.l.]: Chapman Hall/CRC, Boca, 2004.

CIRINEI, M.; BAKER, T. P. EDZL Scheduling Analysis. In: *Proc. of the 19th IEEE Euromicro Conference on Real-Time Systems (ECRTS'07)*. [S.l.: s.n.], 2007. p. 9 – 18. ISSN 1068-3070.

COFFMAN-JR., E. G.; GAREY, M. R.; JOHNSON, D. S. Approximation Algorithms for Bin Packing: A Survey. *Approximation algorithms for NP-hard problems*, p. 46 – 93, 1997.

DAVIS, R. I.; BURNS, A. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems. *ACM Computing Survey*, ACM, v. 43, p. 35:1 – 35:44, 2011. ISSN 0360-0300.

DHALL, S. K.; LIU, C. L. "On a Real-Time Scheduling Problem". *Operations Research*, v. 26, n. 1, p. 127–140, 1978.

EMBERSON, P. *TaskGen: Python Script for Generating Task-Sets for Multi-Processor Systems with Roger Stafford's Randfixedsum Algorithm*. 2010. Disponível em: <`http://retis.sssup.it/waters2010/data/taskgen-0.1.tar.gz`>.

EMBERSON, P.; STAFFORD, R.; DAVIS, R. Techniques for the Synthesis of Multiprocessor Tasksets. In: *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS'10)*. [S.l.: s.n.], 2010. p. 6 – 11.

FARINES, J.; FRAGA, J.; OLIVEIRA, R. *Sistemas de Tempo Real*. 1ª. [S.l.]: IME-USP, 2000.

FUNK, S.; GOOSSENS, J.; BARUAH, S. On-Line Scheduling on Uniform Multiprocessors. In: *Proc. 22nd IEEE Real-Time Systems Symposium (RTSS'01)*. [S.l.: s.n.], 2001. p. 183 – 192. ISBN 0-7695-1420-0.

FUNK, S.; NADADUR, V. LRE-TL: An Optimal Multiprocessor Algorithm for Sporadic Task Sets. In: *Proc. of 17th IEEE Conference on Real-Time and Network Systems (RTNS'09)*. [S.l.: s.n.], 2009. p. 159 – 168.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979. ISBN 0716710455.

GEORGE, L.; COURBIN, P.; SOREL, Y. Job vs. Portioned Partitioning for the Earliest Deadline First Semi-Partitioned Scheduling. *Journal of Systems Architecture*, v. 57, n. 5, p. 518 – 535, 2011. ISSN 1383-7621.

GUAN, N. et al. Fixed-Priority Multiprocessor Scheduling with Liu and Layland's Utilization Bound. In: *Proc. of the 16th IEEE Real Time Technology and Applications Symposium (RTAS'10)*. [S.l.: s.n.], 2010. p. 165 – 174.

HELD, J.; BAUTISTA, J.; KOEHL, S. *From a Few Cores to Many: A Tera-scale Computing Research Overview*. [S.l.], 2006. Disponível em: <`ftp://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf`>.

INTEL. *Intel Reveals Details of Next-Generation High-Performance Computing Platforms*. 2011. `www.intel.com/newsroom/sc11`. Acessado: 17/11/2011.

JENSEN, J. Sur les Fonctions Convexes et les Inégalités entre les Valeurs Moyennes. *Acta Mathematica*, Springer Netherlands, v. 30, p. 175–193, 1906. ISSN 0001-5962.

KATO, S.; YAMASAKI, N.; ISHIKAWA, Y. Semi-partitioned Scheduling of Sporadic Task Systems on Multiprocessors. In: *Proc. of the 21st IEEE Euromicro Conference on Real-Time Systems (ECRTS'09)*. [S.l.: s.n.], 2009. p. 249 – 258. ISSN 1068-3070.

LEVIN, G. et al. DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling. In: *Proc. of the 22nd IEEE Euromicro Conference on Real-Time Systems (ECRTS'10)*. [S.l.: s.n.], 2010. p. 3 – 13.

LIU, C. L.; LAYLAND, J. W. Scheduling Algorithms for Multiprogram in a Hard Real-Time Environment. *Journal of ACM*, v. 20, n. 1, p. 46 – 61, 1973.

MACÊDO, R. J. A. et al. Tratando a Previsibilidade em Sistemas de Tempo-real Distribuídos: Especificação. Linguagens, Middleware e Mecanismos Básicos. . In: *Capítulo 3 do Livro texto para o mini-curso apresentado no XXII Simpósio Brasileiro de Redes de Computadore (SBRC'2004)*. [S.l.: s.n.], 2004. p. 105 – 163. ISBN 85-88442-82-5.

MASSA, E.; LIMA, G. A Bandwidth Reservation Strategy for Multiprocessor Real-Time Scheduling. In: *Proc. of the 16th IEEE Real Time Technology and Applications Symposium (RTAS'10)*. [S.l.: s.n.], 2010. p. 175 – 183.

MCNAUGHTON, R. Scheduling with Deadlines and Loss Functions. *Management Science*, JSTOR, v. 6, n. 1, p. 1 – 12, 1959.

MERRITT, R. *CPU Designers Debate Multi-core Future*. Junho 2008. `http://www.eetimes.com/showArticle.jhtml?articleID=206105179`.

PALENCIA, J. C.; HARBOUR, M. G. "Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems". In: *Proc. of the 20th Real-Time Systems Symposium (RTSS'99)*. [S.l.]: IEEE Computer Society Press, 1999. p. 328–339.

REGNIER, P. et al. RUN: Optimal Multiprocessor Real-time Scheduling via Reduction to Uniprocessor. In: *Proc. of the 32nd IEEE Real-Time Systems Symposium (RTSS'11)*. [S.l.: s.n.], 2011. p. 186 – 195.

