UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ESCALONAMENTO DE TAREFAS E FLUXOS DE COMUNICAÇÃO PARA SISTEMAS SEMI-PARTICIONADOS EM ARQUITETURAS NOC

DISSERTAÇÃO DE MESTRADO

Iaê Santos Bonilha

Santa Maria, RS, Brasil

2014

ESCALONAMENTO DE TAREFAS E FLUXOS DE COMUNICAÇÃO PARA SISTEMAS SEMI-PARTICIONADOS EM ARQUITETURAS NOC

por

Iaê Santos Bonilha

Dissertação apresentada ao Curso de Mestrado do Programa de Pós-Graduação em Informática, Área de Concentração em Computação Aplicada, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de Mestre em Informática.

Orientador: Prof. Osmar Marchi dos Santos

Santa Maria, RS, Brasil

2014

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada, aprova a
Dissertação de Mestrado

ESCALONAMENTO DE TAREFAS E FLUXOS DE COMUNICAÇÃO EM UM MODELO DE ESCALONAMENTO SEMI-PARTICIONADO

elaborado por
Iaê Santos Bonilha

como requisito parcial para obtenção do grau de
Mestre em Ciências da Computação

COMISSÃO EXAMINADORA:

Prof. Dr. Osmar Marchi dos Santos (UFSM)
(Presidente/Orientador)

Prof. Dr. Alice de Jesus Kozakevicius (UFSM)

Prof. Dr. Thais Christina Webber dos Santos (PUCRS)

Santa Maria, 24 de Março de 2014.

RESUMO

Dissertação de Mestrado

Mestrado em Informática

Universidade Federal de Santa Maria

ESCALONAMENTO DE TAREFAS E FLUXOS DE COMUNICAÇÃO PARA SISTEMAS SEMI-PARTICIONADOS EM ARQUITETURAS NOC

Autor: Iaê Santos Bonilha

Orientador: Osmar Marchi dos Santos

Com a popularização de sistemas multi-processador, surgiu uma série de propostas de modelos de escalonamento, na área de sistemas de tempo real que, teoricamente, são capazes de obter um alto aproveitamento dos recursos do sistema. Entretanto, o modelo de escalonamento mais adotado continua sendo um dos primeiros modelos de escalonamento propostos na área, o modelo de escalonamento particionado. O modelo de escalonamento particionado só pode garantir o escalonamento de conjuntos com até cerca de 69% de utilização de processador, sendo limitado se comparado com garantias de escalonamento de até 97% de utilização de modelos mais recentes. O motivo pelo qual o escalonamento particionado continua sendo utilizado é a grande concentração de estudos a respeito do modelo e o desenvolvimento de análises de escalonamento capazes de garantir o escalonamento do modelo em condições reais do sistema. Modelos mais recentes, como o escalonamento semi-particionado, apresentam uma possibilidade de um maior aproveitamento do sistema, porém, ainda possuem estudos limitados e não dispõe de análises de escalonamento capazes de prover garantias temporais para o sistema em condições reais, devido à presença de diversas abstrações no modelo. Neste sentido, este trabalho foca em arquiteturas *Network-on-Chip* que apresentam comunicação explícita, abstraída nos trabalhos encontrados na literatura. Este trabalho tem como objetivo primário o desenvolvimento de uma análise de escalonamento capaz de prover garantias temporais para o modelo de escalonamento semi-particionado levando em consideração fatores previamente abstraídos, como a necessidade de comunicação entre tarefas e o impacto da migração das tarefas nos seus fluxos de comunicação, aproximando o modelo da realidade. O desenvolvimento de tal análise possibilita o estudo preliminar de algoritmos heurísticos de mapeamento de tarefas, capazes de mapear conjuntos de tarefas levando em consideração migrações de tarefas e comunicação entre tarefas em um modelo de escalonamento semi-particionado.

Palavras-chave: sistemas de tempo real, escalonamento semi-particionado, sistemas multiprocessados, análise de escalonamento, migração de tarefas, alocação de recursos

ABSTRACT

Masters Dissertation

Master in Iformatics

Federal University of Santa Maria

SEMI-PARTITIONED SCHEDULING OF TASKS AND COMMUNICATION FLOWS ON NOC ARCHTECTURES

Author: Iaê Santos Bonilha

Advisor: Osmar Marchi dos Santos

Despiste the fact that many scheduling models teoretically capable of high system resource utilization were proposed with the development of the real-time system, the industry still uses the first scheduling model proposed for multi-processor real-time systems, the partitioned scheduling model. This scheduling model can guarantee scheduling of task sets up to around 69% processor utilization, which falls pale in comparison to recent scheduling models that can guarantee scheduling up to 97% processor utilization. The motive behind the utilization of the partitioned scheduling as industrial model is the amount of studies made on this model and the development of scheduling analysis capable of providing temporal guarantees for this model on a real system environment. Recent scheduling models, like semi-partitioned scheduling, offer the possibility of a higher system resource utilization, it still lack studies and scheduling analysis capable of provide temporal guarantees under a real environment. The current scheduling analysis for most of the more recent models take advantage of a series of abstractions, failing to provide guarantees under real circumstances. This papers primary objective is to produce a new scheduling analysis for semi-partitioned scheduling, capable of achieving temporal guarantees taking some of the previously abstracted factors, like task communication and the impact f task migration on its communications flows, approximating the scheduling model to real environmental conditions. With the development of such analysis preliminary studies were made on heuristic task mapping algorithms for semi-partitioned systems.

Key words: real-time systems, semi-partioned scheduling, multiprocessor systems, scheduling analysis, task migration, resource allocation

Sumário

Índice de Figuras

Índice de Gráficos

1. Introdução

Sistemas de tempo real são sistemas que possuem um caráter temporal crítico, de forma que cada uma de suas tarefas possui um intervalo (ou prazo), no qual seus resultados terão validade. Em outras palavras, se uma das tarefas executar além do seu prazo, seus resultados terão pouca ou nenhuma utilidade para o sistema.

Para a obtenção da previsibilidade temporal de um conjunto de tarefas, é necessário que estas tarefas tenham um comportamento conhecido em termos de tempo de execução, frequência de execução e ordem de execução em relação a outras tarefas. O pior tempo de execução possível para cada tarefa (computação - C) e o tempo mínimo entre liberações consecutivas de uma tarefa (período – T) são obtidos através de simulações ou do estudo aprofundado do conjunto de tarefas junto com a arquitetura do sistema. O processo de determinação da ordem no qual as tarefas são executadas é conhecido como escalonamento de tarefas e, normalmente, é guiado por um sistema de prioridades fixas (imutáveis durante a execução de uma tarefa) ou dinâmicas (sendo modificadas no decorrer da execução de uma tarefa). As pesquisas a respeito de escalonamento de tarefas para sistemas de tempo real tiveram inicio nos anos 70, sendo realizadas inicialmente para sistemas uniprocessados (arquitetura mais difundida na época). No decorrer das pesquisas em escalonamento de tempo real para sistemas uniprocessados, foram desenvolvidos algoritmos de escalonamento considerados ótimos, como os propostos em (Liu e Layland, 1973).

Com o avanço do hardware e o barateamento de seus custos, sistemas multiprocessados se tornaram cada vez mais comuns, gerando problemas na área de sistemas de tempo real, pois os algoritmos considerados ótimos para escalonamento uniprocessado mostraram-se não-ótimos para o escalonamento de sistemas multiprocessados. Isso ocorre porque a distribuição das tarefas entre os múltiplos processadores (fator não abordado por algoritmos não processados por se tratar de um único processador) provou-se um fator impactante na escalonabilidade de um sistema.

Com a necessidade crescente da utilização de arquiteturas multiprocessadas para sistemas de tempo real, foram criadas estratégias de escalonamento para contornar o problema da distribuição de tarefas entre os processadores em sistemas multiprocessados, como mostrado em (Davis e Burns, 2010).

As primeiras estratégias de escalonamento criadas, conhecidas como escalonamento particionado e escalonamento global, apresentavam capacidade da obtenção de garantias temporais de tarefas, porém, às custas de um baixo aproveitamento dos recursos do sistema. No escalonamento particionado as tarefas são distribuídas entre os processadores do sistema de acordo com alguma heurística e sua migração não é permitida. Enquanto isso, no escalonamento global, apesar da distribuição inicial das tarefas, sua migração é permitida. O limite de utilização para garantia de sua escalonabilidade em sistemas particionados beira os 69%. O escalonamento global provê escalonamento a sistemas com até 100% de utilização, porém, gerando um *overhead* intenso no sistema que torna seu uso praticamente proibitivo.

Em meados do ano 2000, começou-se a discutir estratégias de escalonamento híbridas que se aproveitavam das vantagens de ambas estratégias tradicionais, tentando minimizar suas desvantagens. Uma das estratégias híbridas que surgiram nesse período foi o escalonamento

semi-particionado. No escalonamento semi-particionado, as tarefas são distribuídas entre os processadores do sistema de forma similar ao escalonamento particionado, porém, para um conjunto limitado dessas tarefas é permitida a migração. A migração no escalonamento semi-particionado é feita de acordo com padrões fixos, as tarefas migratórias migram sempre entre o mesmo conjunto de processadores, na mesma ordem. Através das migrações em padrões fixos de um sub-conjunto fixo de tarefas, os algoritmos de escalonamento semi-particionado são capazes de gerar um maior aproveitamento de recursos, chegando a cerca de 97% com *overhead* de sistema aceitável (Kato, Yamasaki e Ishikawa, 2009), enquanto geram o comportamento temporal previsível característico do escalonamento particionado, podendo usufruir de suas análises de escalonamento.

Outro fator de grande impacto que deve ser considerado no escalonamento de um conjunto de tarefas é a necessidade de comunicação entre elas. Relações de dependência de dados entre tarefas são ocorrências comuns nos sistemas atuais (por exemplo, relações de produtor-consumidor). A comunicação entre tarefas, além de adicionar complexidade ao modelo do sistema, pode impactar diretamente na eficiência do mesmo. Isso ocorre devido ao fato de que tarefas com relações de dependência de dados necessitam de dados provenientes de outras tarefas (que podem estar até mesmo alocadas em um processador diferente) para iniciar sua execução e o atraso no envio (perda de prazo por parte da tarefa geradora dos dados) ou na propagação desses dados pode resultar em perdas de prazos por parte da tarefa receptora.

Em sistemas multiprocessados o problema de roteamento dos dados é acrescentado, pois a distância entre o core da tarefa geradora e da tarefa receptora afeta o tempo de propagação final dos dados. Nesta dissertação utiliza-se uma arquitetura baseada em *Network-on-Chip* (*NoC*), pois tal arquitetura gera padrões previsíveis de roteamento de comunicação. Na literatura atual, existem modelos de análise de escalonamento de comunicações para sistemas particionados considerando arquiteturas *Network-on-Chip* (Shi e Burns, 2010), que tratam isoladamente do escalonamento de comunicações e não lidam com o escalonamento de tarefas em conjunto.

Apesar de apresentar uma grande melhora no aproveitamento do uso de processadores gerada pelo escalonamento semi-particionado, na literatura atual, não existem estudos a respeito de análises de escalonamento de comunicações para esta estratégia de escalonamento (e estratégias que permitam a migração de tarefas em geral). Sendo o comportamento temporal das comunicações um fator de grande impacto para a escalonabilidade geral do sistema (pois pode resultar em perdas de prazos de tarefas dependentes), a aplicabilidade desta estratégia de escalonamento torna-se questionável. Para que uma estratégia de escalonamento possa ser aplicada nos sistemas atuais, o comportamento temporal das suas comunicações deve ser levado em consideração. Portanto, para tornar o escalonamento semi-particionado viável para sistemas atuais, é necessário um método de prover garantias temporais para comunicações em um ambiente com tarefas migratórias, algo não encontrado na literatura atual.

1.1 Objetivos

Este trabalho tem como principal objetivo, obter o aumento do aproveitamento do poder computacional de sistemas de tempo real multiprocessados através da viabilização do escalonamento semi-particionado para sistemas com necessidade de comunicação entre tarefas considerando arquiteturas *NoC*. Para alcançar este objetivo principal, os seguintes objetivos foram desenvolvidos nesta dissertação:

1) Redução do pessimismo desnecessário presente na análise de escalonamento de comunicações proposta em (Shi e Burns, 2010)
2) Desenvolvimento de um modelo de análise de escalonamento que leve em consideração o impacto do tempo de execução das tarefas no comportamento temporal dos fluxos de comunicação produzidos pelas mesmas.
3) Adaptação do modelo de análise de escalonamento desenvolvido para que o impacto de mudanças de roteamento geradas por migração de tarefas seja levado em conta.
4) Utilização de algoritmos de busca de solução juntamente com o modelo de análise desenvolvido para realização de testes de mapeamento de tarefas automático em um sistema de tempo real semi-particionado.

1.2 Estrutura do Texto

Este trabalho é estruturado da seguinte forma. No Capítulo 2, é apresentada uma revisão da literatura utilizada para o desenvolvimento deste trabalho, abordando o escalonamento de tarefas e fluxos de comunicação em sistemas multiprocessados. No Capítulo 3, é exposto o processo de criação de uma nova análise de escalonamento de fluxos de comunicação baseada na análise proposta em (Shi e Burns, 2010). O Capítulo 4 apresenta a discussão do impacto da migração de tarefas na previsibilidade da análise de escalonamento de fluxos de comunicação, levando a uma proposta de um modelo de análise de escalonamento para obtenção de garantias temporais para fluxos em sistemas com escalonamento semi-particionado. No Capítulo 5, é abordado o processo de mapeamento de tarefas em sistemas semi-particionados, considerando-se o comportamento temporal de fluxos de comunicação e tarefas. Finalmente, no Capítulo 6, são apresentadas as considerações finais e prospectos em relação ao trabalho desenvolvido.

2. Referencial Teórico

Este capítulo contém a base teórica na qual este trabalho foi fundamentado. São expostas neste capítulo as bases teóricas para o desenvolvimento da análise de escalonamento proposta neste trabalho juntamente com uma discussão a respeito das estratégias de escalonamento presentes na literatura atual e a viabilidade da obtenção de garantias temporais de comunicação para cada uma delas.

2.1 Escalonamento de Tarefas

De modo a oferecer garantias temporais para as tarefas de um sistema Liu e Layland introduziram a noção de instante crítico (Liu e Layland, 1973) juntamente com o modelo de taxa monotônica, no qual cada tarefa receberá uma prioridade única e inversamente proporcional ao seu período. Dado um conjunto de tarefas com prioridades arbitrárias, o instante crítico de uma tarefa ocorre quando a mesma chegar ao escalonador simultaneamente com todas as tarefas que possuem prioridade igual ou maior que a sua. O instante crítico é considerado a pior situação possível para o escalonamento. Se existir uma ordem para as tarefas serem executadas na qual nenhuma das tarefas perderá seu prazo, o sistema é considerado escalonável.

Ainda no mesmo artigo, Liu e Layland definiram o primeiro modelo para a análise (ou teste) de escalonamento de um sistema, em que um conjunto de tarefas, chegando simultaneamente no escalonador, com períodos definidos e priorizadas de acordo com o tamanho de seu período (definição de prioridades por taxa monotônica ou *Rate Monotonic*) é praticável, sem que nenhuma delas perca seu prazo, desde que a utilização de CPU não passe do limite dado pela Equação (1).

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \qquad (1)$$

Sendo *Ci* o tempo de computação da tarefa *i*; *Ti* é o período da tarefa *i*; e *n* o número total de tarefas no sistema.

Para que a garantia temporal provida pela Equação (1) se confirmasse, as seguintes restrições deviam ser obedecidas:

- As tarefas devem ser independentes umas das outras;
- Todas as tarefas devem ser periódicas;
- Nenhuma tarefa pode sofrer bloqueio devido a eventos externos;
- Todas as tarefas devem chegar simultaneamente ao escalonador;
- Os prazos das tarefas devem ser iguais aos seus períodos.

O teste de escalonamento dado pela Equação (1) é dito suficiente (se o sistema passar no teste, o sistema é escalonável), porém, não necessário (se o sistema falhar no teste, ele pode ou não ser escalonável) .

Cerca de 10 anos após a proposição do primeiro modelo de teste de escalonamento, foi proposto um novo teste para verificar se um conjunto de tarefas era escalonável através de um modelo de atribuição de prioridades arbitrário, como descrito em (Audsley et al, 1995), conhecido como *response time*. Esse teste produz como resultado o pior tempo possível de respostas (*r - response time*) para uma tarefa no seu instante crítico e é calculado, de forma iterativa, pela seguinte equação.

$$r_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{T_i} \right\rceil C_j \qquad (2)$$

Em que C_i é a computação da tarefa *i*, *Bi* é o tempo máximo de bloqueio da tarefa *i*; e *hp(i)* representa o conjunto de tarefas de prioridade mais alta que a tarefa *i*.

O escalonamento de tarefas em sistemas multiprocessados gerou problemas na área de sistemas de tempo real, pois os algoritmos ótimos para escalonamento uniprocessado provaram-se sub-ótimos para o escalonamento de sistemas multiprocessados (Davis e Burns, 2011). Isso ocorre, pois a distribuição das tarefas entre os múltiplos processadores (fator não abordado por algoritmos não processados por se tratar de um único processador) é um fator de grande impacto na escalonabilidade de um sistema.

Com a necessidade crescente da utilização de arquiteturas multiprocessadas para sistemas de tempo real, foram criadas estratégias de escalonamento para contornar o problema da distribuição de tarefas entre os processadores em sistemas multiprocessados, como mostrado em (Davis e Burns, 2010). No decorrer desta subseção, diferentes estratégias para o escalonamento de tarefas em sistemas multiprocessados serão discutidas e avaliadas de acordo com suas vantagens e desvantagens. O foco desta revisão são as principais estratégias

encontradas na literatura. Também será discutida a viabilidade da implementação de uma análise de escalonamento para fluxos de comunicação nas mesmas.

2.1.1 Escalonamento Particionado

No escalonamento particionado, todas as tarefas do conjunto de tarefas do sistema são atribuídas para processadores de forma que, quando uma tarefa for atribuída para um processador, ela executará somente neste processador. O objetivo dessa estratégia de escalonamento é se beneficiar dos resultados ótimos de algoritmos de escalonamento uniprocessado, transformando um sistemas multiprocessado com *n* processadores em *n* subsistemas uniprocessados. O maior problema encontrado nessa abordagem é justamente como será dada a disposição das tarefas nos múltiplos processadores, pois essa disposição trata-se de um problema NP-completo conhecido como *bin packing* (Ullman, 1975).

Para realização da distribuição das tarefas nos múltiplos processadores em um sistema com escalonamento particionado, normalmente, são utilizadas heurísticas. Tais heurísticas podem ser simples como *first fit*, *best fit* e *worst fit,* ou mais complexas como a harmonização de períodos proposta em (Burchard, 1995). Além da abordagem de distribuição guiada por uma heurística, pode ser utilizada uma abordagem iterativa baseada em algoritmos de busca de solução como proposto em (Bonilha, 2011).

A grande desvantagem do escalonamento particionado reside na capacidade de processamento não aproveitada em cada um dos processadores, resultante da falta de tarefas que necessitem somente da quantidade de processamento disponível (se encaixem perfeitamente no *slot*). Como não são permitidas migrações de tarefas entre processadores nessa abordagem, uma tarefa não poderá executar parte de sua carga em um processador e o restante em outro processador, resultando no mau aproveitamento dos recursos. Este problema pode ser minimizado através de um bom algoritmo de mapeamento. Apesar do menor aproveitamento de recursos, o escalonamento particionado proporciona uma maior previsibilidade temporal do sistema e uma maior facilidade para a análise de sua escalonabilidade (tornando-se a análise de vários sistemas uniprocessados). Outra vantagem do escalonamento particionado torna-se evidente quando é inserida a noção de dependência entre tarefas e de comunicação entre estas que geram complicações para a migração de tarefas.

Com a noção de dependência entre tarefas introduzida em um sistema, uma nova restrição é inserida com o objetivo de prevenir que a tarefa dependente execute antes da tarefa da qual ela depende, tal restrição gerará um maior *overhead* no gerenciamento de migrações de tarefas, aumentando o custo de uma operação que já é naturalmente custosa. Ao se introduzir a noção de comunicação entre tarefas, o posicionamento das tarefas nos processadores do sistema se torna um fator crítico. Quanto maior a distância entre duas tarefas que se comunicam, maior será o tempo levado para a comunicação e maior será a interferência dessa comunicação em outras comunicações de outras tarefas. Isso ocorre, pois haverá um maior número de comunicações interceptadas para uma maior rota, fato demonstrado pelo modelo de análise de escalonamento para comunicações desenvolvido em

(Ni e McKinley,1993). Em sistemas particionados a localização das tarefas permanecerá fixa e, portanto, se torna consideravelmente mais simples garantir as restrições de dependência e se calcular o pior tempo possível para comunicações, sem gerar um grande *overhead* de intervenção.

Como demonstrado em (Lopez *et al*, 2001), o escalonamento particionado por RM (*rate monotonic*) utilizando uma heurística simples para distribuição de tarefas como o *first fit* (chamando RM-FF), pode alcançar um limite de utilização similar ao limite de utilização obtido para sistemas uniprocessados escalonados por RM, em torno de 69% para sistemas com mais de 16 processadores. Para *earliest deadline first* (algoritmo tradicional de escalonamento uniprocessado que varia a prioridade das tarefas em relação à proximidade de seu prazo), foi provado em (Lopez *et al*, 2004), que qualquer heurística de distribuição razoável (heurísticas capazes de distribuir tarefas entre os processadores sem exceder o limite de utilização de 100%) tem um limite de utilização de $m - (m - 1) \times \alpha$, sendo *m* o número de processadores e α a utilização máxima de uma tarefa. Este limite de utilização torna-se mais alto que o limite apresentado por RM-FF para $\alpha<0.34$, porém, gerando um maior *overhead* no sistema, se comparado ao RM-FF. O maior *overhead* ocorre devido à utilização de prioridades dinâmicas que geram uma maior quantidade de preempções. A Tabela 1 mostra o apanhado das técnicas de escalonamento particionado discutidas.

Técnica	Resultados	Comentários
EDF com heurística de distribuição razoável	Limite de utilização de $m - (m - 1) \times \alpha$.	Apresenta um melhor resultado em relação ao RM-FF em conjuntos de tarefas que não possuem *chuncky tasks*. Gera um *overhead* considerável no sistema devido à maior ocorrência de migrações, comparado a algoritmos de prioridade fixa.
RM-FF	Limite de utilização em torno de 69% para sistemas com mais de 16 processadores.	Gera um menor *overhead* em termos de migrações no sistema.

Tabela 1: Quadro comparativo de técnicas de escalonamento particionado.

2.1.2 Escalonamento Global

A estratégia de escalonamento global visa inserir todas as tarefas do conjunto de tarefas em

uma única fila de escalonamento, sendo que as tarefas são capazes de migrar entre processadores. Tais migrações, segundo (Carpenter *et al*, 2004), podem acontecer de acordo com dois critérios diferentes:

- **Migração em nível de tarefa:** a migração ocorre entre diferentes instâncias (execuções completas) de uma tarefa (conhecidas como *jobs*). Após iniciada a execução de um *job* em um processador, a tarefa permanece nesse processador até o fim da execução de seu *job*;

- **Migração em nível de *job*:** uma tarefa poderá migrar entre processadores mesmo durante a execução de um *job*.

A maior vantagem do escalonamento global reside na capacidade de migração das tarefas, especialmente, na migração em nível de *job*, possibilitando que a carga de uma tarefa seja dividida em duas porções que serão executadas em processadores distintos. Com a divisão da carga de uma tarefa, um maior aproveitamento de recursos é possível, pois os *slots* de tempo previamente não utilizados na abordagem particionada são preenchidos por cargas parciais de tarefas. Porém a própria migração de tarefas gera um grande *overhead* no sistema devido à troca de contexto nos processadores e transferência de código, podendo gerar custos proibitivos. A existência de migração de tarefas no sistema também acarreta em um comportamento temporal menos previsível para o sistema que deve ser compensado por intervenções maiores do escalonador (gerando um maior *overhead* de intervenção) ou análises mais complexas de escalonabilidade.

Para realizar o escalonamento de acordo com o escalonamento global são normalmente utilizadas duas classes principais de algoritmos de escalonamento, como mostrado em (Qi, Zhu e Aydin, 2011), os algoritmos de escalonamento guiados por prioridade (ou *priority driven scheduling*) e a classe de algoritmos de escalonamento conhecida como escalonamento justo (ou *fair scheduling*). Tais classes de algoritmo são detalhadas nas seções a seguir.

2.1.2.1 Priority Driven Scheduling

Como o próprio nome já diz, no *priority driven scheduling,* a ordem na qual as tarefas serão executadas é definida baseada em um sistema de prioridades. Nessa classe de algoritmos de escalonamento global, normalmente, são utilizados algoritmos de escalonamento particionado com adaptações visando gerenciar a migração das tarefas entre os vários processadores. De acordo com a classificação de (Carpenter *et al*, 2004), os algoritmos *priority driven* podem ser classificados como:

- **Prioridade fixa em nível de tarefa:** após atribuídas as prioridades para cada uma das tarefas, essas prioridades permanecerão fixas durante a vida do sistema;

- **Prioridade fixa em nível de *jobs*:** uma tarefa pode sofrer variação de prioridade entre execuções de seus *jobs*, porém, uma vez que uma prioridade tenha sido atribuída para um *job*, esse *job* permanecerá com essa mesma prioridade até o fim de sua execução;

- **Prioridade dinâmica:** a prioridade de uma tarefa irá variar conforme a execução se desdobra, mesmo durante a execução de um *job.*

Na classe de prioridade fixa em nível de tarefa, as abordagens mais bem sucedidas para o escalonamento global foram baseadas no algoritmo de escalonamento conhecido como *rate monotonic* (RM) proposto em (Liu e Layland, 1973). De acordo com (Qi, Zhu e Aydin, 2011), a abordagem conhecida como global-RMS (técnica de escalonamento global baseada em RM) obteve um limite de utilização (utilização máxima onde todos os conjuntos de tarefas serão escalonáveis) igual a $(m/2) \times (1 - \alpha) + \alpha$, como mostrado em (Baker, 2003), onde m é igual ao número de processadores do sistema e α é a utilização máxima de uma única tarefa. Baker também demonstrou, em (Baker, 2003) que o algoritmo RM-US (*rate monotonic utilization sensitive*), proposto em (Andersson *et al*, 2001), onde as tarefas são divididas em dois níveis de prioridade de acordo com um *threshold* estabelecido de utilização, atinge um limite de utilização de $(m + 1)/3$ para o *threshold* de 1/3 de utilização. Tais resultados para algoritmos de prioridade fixa em nível de tarefa são uma regressão do limite de utilização obtido em (Liu e Layland, 1973), que girava em torno de 66% de utilização.