SHA, L. et al. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Systems*, Kluwer Academic Publishers, Norwell, MA, USA, v. 28, n. 2-3, p. 101 – 155, 2004. ISSN 0922-6443.

STANKOVIC, J. A. et al. *Deadline Scheduling For Real-Time Systems EDF and Related Algorithms*. 1st. [S.l.]: Kluwer Academic Publishers, 1998.

TILERA. *TILE-Gx Processor Family*. 2011. `http://www.tilera.com/products/processors/TILE-Gx_Family`. Acessado: 17/11/2011.

TINDELL, K.; BURNS, A.; WELLINGS, A. J. "Analysis of Hard Real-Time Communications". *Real-Time Systems*, v. 9, n. 2, p. 147 – 171, 1995.

TINDELL, K.; CLARK, J. "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems". *Microprocessing and Microprogramming, Euromicro Journal*, v. 40, p. 117 – 134, 1994.

TERMO DE AUTORIZAÇÃO PARA PUBLICAÇÃO DIGITAL NA BIBLIOTECA DIGITAL DA UFBA

1 Identificação do tipo de documento

Tese [ ] Dissertação [X] Monografia [ ] Trabalho de Conclusão de Curso [ ]

2 Identificação do autor e do documento

Nome completo: JOSÉ AUGUSTO MATOS SANTOS JÚNIOR

CPF: 010.936.035-42

Telefone: (071) 32340792 e-mail: JAMJUNIOR@UFBA.BR

Programa/Curso de Pós-Graduação/Graduação/Especialização: _____

Título do documento: ESCALONAMENTO EM SISTEMAS DE TEMPO REAL MULTIPROCESSADOS COM BAIXO CUSTO DE IMPLEMENTAÇÃO Data da defesa: 15 /03 /12

3 Autorização para publicação na Biblioteca Digital da UFBA

Autorizo com base no disposto na Lei Federal nº 9.610, de 19 de fevereiro de 1998 e na Lei n o 10.973, de 2 de dezembro de 2004, a Universidade Federal da Bahia (UFBA) disponibilizar gratuitamente sem ressarcimento dos direitos autorais, o documento supracitado, de minha autoria, na Biblioteca Digital da UFBA para fins de leitura e/ou impressão pela Internet a título de divulgação da produção científica gerada pela Universidade.

Texto completo [X] Texto parcial [ ]

Em caso de autorização parcial, especifique a (s) parte(s) do texto que deverão ser disponibilizadas:

_____

_____

SALVADOR , 13/04/12 José Augusto M. Santos Jr
Local Data Assinatura do (a) autor (a) ou seu representante legal

4 Restrições de acesso ao documento

Documento confidencial? [X] Não

[ ] Sim Justifique: _____

Informe a data a partir da qual poderá ser disponibilizado na Biblioteca Digital da UFBA:

15 /03/2013 [ ] Sem previsão

Assinatura do Orientador: _____ (Opcional)

O documento está sujeito ao registro de patente? Não [X]
Sim [ ]

O documento pode vir a ser publicado como livro? Sim [ ]
Não [X]

Preencher em três vias. A primeira via deste formulário deve ser encaminhada ao Sistema de Bibliotecas da UFBA/Biblioteca Central; a segunda deve ser enviada para a Biblioteca de sua Unidade, juntamente com o arquivo contendo o documento; a terceira via deve permanecer no Programa de Pós-Graduação para o registro do certificado de conclusão do Curso.

Universidade Federal da Bahia
Sistema de Biblioteca da UFBA
Grupo Técnico da Biblioteca Digital da UFBA

CADASTRO DE INFORMAÇÕES PARA PUBLICAÇÃO DIGITAL
NA BIBLIOTECA DIGITAL DA UFBA

1. **Identificação do tipo de material**
Tese ( ) Dissertação (X) Monografia ( ) Trabalho de Conclusão de Curso ( )
2 . Colegiado do Curso de Pós-Graduação:

Título: ESCALONAMENTO EM SISTEMAS DE TEMPO REAL MULTIPROCESSADOS	
COM BAIXO CUSTO DE IMPLEMENTAÇÃO	
Autor(a): JOSÉ AUGUSTO MATOS SANTOS JÚNIOR	
CPF: 010.936.035-42	*E-mail:* JAMJUNIOR@UFBA.BR
Orientador(a): :	
Nome: GEORGE MARCONI DE ARAÚJO LIMA	
CPF: 465.302.745-53	*E-mail* GMLIMA@UFBA.BR
Co-Orientadores	
Nome:	
CPF:	*E-mail*

Membros da Banca	
Nome: GEORGE MARCONI DE ARAÚJO LIMA	
CPF: 465.302.745-53	*E-mail* GMLIMA@UFBA.BR
Nome: EDUARDO CAMPONOGARA	
CPF: 513.988.180-20	*E-mail* CAMPONOG@DAS.UFSC.BR
Nome: RÔMULO SILVA DE OLIVEIRA	
CPF: 407.617.200-06	*E-mail* ROMULO@DAS.UFSC.BR
Nome:	
CPF:	*E-mail*

Data de Homologação Pós Graduação: 27/04/2012
Financiadores: FAPESB
Data: 13/04/2012
Assinatura: José Augusto M. Santos Jr.

Salvador, 13/04/2012

DECLARAÇÃO

Declaro para os devidos fins que o texto apresentado na dissertação apresentada para a conclusão do meu curso de Mestrado em Mecatrônica da Universidade Federal da Bahia é de minha autoria e que quaisquer informações utilizadas neste texto que tenha proveniente de outros trabalhos tem fonte claramente expressa e, quando for o caso, devidamente autorizada.

José Augusto M. Santos Jr.

Nome: José Augusto Matos Santos Jr.
CPF: 01093603542