Em prioridade fixa em nível de *job,* o algoritmo mais bem sucedido foi o global-EDF, mostrado em (Goossens *et al*, 2003), obtendo um limite de utilização de $m \times (1 - \alpha) + \alpha$. O algoritmo de global-EDF, trata-se de uma adaptação do *earliest deadline first* (EDF), um algoritmo tradicional em sistemas uniprocessados. No global-EDF, quando é dado o *release* de um *job* de uma tarefa, é verificada a proximidade do tempo de *release* em relação a seu prazo e a proximidade dos outros *jobs* em relação a seus prazos, a maior proximidade do prazo resultará na atribuição de uma maior prioridade para o *job*. Deve ser ressaltada a complexidade de um teste de escalonabilidade exato para esse algoritmo, proposto em (Goossens e Yomsi, 2010), onde é necessária a análise de um intervalo que pode chegar a $(\sum C) \times P$ unidades de tempo, sendo C a carga computacional de cada uma das tarefas e P, o mínimo múltiplo comum de seus períodos. Deve também ser ressaltado que a variação de prioridade entre jobs resulta em uma maior variação do posicionamento das tarefas, o que torna a previsão de tempo de comunicação entre tarefas extremamente difícil. Para o global-EDF, a introdução de uma única tarefa com uma utilização acima de 34%, dependendo do número de processadores, poderá resultar em um limite de utilização abaixo do limite obtido em 1973 para sistemas uniprocessados. Esse limite de utilização obtido para o global-EDF também provê limites de utilização razoavelmente próximos aos obtidos através do EDF particionado com heurísticas de distribuição razoáveis.

Para o esquema de prioridade dinâmica não serão citados exemplos, pois a variação frequente de prioridade gera uma imprevisibilidade de localização das tarefas muito grande, gerando *overhead* de migração proibitivos e a impossibilidade de garantias de tempo de

comunicação.

Os algoritmos de escalonamento global orientados à prioridade têm como objetivo obter um maior aproveitamento de recursos do sistema, comparados aos algoritmos particionados através da migração de tarefas, sem gerar um *overhead* de intervenção significantemente maior. Porém, na literatura atual não foram propostos algoritmos que resultem em uma melhora significativa, sendo que muitos algoritmos apresentam resultados piores que o particionado. Agregando a esse fato, deve ser mencionado que a migração de tarefas possui um custo elevado para o sistema, gerando também uma maior imprevisibilidade no comportamento temporal da comunicação entre as tarefas. A Tabela 2 mostra o resumo das técnicas de escalonamento global baseado em prioridades discutidas.

Técnica	Resultados	Comentários
Global-RMS	Limite de utilização de$(m/2) \times (1-\alpha) + \alpha$.	Maior previsibilidade temporal comparado com técnicas de prioridade fixa em nível de *job*. Limite de utilização abaixo do obtido por métodos de escalonamento particionado.
RM-US	Limite de utilização de$(m+1)/3$.	Maior previsibilidade temporal comparado com técnicas de prioridade fixa em nível de *job*. Limite de utilização abaixo do obtido por métodos de escalonamento particionado.
Global-EDF	Limites de utilização similares ao EDF particionado com heurística de distribuição razoável.	Menor previsibilidade temporal que demanda análises de escalonamento mais complexas. Maior *overhead* no sistema comparado ao EDF particionado devido às migrações.

Tabela 2: Quadro comparativo de técnincas de escalonamento global baseadas em prioridade.

2.1.2.2 Fair Scheduling

Os algoritmos de *fair scheduling* baseiam-se no conceito de escalonamento fluído, demonstrado em (Srinivasan *et al*, 2003), que consiste em considerar um sistema onde trocas de contexto, migrações e preempções não possuem custo. Em tal sistema idealizado, as tarefas podem constantemente fazer a preempção umas das outras de forma que todas apresentem um progresso relativo ao tempo proporcional. Quando o intervalo de preempções se torna muito pequeno surge a ilusão de que todas as tarefas estão executando simultaneamente, gerando uma curva de escalonamento fluído onde todas apresentam o mesmo progresso proporcional ao mesmo tempo. O objetivo do *fair scheduling* é aproximar-se dessa curva de escalonamento

fluído pois, dessa forma, mesmo com uma utilização de 100% do sistema, desconsiderando *overheads*, todas as tarefas cumprirão seus prazos.

O primeiro algoritmo de destaque desenvolvido nessa área foi o *P-fair* (P*roportional Fair*)*,* proposto em (Baruah *et al*, 1996). Este algoritmo utiliza um escalonador que intervem a cada unidade de tempo do processador com o objetivo de obter a maior proximidade possível com o escalonamento fluído. A cada unidade de tempo é calculada a quantidade de tempo de processador que deveria ter sido atribuída para cada tarefa se o escalonamento fosse completamente fluído. Então, essa quantidade de tempo é confrontada com a quantidade de tempo de processador que cada tarefa não obteve. Essa diferença entre a quantidade devida e a quantidade recebida é chamada de *lag*, o objetivo do *P-fair* é não permitir que o *lag* de nenhuma das tarefas do sistema ultrapasse o valor 1 (significando a discrepância de uma unidade de tempo entre a quantidade de tempo recebida e a quantidade de tempo devida). Desconsiderando-se *overheads* de migração e troca de contexto, esse algoritmo garante escalonabilidade de sistemas com até 100% de utilização, porém, ao se considerar esses *overheads,* nota-se que esse algoritmo beira o inviável.

Em 2003, um novo algoritmo de *fair scheduling* foi proposto em (Zhu, Mossé e Melhem, 2003), visando a redução do *overhead* de intervenção, preempção e migração resultantes do algoritmo P-*fair*. Esse algoritmo, conhecido como B-*fair* (*Boundary Fair*), se foca no fato de que, para garantir o cumprimento de todos os prazos, não é necessária uma curva de escalonamento tão próxima da fluída como o P-*fair* gera. Para garantia do comprimento de todos os prazos, é necessário que a curva de escalonamento intercepte a curva de escalonamento fluído nos limites de prazo de cada uma das tarefas, ou seja, quando o prazo de uma tarefa chegar, a distribuição do tempo deve ser justa, não importando como o tempo foi distribuído no meio tempo entre prazos. O algoritmo interfere no escalonamento sempre que um prazo é atingido, formulando o escalonamento a ser realizado até o próximo prazo. A formulação do escalonamento leva em conta o tempo que cada tarefa deveria receber no intervalo entre prazos, cada tarefa receberá pelo menos a parte inteira do seu tempo necessário. A quantidade de tempo que uma tarefa precisa pode ser fracionária. Caso restem unidades de tempo após o escalonamento de todas as partes inteiras, serão determinadas tarefas para receber esses *slots* opcionais, sendo esse processo guiado através de um sistema de prioridades (normalmente, orientado a utilização). O tempo não atendido (parte fracionária de tarefas que não executaram em *slots* opcionais) será acumulado e somado ao próximo cálculo de tempo necessário de execução. Na abordagem do B-*fair*, o número de migrações é reduzido drasticamente em relação ao P-*fair*, que passa a ser uma intervenção por unidade de tempo para uma intervenção a cada marcador de prazo na escala temporal. Uma redução cada vez maior é apresentada conforme os períodos das tarefas do conjunto se aproximam do harmônico, ainda mantendo a garantia de escalonabilidade de sistemas com 100% de utilização. Apesar da redução drástica no *overhead*, o *overhead* do B-*fair* para escalonamento global ainda pode ser proibitivo, como mostrado em (Qi, Zhu e Aydin, 2011). As migrações de tarefas entre processadores também geram imprevisibilidade no comportamento temporal das comunicações entre tarefas e dificulta a introdução da noção de dependência. A Tabela 3 mostra em suma as técnicas de *fair scheduling* abordadas.

Técnica	Resultados	Comentários

B-Fair	Escalona tarefas em sistemas com até 100% de ocupação.	Alta ocorrência de trocas de contexto devido a migrações e preempções, resultando em um *overhead* proibitivo no sistema.
P-Fair	Escalona tarefas em sistemas com até 100% de ocupação.	Apesar de reduzir significativamente o número de intervenções e de preempções em relação ao B-Fair, ainda gera *overhead* significativo devido a troca de contexto.

Tabela 3: Quadro comparativo de técnicas de escalonament global com a abordagem fair scheduling.

2.1.3 Escalonamento Semi-particionado

O objetivo do escalonamento semi-particionado é tentar reduzir a subutilização de recursos do escalonamento particionado de tal forma que seja gerado um comportamento temporal e de localização de tarefas em processadores mais previsível que o encontrado em escalonamento global. Para a obtenção de uma maior previsibilidade do sistema, esse algoritmo se sustenta no conhecimento de quais tarefas migrarão e para onde elas migrarão, dessa forma gerando um sistema onde a grande maioria de suas tarefas permanecem no mesmo processador em que iniciaram e apenas um grupo seleto de tarefas tem permissão para migrar entre processadores, sendo que seus processadores de origem e destino são previamente conhecidos e formam um ciclo (a tarefa sempre migrará entre os mesmos processadores de forma cíclica). Como a maioria das tarefas ficam restritas a um único processador e as tarefas migratórias migram em padrões pré-determinados, essa abordagem fornece um comportamento temporal previsível para comunicações, diferente do escalonamento global.

O escalonamento semi-particionado começou a ser discutido em (Andersson e Tovar, 2006), com a proposição do algoritmo EKG. Esse algoritmo baseia-se em dois conceitos principais: isolar *chuncky tasks* (tarefas que geram alta utilização) em processadores próprios e quebrar tarefas que não puderam ser escalonadas em nenhum dos processadores em sub-tarefas, dividindo sua carga computacional de forma que seja modelada a migração da tarefa em questão. O algoritmo inicia separando tarefas consideradas muito grandes de tarefas menores, essa seleção é dada através de um valor limite *SEP*, calculado pela fórmula $\frac{k}{(k+1)}$, onde k é um parâmetro escolhido pelo desenvolvedor. Se o k escolhido for igual ao número de processadores, *SEP* é considerado 1 (apenas tarefas com 100% de utilização serão separadas), caso contrário, o SEP partirá de 0.5 (para $k=1$). Após feita a separação, as tarefas serão ordenadas de forma que as tarefas de alta utilização encabeçarão a fila e, então, serão distribuídas entre os processadores. A distribuição isolará tarefas de alta utilização em processadores, inserindo o restante das tarefas nos processadores restantes de forma que a utilização desses processadores não exceda 100%. O algoritmo agrupará os processadores restantes em grupos de k processadores, caso não tenha sido possível acomodar uma tarefa no último dos k processadores de um grupo, essa tarefa será quebrada em duas sub-tarefas que serão alocadas em dois processadores adjacentes dentro do grupo. Caso não haja espaço nos

processadores restantes após o isolamento das tarefas de alta utilização, será considerado que o algoritmo falhou e um novo *k* deverá ser determinado. Após feita a separação das tarefas, as tarefas particionadas serão escalonadas por EDF (algoritmo de prioridade dinâmica) e as tarefas divididas serão consideradas de alta prioridade, sendo executadas antes das tarefas particionadas, restrições são aplicadas para que duas partes da mesma tarefa não sejam executadas simultaneamente. Segundo (Andersson e Tovar, 2006), esse algoritmo provou ter um limite de utilização de 66% para *k=2*, podendo chegar a um limite de 100% de utilização para *k=m*, porém, gerando um *overhead* significativo. De acordo com (Levin *et al*, 2010) não foram derivados limites de ocorrência de preempção para algoritmos *fair* e para o EKG, mas é esperado que o EKG apresente uma melhor performance que algoritmos como o B-*fair* em termos de *overhead* de preempção.

Em (Kato e Yamasaki, 2009), foi proposto um algoritmo de escalonamento, conhecido como semi-particionado baseado no algoritmo de prioridade fixa DM (*deadline monotonic*), sendo o DM idêntico ao RM caso os períodos das tarefas sejam igual ao seu prazo (convenção normalmente adotada nos modelos computacionais de tempo real). Esse algoritmo, conhecido como DM-PM, não realiza o isolamento de tarefas de alta prioridade como o EKG mas, procura alocá-las nos processadores antes das demais tarefas. Após as tarefas de alta utilização (acima de 50%) tenham sido alocadas, as demais tarefas são ordenadas em ordem não crescente de prazos e, então, distribuídas dentre os processadores dessa forma. Tarefas serão inseridas em um processador até que o sistema se torne não escalonável, sendo que sua escalonabilidade é determinada por uma modificação da análise por *response time* (Joseph e Pandya, 1986). Quando o conjunto de tarefas em um processador se torna não escalonável, a última tarefa a ser inserida é quebrada em até 3 sub-tarefas, de forma que as sub-tarefas inicias devem esgotar toda capacidade de execução restante nos processadores aos quais foram alocadas, caso toda carga de execução tenha sido esgotada na segunda sub-tarefa, uma terceira não será gerada. Nesse algoritmo todas as tarefas serão escalonadas da mesma forma, através do algoritmo DM. Como mostrado em (Kato e Yamasaki, 2009), esse algoritmo apresenta o pior caso de limite de utilização, quando existem tantas tarefas de alta utilização quanto processadores, apresentando um limite superior a 50% de utilização, variando de acordo com o tamanho das tarefas. Para casos onde existam uma ampla variação entre utilização das tarefas do conjunto, esse algoritmo provou, através de simulações, obter uma taxa de sucesso de 100% para conjuntos de tarefas com até 90% de utilização do sistema. Em termos de *overhead* de preempção, esse algoritmo deve gerar menos preempções que o EKG, tendo em vista que ele usa um sistema único de prioridade fixa para alocar todas suas tarefas e permite que uma tarefa seja dividida em, no máximo, 2 partes. Um limite para preempções para o DM-PM foi calculado, porém, não existem limites definidos para o EKG, então, uma comparação exata não é possível.

Kato também apresentou, em (Kato, Yamasaki e Ishikawa, 2009), um algoritmo de escalonamento semi-particionado baseado em EDF utilizando a noção de prazos arbitrários (prazos menores que o período da tarefa). Esse algoritmo, conhecido como EDF-WF, distribui as tarefas através de qualquer heurística razoável de distribuição entre os vários processadores, transformando o conjunto de tarefas restantes (que não foram alocadas com sucesso em nenhum processador) em tarefas migratórias. Tais tarefas são divididas em sub-tarefas com período idênticos, porém, com prazos arbitrários, sendo esses prazos calculados através do algoritmo proposto em (Kato, Yamasaki e Ishikawa, 2009) que utiliza uma análise de escalonamento conhecida como *demand bound function*, proposto em (Baruah *et al*, 1999). Esse algoritmo apresentou uma performance similar ao DM-PM, obtendo uma taxa de 100% de sucesso para escalonamento de conjuntos de tarefas acima de 90% de utilização total do

sistema. Porém, sem apresentar o pior caso descrito pelo algoritmo anterior quando deparado com conjuntos com um grande número de tarefas com utilização acima de 50%. Apesar de não possuir o pior caso apresentado no DM-PM, esse algoritmo resulta em um maior *overhead* de intervenção por utilizar um sistema de prioridades dinâmicas e prazos arbitrários.

Em (Dorin, Yomsi e Goossens, 2010), foi proposto um algoritmo de escalonamento particionado seguindo um paradigma diferente dos anteriores. Em certas ocasiões a carga direta de trabalho de uma tarefa durante um ou mais períodos não causa a perda de prazo de outras tarefas, no entanto, a execução de seus *jobs* consecutivos acumula interferência temporal em outras tarefas o que acaba gerando uma perda de prazo. Tal algoritmo tenta evitar que essa interferência se acumule a ponto de gerar perdas de prazos, fazendo com que uma tarefa realize a migração entre *jobs*, de forma que o job que acumularia a interferência a ponto de causar uma perda de prazo execute em outro processador. Inicialmente, as tarefas são alocadas através de *first fit* seguindo a ordem decrescente de período e, quando um conjunto de tarefas em um processador se tornar não escalonável, essa tarefa será transformada em uma *multi-framed task* que constitui uma tarefa com um padrão de execução definido (será mapeado em quais cores cada job da tarefa deve executar em um intervalo igual aos mínimo múltiplo comum dos períodos das tarefas do sistema). O padrão de execução deve ser cíclico e é definido através de dois algoritmos apresentados em (Dorin, Yomsi e Goossens, 2010), os quais necessitariam de uma explicação muito extensa para constar neste trabalho. Para a definição da escalonabilidade dos conjuntos de tarefas nos processadores é utilizada uma variação do *demand bound function*, pois essa análise em particular determina a interferência cumulativa gerada pela execução consecutiva de *jobs* de uma tarefa em um intervalo de tempo arbitrário. Após a alocação e divisão das tarefas, as tarefas são escalonadas de acordo com o algoritmo EDF. Este algoritmo apresentou uma performance inferior ao EDF-WF em termos de taxa de sucesso para altas utilizações, apresentando uma taxa de sucesso de 100% de escalonamento para conjuntos até 80% de utilização, começando a decair acima dos 80% de utilização do sistema. Apesar de uma taxa de sucesso mais baixa que o EDF-WF, esse algoritmo resulta em uma menor taxa de preempções pois faz a migração entre *jobs*, ou seja, a tarefa já haverá terminado de executar e não precisará ser interrompida para o início da migração. A Tabela 4 mostra um apanhado das técnicas de escalonamento semi-particionado discutidas.

Técnicas	Resultados	Comentários
EKG	Escalonamento de sistemas com utilização entre 66% e 100%.	Faz um *trade-off* de maiores utilizações por um maior *overhead* de migração. Gera menos *overhead* devido a trocas de contexto se comparado com algoritmos *fair* mas ainda gera um *overhead* considerável.

DM-PM	100% de sucesso para escalonamento de conjuntos de tarefas com utilização de até 90% de utilização.	Gera um menor *overhead* devido a trocas de contexto, se comparado ao EKG.
EDF-WF	100% de sucesso para o escalonamento de conjuntos de tarefas com utilização de até cerca de 94%.	Gera um menor *overhead* devido a trocas de contexto relacionado ao EKG, porém, maior que o EDF-WF.
Algoritmo de Dorin, Yomsi e Goossens	100% de sucesso no escalonamento de conjuntos de tarefas com até cerca de 80% de utilização.	Gera um menor *overhead* de troca de contexto em migrações, devido a realização de migrações entre *releases* de *jobs*.

Tabela 4: Quadro comparativo de técnicas de escalonamento semi-particionado.

2.1.4 Escalonamento Clusterizado

O escalonamento clusterizado tem como objetivo reduzir o alto *overhead* gerado pelo escalonamento global, enquanto se aproveita da maior taxa de utilização possível de ser obtida através dessa abordagem. Nessa estratégia de escalonamento, o sistema é dividido em n subsistemas compostos por k processadores (sendo o conjunto de k processadores chamado de *cluster*), sendo que o número total de processadores do sistema é dado por $n \times k$. As tarefas do conjunto de tarefas do sistema são alocadas aos diversos *clusters*, através de uma heurística de distribuição como as utilizadas na distribuição de tarefas em sistemas particionados. Em cada *cluster*, as tarefas serão escalonadas através de um algoritmo de escalonamento global, sendo esse, normalmente, um algoritmo *fair,* pois essa classe de algoritmo é capaz de escalonar conjuntos de tarefas que gerem até 100% de utilização do sistema. A estratégia de escalonamento clusterizado pode fornecer previsibilidade temporal para comunicações de duas formas: através de uma análise pessimista das comunicações dentro do *cluster* e que partem dele (que funcionaria apenas com *clusters* pequenos sem se tornar proibitivamente pessimista) ou considerando-se uma arquitetura onde os processadores de um *cluster* compartilham memória, tornando o tempo de comunicação entre eles consideravelmente mais baixo e possivelmente desprezíveis (dessa forma apenas a comunicação entre clusters deveria ser analisada).

Em (Qi, Zhu e Aydin, 2010), é proposto um algoritmo de escalonamento clusterizado

focado no escalonamento por B-*fair* e com o objetivo de reduzir o *overhead* gerado por esse algoritmo. Essa abordagem baseia-se no fato de que a quantidade de intervenções realizadas pelo escalonador B-*fair* está diretamente relacionada à desarmonização dos períodos das tarefas. Se os períodos forem completamente harmônicos, haverá apenas uma intervenção a cada intervalo de tempo condizente com o menor período dentre as tarefas. Para a distribuição das tarefas entre os *clusters* é utilizada a heurística PAFF (*period aware first fit*), no qual as tarefas primeiramente são agrupadas baseadas na multiplicidade de seus períodos. Tarefas que compartilhem um mínimo múltiplo comum diferente da multiplicação de seus períodos são agrupadas. Após o agrupamento, todos os grupos são inseridos na cabeça da lista e, após isso, o restante das tarefas são inseridas e, então, as tarefas são distribuídas nos *clusters* através do *first fit.* O objetivo do PAFF é agrupar o maior número possível de tarefas harmônicas nos clusters, assim, reduzindo a quantidade de pontos de intervenção do escalonador B-*fair*. Um estudo foi realizado para determinação do limite de utilização para B-*fair* clusterizado e a redução provocada no *overhead* gerado por essa abordagem em relação ao B-*fair* global, como mostrado em (Qi, Zhu e Aydin, 2011). O PAFF B-*fair* clusterizado apresentou taxa de 100% de sucesso no escalonamento de conjuntos de tarefas acima de 95% de utilização de sistema para sistemas com 16 e 64 processadores (com *clusters* de 4 ou mais processadores), sendo o pior caso de utilização dado por $\frac{([k/\alpha]\times m+1)}{([k/\alpha]+1)}$, sendo k o número de *clusters* no sistema, m o número de processadores no sistema e α a utilização máxima apresentada entre as tarefas do sistema. Foi constatada, em relação ao B-*fair* global, uma redução de mais de 90% nas trocas de contexto e migrações para $k=2$, continuando a apresentar uma redução substancial para *clusters* maiores. Finalmente, através de testes de execução, foi reportada uma redução no *overhead* gerado pelo escalonador a cada ponto de intervenção de até 90% para *clusters* de 4 processadores em sistemas de 16 e 64 processadores. O PAFF B-*fair* clusterizado apresenta uma melhora dramática em relação à abordagem global e provê a possibilidade da obtenção de comportamento temporal previsível para comunicações, considerando as hipóteses previamente discutidas.

2.2 Escalonamento de Fluxos de Comunicação

Desde a criação dos conceitos da área de sistemas de tempo real, a complexidade dos sistemas vem crescendo. Nos sistemas atuais, é comum existir relações de dependência de dados entre tarefas, aonde uma tarefa necessitará de dados processados por uma segunda tarefa para poder desempenhar sua função de forma apropriada. Tais relações não podem ser ignoradas em um modelo de análise de escalonamento em um ambiente de tempo real pois, para obtenção de garantias temporais, é necessário que o pior comportamento temporal por parte de cada uma das tarefas do sistema seja levado em consideração.

Para que tais garantias temporais sejam obtidas para as tarefas do sistema, é necessário conhecer o comportamento temporal dos fluxos de comunicação do sistema, pois as relações de dependência de dados previamente mencionadas podem causar atrasos nas tarefas dependentes. O atraso na chegada de um fluxo de comunicação pode causar, desde a perda da qualidade de serviço de sistema (causando a deterioração nos resultados de uma tarefa devido

à utilização de dados posteriores) até a falha no cumprimento de prazo de uma tarefa que não possa operar sem dados atualizados. Esta subseção trata do modelo de análise de escalonamento de fluxos utilizado para o desenvolvimento deste trabalho.

Neste trabalho será considerada a arquitetura de NoC, que comumente utiliza algoritmos determinísticos para o roteamento de comunicação entre tarefas provendo uma maior previsibilidade do comportamento dos fluxos de comunicação. Para comunicação entre tarefas, é considerado o modelo de comunicação baseado em NoCs proposto em (Ni e McKinley, 1993) conhecido como Wormhole Switching. O modelo de Wormhole Switching consiste em uma arquitetura quatro-conectada, com um roteador no ponto intermediário de cada barramento. Esses roteadores possuem um buffer capaz de guardar n pacotes. O envio dos pacotes é orientado pela prioridade associada a cada fluxo e é feito de acordo com o algoritmo XY, que consiste na propagação do fluxo primeiramente no eixo x até atingir a coordenada desejada do destino e, então, a propagação final até o destino pelo eixo y. O envio de cada pacote só é realizado, caso haja espaço no buffer do próximo roteador na trajetória do pacote. Caso contrário, o pacote permanecerá armazenado no buffer do roteador atual até que haja espaço para sua recepção no próximo roteador. Para o manejamento de confronto de rotas, as rotas do sistema são representadas através de canais virtuais sendo esses um conjunto ordenado de roteadores por onde um fluxo de dados deve passar até seu destino. Um fluxo de comunicação estará alocado a um canal virtual caso sua rota (respeitando ordenação) esteja contida no mesmo, podendo compartilhar este canal virtual com outros fluxos que atendam a mesma condição.

2.2.1 Anatomia de um fluxo

Em (Shi e Burns, 2010), foi proposto um modelo para o escalonamento de comunicações baseado no modelo de análise de escalonamento proposto em (Audsley *et al*, 1995) e utilizando o *Wormhole Switching* como modelo de comunicação. Este modelo utiliza equações baseadas no *response time analysis*.

De acordo com o modelo apresentado em (Shi e Burns, 2010), cada fluxo pode ser representado por uma série de atributos, possuindo uma trajetória ou rota (*t* - *route*) que se trata de um conjunto ordenado de barramentos pelo qual os dados têm que passar. Um prazo (*D* - *deadline*) que representa o tempo limite para a chegada de seus dados no destino, uma latência (*C* - *latency*) que define o pior tempo que o conjunto de dados que compõe o fluxo levará para passar por um barramento, um período (*T* - *period*) que representa o mínimo intervalo entre duas liberações consecutivas do conjunto de pacotes pertencentes ao fluxo e uma prioridade (*P* - *priority*) que define a precedência dos pacotes pertencentes ao fluxo em relações a pacotes pertencentes a outros fluxos. A prioridade pode ser compartilhada por múltiplos fluxos, pois ela não é atribuída diretamente ao fluxo, ela é atribuída a canais virtuais que consistem em uma rota pré-determinada. Todo fluxo ao qual sua rota estiver contida no canal virtual, fará parte do mesmo e, consequentemente, receberá a prioridade designada para

este canal virtual. Finalmente, os fluxos de dados também possuem um atraso de liberação (J^R – *release jitter*) que indica o possível atraso no inicio do envio de dados de um fluxo após sua liberação e o atraso devido à interferência (J^i - *interference jitter*) que indica o desvio máximo entre o envio de conjuntos de pacotes consecutivos pertencentes ao fluxo em relação ao período do mesmo.

Definição 2.1 (Fluxo de Comunicação). Um fluxo de comunicação representado pela tupla $F_i(T_i, J_i^R, C_i)$ é composto por um período T_i, um atraso de liberação J_i^R e um tempo de transmissão C_i. Considera-se que o prazo do fluxo é igual ao seu período.

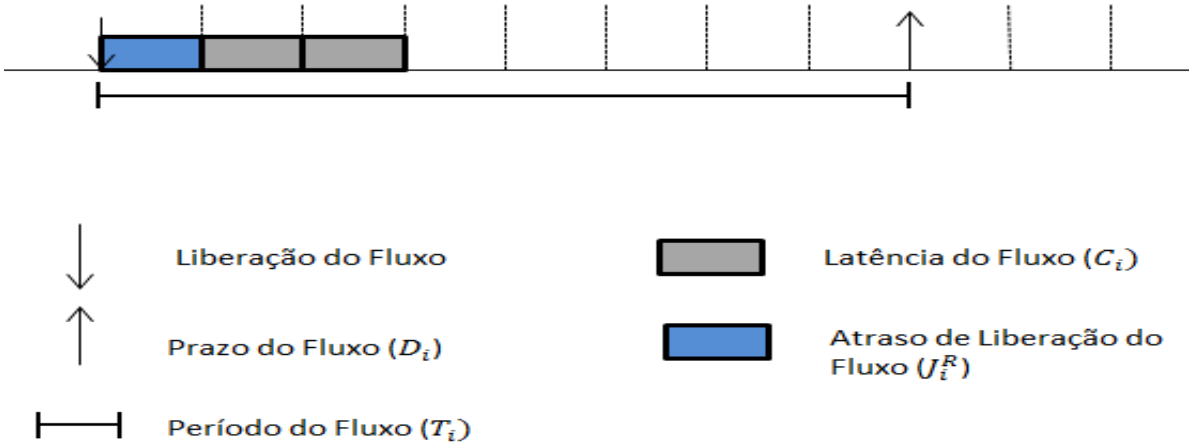

Figura 1: *Anatomia de um fluxo de comunicação.*

2.2.2 Analisando o Escalonamento de Fluxos

No modelo de análise de escalonamento proposto por Shi e Burns, é introduzido o conceito de interferência indireta. Tal interferência é proveniente de fluxos que não compartilham nós em comum com a rota do fluxo sendo analisado, mas compartilham pelo menos um nó com a rota de fluxos que possuem uma relação de interferência direta com o fluxo sendo analisado. Shi e Burns provaram que apesar de tais fluxos não possuírem uma relação de interferência direta com o fluxo sendo analisado, sua interferência em outros fluxos, que possuem relação de interferência direta, podem reduzir a janela de tempo entre dois hits sucessivos dos mesmos (fenômeno conhecido como " back to back hit" (Audsley et al, 1993), impondo atraso no término da atividade de tais fluxos. Considerando esse cenário, o atraso sofrido por fluxos interferentes no término de suas atividades é modelado como uma soma do atraso de liberação dos fluxos e o atraso de interferência (causado por fluxos com uma relação de interferência indireta). Tais atrasos são modelados na equação clássica de *response time* com atraso (jitter) (Audsley et al, 1993), resultando na seguinte equação:

$$W(i)^k = C_i + \sum_{j \in Hp(i)} \left\lceil \frac{W(i)^{K-1} + (J_j^R) + (J_j^I)}{T_j} \right\rceil \times C_j \quad (1)$$

Sendo hp(i) o conjunto de fluxos de mais alta prioridade que o fluxo i que compartilham pelo menos um nó da rota do fluxo i. j é cada uma das instâncias de hp(i), J_j^R é o atraso de liberação do fluxo j e J_j^I é o atraso de interferência imposto no fluxo j e *k* o índice da iteração do método numérico.

Tendo sido calculada a janela de prioridade final (referida como $W(i)$), é feita uma análise comparando o valor da janela de prioridade com o período do fluxo. Essa comparação pode resultar em três diferentes situações:

- $W(i) \leq min(T_n - J_n^R)$, sendo *n* qualquer fluxo do sistema fora o fluxo *i*. Condição que indica que a janela de prioridade do fluxo acaba antes de liberações repetidas de fluxos em qualquer um dos canais virtuais envolvidos na análise.

- $\min(T_n - J_n^R) < W(i) \leq T_i - J_i^R$, representando o fato de que liberações repetidas de fluxos pertencentes a outros canais virtuais ocorreram. Porém, nenhuma liberação repetida de fluxos pertencentes ao canal ao qual está se analisando no momento ocorre.

- $W(i) > T_i - J_i^R$, situação na qual ocorreram liberações repetidas em outros canais e no canal que está sendo analisado.

Depois que a janela de tempo na qual o fluxo i termina sua atividade de comunicação foi calculada, seu atraso de liberação é somado, resultando no pior tempo possível para o término das atividades de comunicação do fluxo i (conhecido como *response time*).

$$R_i = W(i) + J_i^R \quad (2)$$

Com o objetivo de simplificar a equação, os autores usam um limite superior para o atraso de interferência sofrido pelos fluxos, através da seguinte premissa:

$$J_i^I \leq R_i - C_i \quad (A)$$

É garantido que o valor de J_i^I nunca ultrapassará o valor de $R_i - C_i$, considerando que, se a latência do fluxo for removida de seu *response time*, tudo que restará será a interferência sofrida pelo fluxo i, por fluxos concorrentes, no decorrer de sua execução.

2.3 Considerações finais

No decorrer deste capítulo foram discutidos trabalhos relacionados ao trabalho sendo desenvolvido. Diversas categorias de algoritmos de escalonamento foram discutidas e suas principais vantagens e desvantagens expostas. Durante essa análise duas categorias de algoritmos se destacaram: escalonamento semi-particionado e escalonamento clusterizado. Devendo ser ressaltado que nenhum dos trabalhos estudados leva em conta comunicação entre tarefas, sendo a possibilidade do desenvolvimento de um modelo de comunicação para estas abordagens levada em consideração neste trabalho.

Apesar de a técnica EKG conseguir escalonar conjuntos de tarefas de até 100% de utilização, ela não é a técnica mais eficiente em escalonamento semi-particionado. Pois é realizado um *trade-off* de capacidade de escalonamento para o conjunto com maior utilização de processador por um maior *overhead* no sistema, sendo esse *overhead* proíbitivo para conjuntos de tarefas com 100% de utilização. As técnicas que se provaram mais eficientes para escalonamento semi-particionado foram as duas abordagens propostas por Kato (DM-PM e EDF-WF). A abordagem de EDF-WF chegou a obter um percentual próximo de 100% de sucesso no escalonamento de conjuntos de tarefas com cerca de 97% de utilização e a técnica DM-PM obteve em torno de 100% de sucesso no escalonamento para conjuntos de tarefa em torno de 93%. Apesar de o EDF-WF possuir uma maior capacidade de escalonamento, o *overhead* infligido no sistema pelo DM-PM é menor, pois o DM-PM utiliza prioridades fixas. Outra vantagem da abordagem semi-particionada é o padrão previsível de migrações das tarefas, pois elas migram entre processadores fixos e em momentos conhecidos. Esse padrão de migração conhecido possibilita a obtenção de comportamentos e tempos previsíveis de comunicação, tornando plausível a adaptação de análises de escalonamento de comunicação para sistemas particionados para obtenção de garantias temporais para as mesmas em um sistema semi-particionado .

A abordagem clusterizada obteve uma taxa de sucesso de escalonamento similar ao EDF-WF, porém não existem estudos comparativos entre o *overhead* inflingido pelas duas abordagens. Possivelmente, o *overhead* da abordagem clusterizada será maior que da técnica semi-particionado, pois na abordagem clusterizada os sistemas são escalonados por *B-fair* e essa técnica tende a executar tarefas em pequenas parcelas, gerando um maior número de trocas de contexto. A abordagem clusterizada pode também prover previsibilidade de comunicações. Porém, para obtenção de previsibilidade de comunicações na abordagem clusterizada, deve existir memória compartilhada entre os processadores de cada cluster, de forma que o tempo de comunicação interno do cluster possa ser descartado ou limitado superiormente sem uma análise extremamente pessimista, sendo a análise do escalonamento de comunição entre clusters realizada através de um modelo de análise de escalonamento de comunicações para sistemas particionados como o exposto em (Shi e Burns, 2010).

Para as abordagens globais e particionadas, existem desvantagens significativas a serem consideradas. A abordagem particionada exerce o menor *overhead* no sistema dentre todas as abordagens mencionadas, porém, tem uma taxa de aproveitamento de processamento girando em torno de 69%, sendo consideravelmente mais baixa que as outras abordagens. Em termos de previsibilidade de comunicação, já existem modelos eficientes para previsão de comunicações em sistemas particionados na literatura atual. A abordagem de escalonamento global possui técnicas que alcançam 100% de aproveitamento de processamento, as técnicas de *fair schedulling*. Porém, técnicas de *fair schedulling* exercem o maior *overhead* no sistema dentre todas as abordagens estudadas, tornando seu uso proibitivo. Outro grande problema do *fair schedulling* é a o comportamento não fixo de migração das tarefas, tornando a obtenção de previsibilidade nas comunicações uma tarefa difícil.

O resumo da análise comparativa que foi realizada neste trabalho, entre as vantagens e desvantagens de cada uma das abordagem, é exposto na Tabela 5.

Categoria	Capacidade de escalonamento	Vantagens	Desvantagens
Escalonamento particionado	Limite de utilização de aproximadamente. 69% (RM-FF)	Fornece uma maior previsibilidade temporal para comunicações entre tarefas.	Baixo aproveitamento da capacidade computacional do sistema.
Escalonamento global	Escalonamento de sistemas com até 100% de utilização. (*B-fair* e *P-Fair*)	Alto aproveitamento de recursos computacionais com algoritmos de *fair scheduling*.	Geração de *overheads* proibitivos no sistema e imprevisibilidade temporal nas comunicações.
Escalonamento semi-particionado	100% de sucesso no escalonamento de sistemas com até cerca de 100% de utilização. (EKG)	Alto aproveitamento do poder computacional do sistema. Comunicações possuem trajetórias previsíveis.	Gera *overhead* no sistema devido a migrações e trocas de contexto. O *overhead* tende a aumentar junto com o limite da utilização do sistema.

Escalonamento clusterizado	100% de sucesso no escalonamento de sistemas com até cerca de 97% de utilização.	Alto aproveitamento do poder computacional. Previsibilidade nas comunicações entre clusters. Redução considerável nos *overheads* gerados pelo *fair scheduling* global.	Necessita de memória compartilhada em cada cluster para obtenção de previsibilidade total de comunicações. Ainda gera *overheads* consideráveis de escalonamento e troca de contexto.

Tabela 5: Quadro comparativo de todas abordagens consideradas no trabalho focando em comunicação entre tarefas

3. Análise de Escalonamento de Fluxos de Comunicação e Modelo de Sistema

Inicialmente, este Capítulo tem como objetivo apresentar melhorias sobre a análise de escalonamento de fluxos de comunicação descrita em (Shi e Burns, 2010), apresentada na Seção 2.2.2 desta dissertação. Mais especificamente, o limite superior utilizado como interferência indireta na análise de escalonamento de (Shi e Burns, 2010) introduz um grau de pessimismo desnecessário que, em vários casos, pode indicar que um sistema escalonável é dito não-escalonável. Na Secão 3.1 será analisada a causa do pessimismo desnecessário presente na análise de Shi e Burns. Na Seção 3.2 são propostas alterações na mesma para redução do pessimismo ainda mantendo a capacidade da análise de prover garantias temporais. Na Seção 3.3 é levantada uma discussão a respeito da não exatidão da análise de escalonamento e a complexidade computacional envolvida em uma possível análise de escalonamento exata para fluxos de comunição. O modelo de sistema utilizado nesta dissertação é apresentado e discutido na Seção 3.4. Para isso, o conceito de atraso de liberação efetivo é introduzido, demonstrando que o atraso de liberação provocado por tarefas geradoras nos seus fluxos de comunicação impacta no comportamento temporal dos fluxos de duas formas distintas. Finalmente, na Seção 3.5 é feita uma revisão dos resultados obtidos durante este capítulo.

3.1 Pessimismo no Limite Superior

Conforme descrito anteriormente, apesar da Premissa (A), apresentada na Seção 2.2.3 desta dissertação, estar correta, sua utilização como limite superior para o atraso de interferência indireta na análise de escalonamento proposta por Shi e Burns introduz pessimismo desnecessário na análise de escalonamento. Esse pessimismo é evidenciado no Exemplo 1 descrito a seguir.

Exemplo 1. Considere um cenário com 3 fluxos de comunicação: F1(5,2,2),F2(5,0,1) e F3(8,0,3), de acordo com a Definição 2.1. Assume-se que F1 interfere diretamente com F3 e F2 interfere diretamente com F1, distribuídos conforme a Figura 2. F2 é o fluxo de mais alta prioridade, seguido por F1 e F3 respectivamente. Calculando o *response time* do fluxo F3 de acordo com o modelo de análise mencionado previamente, obtém-se:

$$W(3)^0 = 3$$

$$W(3)^1 = 3 + \left\lceil \frac{3 + (2) + (5 - 2)}{5} \right\rceil \times 2 = 7$$

$$W(3)^2 = 3 + \left\lceil \frac{7 + 2 + (5 - 2)}{5} \right\rceil \times 2 = 9$$

$$W(3)^3 = 3 + \left\lceil \frac{9 + 2 + (5 - 2)}{5} \right\rceil \times 2 = 9$$

$$R_3 = 9 + 0 = 9$$

De acordo com a análise de escalonamento acima, F3 perderia seu prazo por uma unidade de tempo. Porém, se for realizada uma análise gráfica da interferência entre os fluxos em uma linha de tempo(Figura 3) pode-se notar que este resultado é desnecessariamente pessimista.

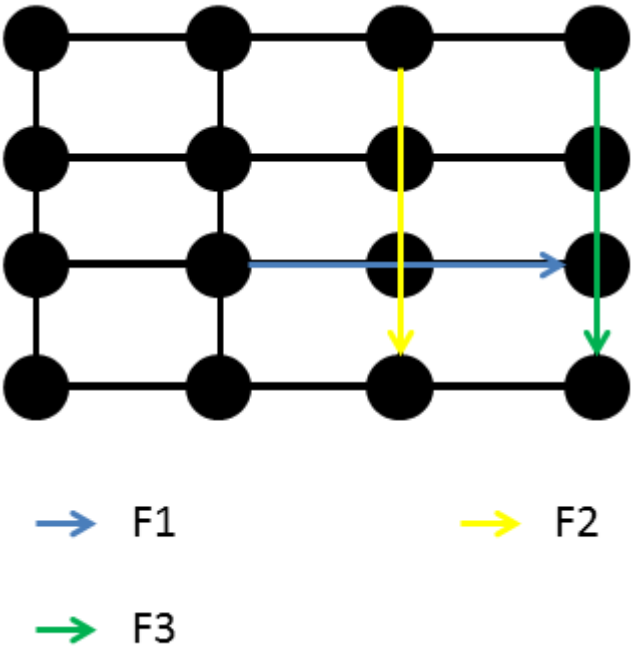

Figura 2: Distribuição de fluxos para o Exemplo 1.

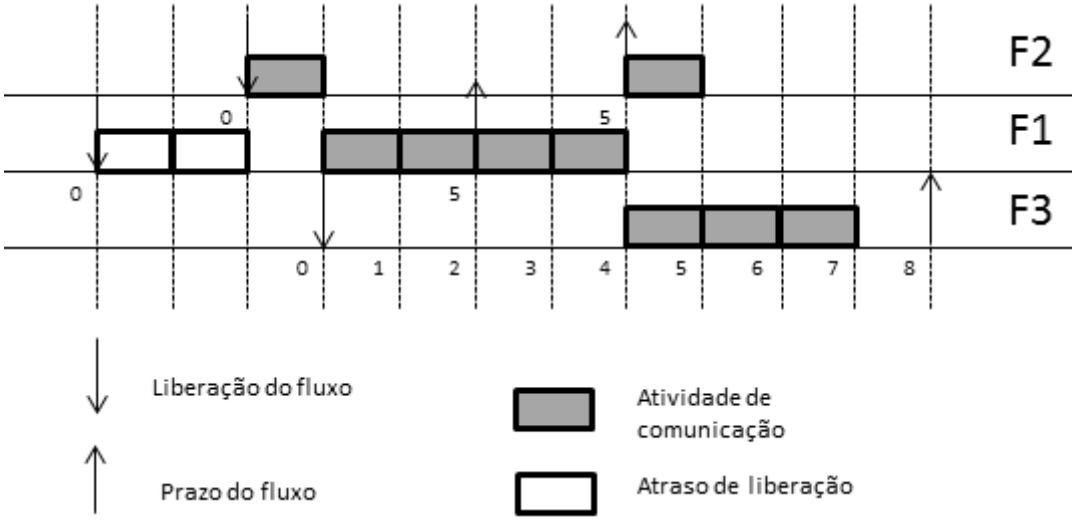

Figura 3: Interpretação Gráfica do Response Time do Fluxo F3.

O exemplo mostra que o pior tempo possível para o término das atividades do fluxo F3 seria 7 e não 9, como resultado pela análise de escalonamento. Isso acontece devido ao pessimismo introduzido pela Premissa (A).

A razão pela qual a utilização da Premissa (A) resulta em uma análise pessimista é que tal premissa utiliza o *response time* do fluxo para calcular seu atraso de interferência e depois soma o mesmo ao atraso de liberação para calcular o atraso total. O problema deste cálculo está no fato de que o atraso de liberação de um fluxo já está incluso no seu *response time* e, ao somar o limite superior do atraso de interferência ao próprio atraso de liberação, resulta na dupla consideração do atraso de liberação. Baseado em tal observação, as próximas seções têm como objetivo a substituição da Premissa (A) por premissas menos pessimistas.

3.2 Reduzindo o Pessimismo

Considerando o pessimismo descrito na seção anterior devido à Premissa (A) (Seção 2.2.3), é possível reduzir o pessimismo da dupla consideração do atraso de liberação especificando uma nova Premissa para o cálculo de *response time*:

$$J_i^R + J_i^I \leq R_i - C_i \qquad \text{(B)}$$

Essa premissa leva em consideração que, na janela de tempo $W(i)$ é considerada a interferência máxima sofrida pelo fluxo i no decorrer de sua execução. Dessa forma, o atraso de liberação do fluxo i é somado a esta janela para resultar no seu *response time*. É possível afirmar que o termo $R_i - C_i$ é o limite superior para o atraso de interferência total do fluxo.

Assim, se a Premissa (B) for utilizada na Equação (1), descrita na Seção 2.2.2, propõe-se a seguinte modificação para a equação:

$$W(i)^k = C_i + \sum_{j \in Hp(i)} \left\lceil \frac{W(i)^{K-1} + R_j - C_j}{T_j} \right\rceil \times C_j \quad (3)$$

O termo $(J_i^R + J_i^I)$ é substituído por $(R_j - C_j)$ e o cálculo de R_i permanence inalterado. Se o *response time* do fluxo F3 do exemplo anterior for calculado através da Equação (3), será obtido o seguinte resultado:

$$W(3)^0 = 3$$

$$W(3)^1 = 3 + \left\lceil \frac{3 + (5-2)}{5} \right\rceil \times 2 = 7$$

$$W(3)^2 = 3 + \left\lceil \frac{7 + (5-2)}{5} \right\rceil \times 2 = 7$$

$$R_3 = 7 + 0 = 7$$

Como pode ser observado, os resultados da análise de escalonamento através da Equação (3) condizem com a interpretação gráfica presente na Figura 1, que descreve o pior caso exato para o Exemplo 1.

Apesar do fato da Equação (3) ter alcançado um resultado exato para o Exemplo 1, eliminando o pessimismo desnecessário da análise com relação ao mesmo, a análise resultante ainda possui pessimismo desnecessário. Tal pessimismo desnecessário é ilustrado pelo Exemplo 2.

Exemplo 2. Considerando-se um cenário com 4 fluxos: F1 (5,2,1), F2 (5,0,1), F3 (5,0,1),F4 (8,0,3), distribuídos de acordo com a Figura 4. Sendo F3 o fluxo de mais alta prioridade, seguido por F2, F1 e F4 respectivamente. F2 interfere diretamente com F1, F1 interfere diretamente com F4 e F3 interfere diretamente com F2, F1, F4. Se for feita a análise de escalonamento do fluxo F4 pela Equação (3), os seguintes resultados são obtidos:

$$W(4)^0 = 3$$

$$W(4)^1 = 3 + \left\lceil \frac{3 + (5-1)}{5} \right\rceil \times 1 + \left\lceil \frac{3 + (1-1)}{5} \right\rceil \times 1 = 6$$

$$W(4)^2 = 3 + \left\lceil \frac{6 + (5-1)}{5} \right\rceil \times 1 + \left\lceil \frac{6 + (1-1)}{5} \right\rceil \times 1 = 7$$

$$W(4)^3 = 3 + \left\lceil \frac{7 + (5-1)}{5} \right\rceil \times 1 + \left\lceil \frac{7 + (1-1)}{5} \right\rceil \times 1 = 8$$

$$W(4)^4 = 3 + \left\lceil \frac{8 + (5-1)}{5} \right\rceil \times 1 + \left\lceil \frac{8 + (1-1)}{5} \right\rceil \times 1 = 8$$

$$R_4 = 8 + 0 = 8$$

De acordo com a análise de escalonamento modificada, F4 finalizaria suas atividades de comunicação no tempo 8. Porém, se for construída a interpretação gráfica da interferência entre os fluxos do Exemplo 2 (Figura 5), pode-se notar que o resultado da análise não é exato.

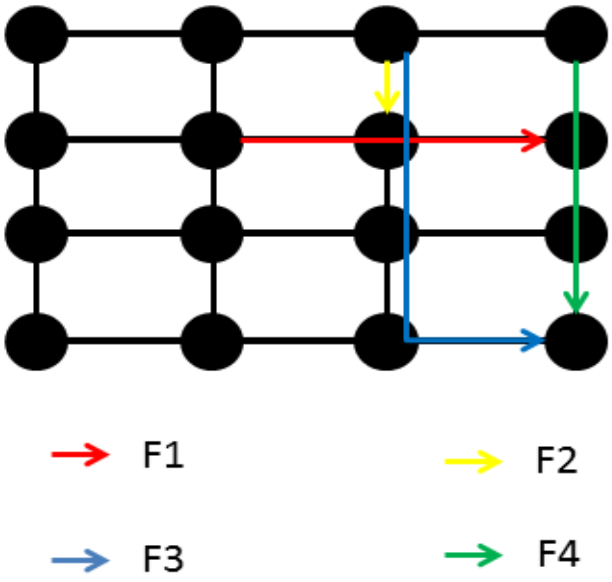

Figura 4: Distribuição de fluxos para o Exemplo 2.

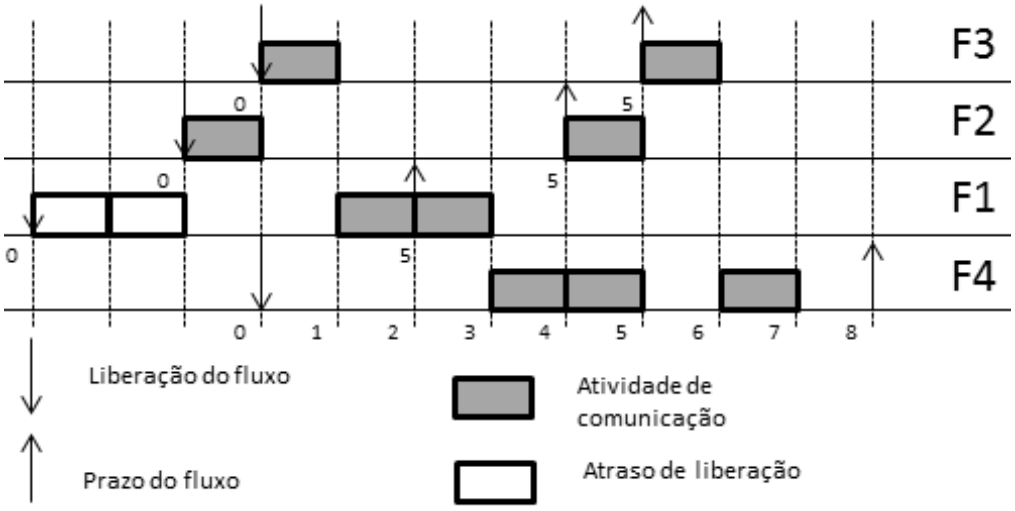

Figura 5: Interpretação Gráfica do Response Time do Fluxo F4.

O resultado pessimista obtido através da análise pela Equação (3), é causado pela utilização do limite superior do atraso total obtido através da Premissa (B). Este limite superior obteve um resultado exato para o Exemplo 1, pois toda interferência sofrida pelo fluxo F1 (presente no seu *response time*) é proveniente de fluxos que não possuem uma relação de interferência com o fluxo F3 (fluxo sendo analisado). Porém, no Exemplo 2, o *response time* do fluxo F1 possui interferência do fluxo F3 que exerce interferência direta no fluxo F4 (fluxo sendo analisado). O limite superior proveniente da Premissa (B) utiliza a interferência direta do fluxo F3 como interferência indireta, resultando em um pessimismo desnecessário na Equação (3).

Na Equação (1), os termos para o atraso de interferência (J_i^I) e atraso de liberação (J_i^R) são considerados discriminadamente. O pessimismo mencionado durante este capítulo é introduzido pelo limite superior construído através da Premissa (A). Entretanto, é possível realizar o cálculo do limite superior do atraso de interferência considerando-se apenas fontes de interferência indireta. Tal cálculo é demonstrado na Equação (4).

$$J_{ji}^I = \sum_{k \in Hpi(j)} \left\lceil \frac{R_j - J_j^R + J_k^R + J_{kj}^I}{T_k} \right\rceil \times C_k \qquad (4)$$

Onde J_{ji}^I é o atraso de interferência do fluxo j quando analisando o *response time* do fluxo i, Hpi(j) é o conjunto de fluxos que interferem diretamente com o fluxo j e não possuem relação de interferência direta com o fluxo i, R_j é o *response time* do fluxo j, T_k é o período do fluxo k pertencente ao conjunto Hpi(i) e C_k é a latência do fluxo k, J_k^R é o atraso de liberação do fluxo *k* e J_{kj}^I é o atraso de interferência do fluxo k em relação ao fluxo j.

O atraso de liberação do fluxo j é desconsiderado pois apenas atrasos impostos após o atraso de liberação acrescentarão interferência direta, caso contrário, a interferência indireta se

sobreporá ao atraso de liberação e nenhuma alteração no atraso total do fluxo j será percebida. Ao incluir-se a Equação (4) na Equação (1), obtém-se a Equação (5), onde o atraso de interferência e o atraso de liberação são considerados isoladamente.

$$W(i)^k = C_i + \sum_{j \,\epsilon\, Hp(i)} \left\lceil \frac{W(i)^{K-1} + J_j^R + \sum_{Hpj(i)} \left\lceil \frac{R_j - J_j^R + J_k^R + J_{kj}^I}{T_k} \right\rceil \times C_k}{T_j} \right\rceil \times C_j \qquad (5)$$

Sendo que J_{kj}^I representa o atraso de interferência exercido no fluxo *k* em relação ao fluxo *j* e $Hpj(i)$ representa o conjunto de fluxos que interferem diretamente no fluxo *j*, mas não interferem diretamente no fluxo *i*.

Utilizando-se a Equação (5) para o cálculo do *response time* do fluxo F4, são obtidos os seguintes resultados:

$W(4)^0 = 3$

$$W(4)^1 = 3 + \left\lceil \frac{3 + 2 + \left\lceil \frac{5-2}{5} \right\rceil \times 1}{5} \right\rceil \times 1 + \left\lceil \frac{3+0+0}{5} \right\rceil \times 1 = 6$$

$$W(4)^2 = 3 + \left\lceil \frac{6 + 2 + \left\lceil \frac{5-2}{5} \right\rceil \times 1}{5} \right\rceil \times 1 + \left\lceil \frac{6+0+0}{5} \right\rceil \times 1 = 7$$

$$W(4)^3 = 3 + \left\lceil \frac{7 + 2 + \left\lceil \frac{5-2}{5} \right\rceil \times 1}{5} \right\rceil \times 1 + \left\lceil \frac{7+0+0}{5} \right\rceil \times 1 = 7$$

$R_4 = 7 + 0 = 7$

Através da obtenção do limite superior para o atraso de interferência, divididas em interferência direta e indireta, a Equação (5) obteve o *response time* exato para o Exemplo 2. Como pode ser observado nos cálculos a seguir, a Equação (5) também obtém um resultado exato para o Exemplo 1.

$W(3)^0 = 3$

$$W(3)^1 = 3 + \left\lceil \frac{3 + 2 + \left\lceil \frac{5-2}{5} \right\rceil \times 1}{5} \right\rceil \times 2 = 7$$

$$W(3)^2 = 3 + \left\lceil \frac{7 + 2 + \left\lceil \frac{5-2}{5} \right\rceil \times 1}{5} \right\rceil \times 2 = 7$$

$$R_3 = 7 + 0 = 7$$

A Equação (5) foi capaz de obter resultados exatos para ambos os exemplos, sendo considerada uma abordagem válida para ambos os problemas apresentados neste Capítulo.

3.3 Análise do tempo exato de término dos fluxos de comunicação

Apesar de ter obtido resultados exatos para ambos os exemplos posteriores, a última análise apresentada ainda não pode ser considerada uma análise exata. A análise considera a interferência total causada por fluxos de interferência indireta no intervalo de execução de um fluxo de interferência direta. Esta pressuposição pode levar a um pessimismo desnecessário para certos casos. O Exemplo 3 ilustra como tal pressuposição pode resultar em um pessimismo desnecessário.

Exemplo 3. Considerando-se um cenário com 4 fluxos: F1(7,1,2), F2(3,0,1), F3(3,0,1), F4(8,0,3), sendo F3 o fluxo de mais alta prioridade, seguido por F2, F1 e F4. F2 e F3 interferem diretamente com o fluxo F1 e F1 interfere diretamente com o fluxo F4, como demonstrado pela Figura 6. Calculando-se o atraso de interferência do fluxo F1 em relação ao escalonamento do fluxo F4 (interferência indireta infligida em F1 através da interferência direta dos fluxos F2 e F3 no fluxo F1), é obtido o seguinte resultado:

$$J_{F1}^I = \left\lceil \frac{7-1}{3} \right\rceil \times 1 + \left\lceil \frac{7-1}{3} \right\rceil \times 1$$

$$J_{F1}^I = 4$$

O cálculo do atraso de interferência através da Equação (4) considera que toda interferência sofrida pelo fluxo, oriunda de fluxos com relação de interferência indireta com o fluxo escalonado, se apresenta em um bloco contínuo de tempo seguido pela execução ininterrupta do fluxo sofrendo tal interferência. Tal condição pode ou não se apresentar,

dependendo do padrão de interferência sofrido pelo fluxo, possibilitando a ocorrência de resultados desnecessariamente pessimistas provenientes da análise de escalonamento pela Equação 5. Analisando graficamente o comportamento temporal dos fluxos do Exemplo 3 (Figuras 7 e 8), pode-se concluir que o cálculo do atraso de interferência pela Equação (4) obteve um resultado não exato.

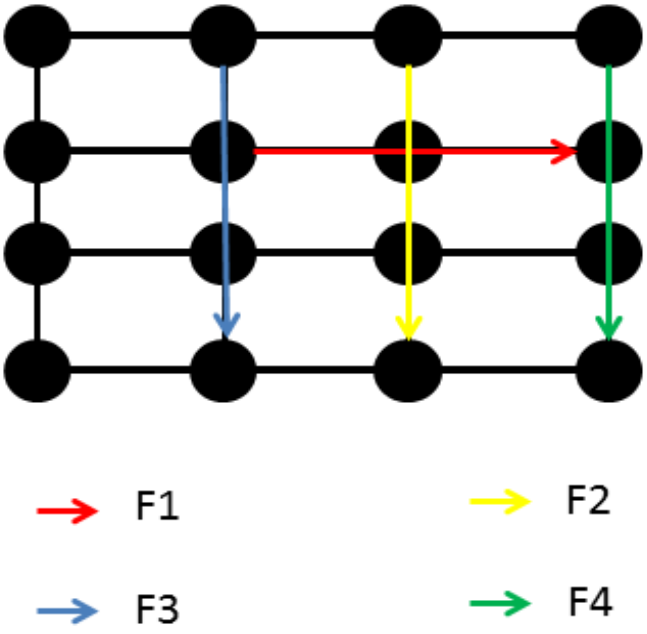

Figura 6: Distribuição de fluxos para o Exemplo 3.

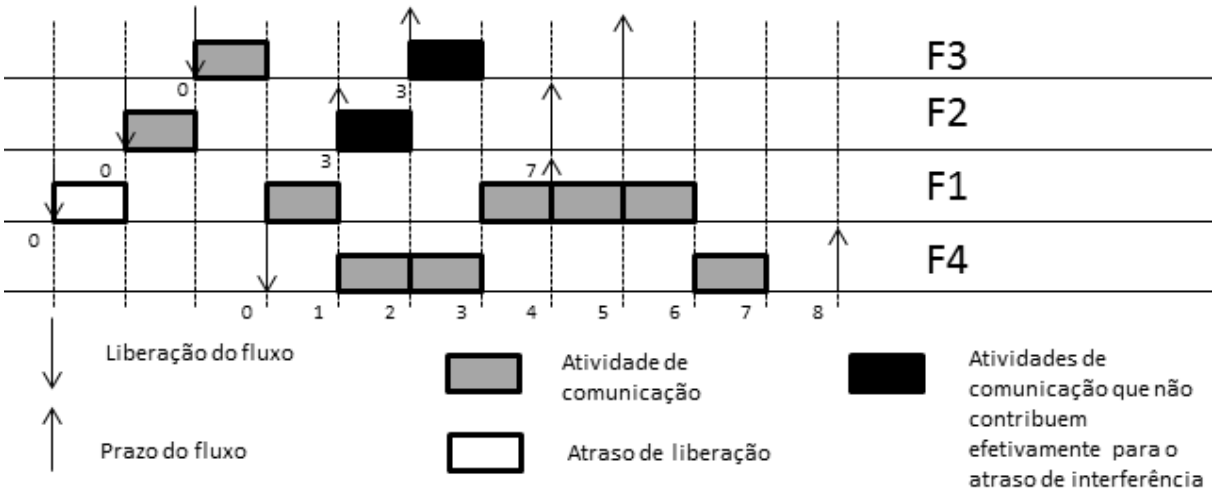

Figura 7: Análise do comportamento temporal dos fluxos do exemplo 3.

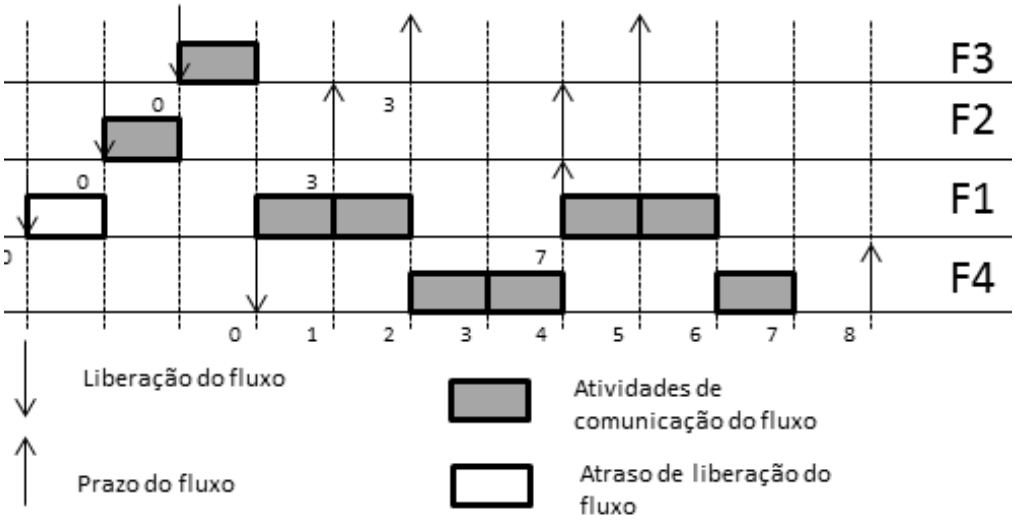

Figura 8: Comportamento assumido pelos fluxos referente ao pior caso de atraso de interferência.

Observando-se a Figura 7, nota-se que a interferência sofrida pelo fluxo F1, oriunda dos fluxos F2 e F3, no período de atividade do fluxo F1, não se apresenta na forma de um bloco de tempo ininterrupto. A presença de um *slot* de tempo entre liberações dos fluxos causadores de interferência possibilita que o fluxo F1 inicie suas atividades no tempo 3, transmitindo por uma unidade de tempo até a próxima liberação dos fluxos interferentes. Como o momento crítico prevê que o fluxo analisado (F4) deve ser liberado juntamente com o início das atividades de todos os fluxos que possuam uma relação de interferência direta, F4 é liberado juntamente com o início das atividades de F1. Nesse cenário o atraso de interferência efetivo sofrido pelo fluxo F1 consiste nas duas primeiras unidades de tempo ocupadas ininterruptamente por fluxos com relação de interferência indireta com F4.

Em casos onde o impacto total de fluxos de interferência indireta não se apresenta na forma de um bloco contínuo de tempo, o maior impacto possível oriundo de fontes de interferência direta ocorrerá caso os fluxos fontes de tal interferência não sejam liberados após o término do primeiro bloco ininterrupto de tempo (ocorrência do primeiro *slot* de tempo livre). Como ilustrado na Figura 8, o maior impacto possível devido ao atraso de interferência ocorrerá se os fluxos F2 e F3 não forem liberados após seus primeiros períodos de atividade, pois dessa forma, o fluxo F4 sofrerá dois *hits* do fluxo F1 em um intervalo de tempo menor que o período de F1. Se considerarmos que o fluxo F4 possui uma latência de 2 ao invés de 3, no comportamento mostrado na Figura 3, seu *response time* seria de 3 unidades de tempo, enquanto, no cenário da Figura 4 seria de 4 unidades de tempo.

No caso do Exemplo 2, é possível a obtenção do valor exato do atraso de interferência de um fluxo através de uma nova análise de escalonamento do mesmo, considerando apenas fontes de interferência indireta (em relação ao fluxo sendo escalonado). Tal análise deve ser realizada considerando que o fluxo possua uma latência unitária. Ao se realizar uma análise

de escalonamento com tais condições, pode-se determinar a localização do primeiro *slot* de tempo livre no padrão de interferência indireta do fluxo F1.

Apesar de o "*slot"* livre ser relativamente simples de ser obtido para o Exemplo 2, tal exemplo possui interferência direta em apenas um nível. No entanto, para a obtenção de um resultado exato para a interferência indireta sofrida por um fluxo, é necessária a análise de cada camada de interferência que compõe a interferência total sofrida pelo fluxo. Caso existissem fluxos que interferissem com F2 e F3, seria necessária uma análise de "*slot"* para os fluxos F2 e F3 considerando tais interferências para que seja possível uma análise de "slot" exata para o fluxo F1. Tal análise recursiva gera um problema de decisão: qual padrão de interferência indireta deve ser apresentado para se maximizar a interferência no fluxo, no qual está sendo analisado o *response time*? Para ilustrar o problema, considera-se um cenário com quatro fluxos dispostos como demonstrado pela Figura 9. Considera-se a prioridade de cada um dos fluxos apresentados na figura seguinte sendo inversamente proporcional ao seu índice (F1 tem a mais alta prioridade, seguido por F2 e assim sucessivamente).

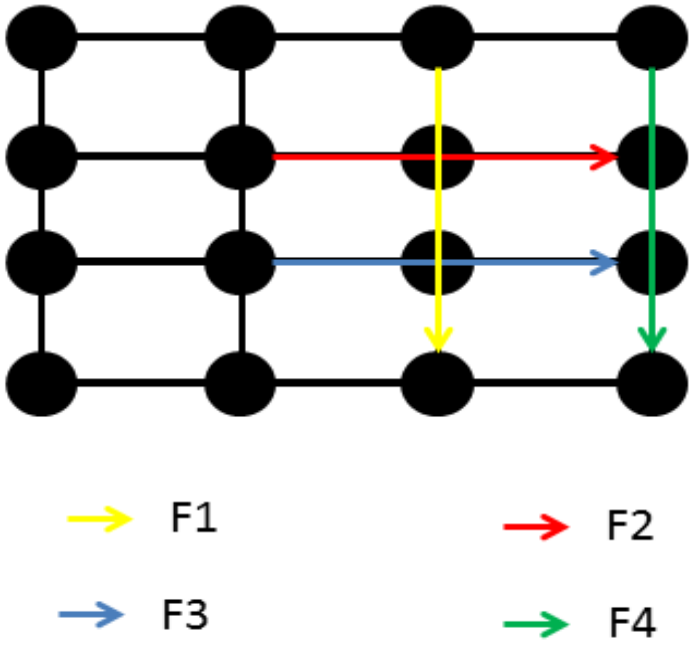

Figura 9: Distribuição inicial de fluxos para o problema de decisão.

Supondo que esteja sendo realizada a análise do *response time* do fluxo F4 no exemplo apresentado pela Figura 9. Para que a interferência direta exata oriunda dos fluxos F3 e F2 seja calculada, deve-se definir a interferência indireta exata exercida pelo fluxo F1 nos fluxos F3 e F2, algo que deveria ser obtido por uma análise de "*slot"* dos fluxos F3 e F2. Porém, para que o cálculo seja exato, deve ser definido qual será o padrão de interferência do fluxo F1. Tal fluxo interferirá integralmente em F3, em F2 ou em ambos os fluxos simultaneamente?

Este questionamento é necessário, pois para que o fluxo F1 interfira integralmente na forma de atraso de interferência indireta para um dos dois fluxos em questão, ele não poderá ser liberado após a janela de atraso máximo para cada um desses fluxos (obtida através da análise de "*slot*"). Portanto, dependendo da dimensão da janela máxima de atraso dos fluxos F2 e F3, será possível que o fluxo F1 haja integralmente como atraso para apenas um dos fluxos. A satisfação da condição de atraso integral de um dos fluxos resultará em *releases* a mais ou a menos para a condição de atraso integral do outro fluxo. Mais especificamente, caso o limite superior da divisão da janela de atraso máximo de F3 pelo período do fluxo F1 seja diferente do resultado do mesmo cálculo para o fluxo F2, não será possível a satisfação da condição de atraso integral para ambos os fluxos. Levando em consideração tais condições, deve-se decidir qual dos fluxos deverá receber o atraso integral.

A decisão de qual dos fluxos deverá receber o atraso integral deve ser feita de forma a obter-se a maior interferência possível no fluxo F4, ou seja, deve-se analisar quais padrões de interferência farão com que os fluxos F3 e F2 exerçam a maior interferência conjunta possível. Para determinar o atraso máximo de cada um dos fluxos para cada um dos padrões de interferência, será necessária uma análise de escalonamento do fluxo F4 para cada um dos padrões de interferência. A determinação do padrão de interferência mais intenso agregará grande complexidade à análise.

Considere um cenário onde existam mais fluxos interferindo em F3 e F2 ou mais fluxos interferindo em F1. Nesse caso, a complexidade de uma análise exata da interferência indireta é agravada devido à necessidade da análise em múltiplos níveis. Para que o problema de análise de padrões de interferência em múltiplos níveis seja ilustrado, considera-se um cenário com sete fluxos, demonstrado pela Figura 10. O mesmo esquema de prioridades do exemplo anterior será considerado para este exemplo.

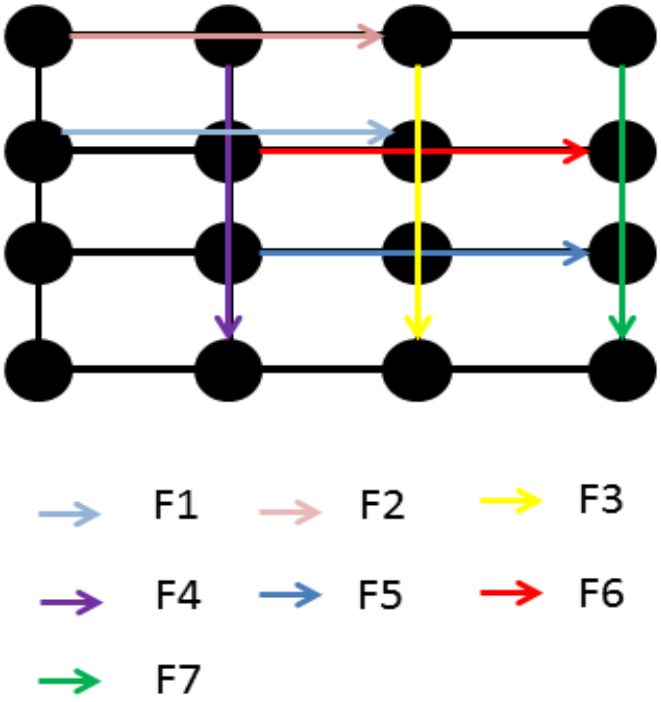

Figura 10: Distribuição de fluxos final para o problema de decisão.

Se a análise do *response time* de F7 for levada em consideração, um novo nível de interferência é adicionado, a interferência dos fluxos F1 e F2 nos fluxos F3 e F4. Para que a máxima interferência possível dos fluxos F6 e F5 no fluxo F7 sejam determinadas, será necessária a análise dos padrões de interferência dos fluxos F4 e F3 nos fluxos F6 e F5 e para que a máxima interferência possível dos fluxos F4 e F3 seja determinada, a máxima interferência possível dos F2 e F1 nos fluxos F4 e F3 deve ser determinada. Portanto, a análise se depara com dois problemas de decisão: Qual será o padrão de interferência dos fluxos F4 e F3 (Decisão1)? Qual será o padrão de interferência dos fluxos F2 e F1 (Decisão 2)?

O problema se torna mais complexo considerando que o resultado da Decisão 2 afetará o resultado da Decisão 1 e, portanto, para a obtenção de garantias temporais, deve-se analisar todas as combinações possíveis de configurações da Decisão 1 e da Decisão 2. Portanto, para que o *response time* exato de F7 seja calculado, é necessário, primeiramente, que sejam analisados os possíveis padrões de interferência dos fluxos F2 e F1 nos fluxos F4 e F3. Neste exemplo existem 4 padrões de interferência distintos: (i) F2 e F1 exercem interferência plena em F3; (ii) F2 e F1 exercem interferência plena em F4; (iii) F2 exerce interferência plena em F3 e F1 exerce interferência plena em F4; (iv) F1 exerce interferência plena em F3 e F2 exerce interferência plena em F4. Deve ser feita uma análise de *slot* para se determinar o atraso de interferência exata para cada fluxo em cada um dos padrões de interferência (4 padrões e 2 fluxos resultando em 8 análises). Após os atrasos de interferência exatos dos fluxos F3 e F4 terem sido obtidos, deve ser realizada a análise de todos os padrões possíveis

de interferência de F3 e F4 em F5 e F6, totalizando quatro padrões distintos. Entretanto, como mencionado anteriormente, a Decisão 2 afeta o resultado da Decisão 1, portanto, deve-se ser analisado o impacto de cada alternativa possível para Decisão 2 em cada um dos padrões de interferência referentes a Decisão 1. A Decisão 1 possui 4 padrões de interferência distintos que devem ser analisados em relação a cada uma das 4 alternativas da Decisão 2, resultando em 16 possibilidades. Deve ser realizada a análise de *slot* de cada fluxo para cada uma das alternativas (2 fluxos e 16 alternativas resultando em 32 análises). Finalmente, deve-se determinar qual das 16 possíveis combinações de atraso de liberação geram o atraso mais severo (resultam em *back to back hits*) podendo variar entre 1 e 16 análises de *response time* para o fluxo F7. Como pode ser observado, mesmo em um cenário limitado, constituído de sete fluxos, podendo ser necessárias um total de 56 análises com a mesma complexidade de uma análise de *response time*, tornando o procedimento extremamente custoso.

Para a obtenção do valor exato do atraso de interferência de um fluxo, deve-se fazer uma nova análise de escalonamento do mesmo, considerando-se apenas fontes de interferência indireta (em relação ao fluxo sendo escalonado). Tal análise deve ser realizada considerando que o fluxo possua uma latência igual a 1. Ao se realizar uma análise de escalonamento com tais condições, pode-se determinar a localização do primeiro *slot* de tempo livre no padrão de interferência indireta dos fluxos. Apesar de esta abordagem ser possível, ela resultará em um aumento significativo na complexidade da análise de escalonamento, pois para cada fluxo sendo analisado, deverá ser realizada uma nova análise de escalonamento que possua fontes de interferência indireta em relação ao fluxo sendo analisado. Análises de *slot* não poderão ser reaproveitadas entre análises de escalonamento, pois o padrão de interferência indireta possivelmente se modificará de fluxo para fluxo.

3.4 Modelo de Sistema

No modelo computacional utilizado neste trabalho, considera-se que relações de dependência de dados se manifestam na forma de uma relação de produtor-consumidor entre duas tarefas. É considerado que um fluxo de comunicação será lançado apenas após o término da execução de sua tarefa geradora, sempre sendo encaminhado para o processador no qual a tarefa destinatária inicia sua execução. Em tal modelo, duas implicações devem ser consideradas: (i) o impacto do tempo de execução da tarefa geradora no tempo de recepção do seu fluxo; (ii) o impacto do tempo de chegada de tal fluxo na tarefa receptora.

Para que o impacto do tempo de recepção do fluxo na tarefa receptora seja considerado de forma exata, deve-se estabelecer o prazo máximo para a chegada do fluxo de comunicação ao seu destino, implicando na criação de uma nova análise de escalonamento dependente do resultado de duas outras análises de escalonamento (da tarefa receptora e do fluxo gerado). Se a possibilidade de múltiplos fluxos destinarem-se para uma mesma tarefa

for considerada, é possível perceber que tal análise pode se tornar muito complexa. Tendo em vista a redução da complexidade da análise, o impacto do tempo de recepção de uma comunicação será quantificado através de um prazo fixo pré-estabelecido para tal comunicação. É considerado que a perda de prazo de chegada por parte de um fluxo de comunicação, possui a mesma severidade de uma perda de prazo por parte de uma tarefa, portanto, um sistema necessita que todas as suas tarefas e fluxos cumpram seus prazos para que o mesmo seja considerado escalonável.

Apesar da utilização de prazos dinâmicos para fluxos aumentar significativamente a complexidade da análise de escalonamento de fluxos, o impacto do término da execução da tarefa geradora no seu fluxo de comunicação pode ser analisado com uma complexidade similar a análise de escalonamento de uma tarefa. Um primeiro passo intuitivo seria agregar o tempo de execução da tarefa geradora ao tempo de chegada do fluxo de comunicação. O tempo de resposta da tarefa geradora é exercido como atraso no fluxo de comunicação gerado pela mesma. Porém, este atraso não se comporta da mesma forma que o atributo atraso de liberação (J_i^R) previsto no modelo de análise de escalonamento apresentado em (Shi e Burns, 2010). A análise de escalonamento de tarefas com atraso proposto em (Audsley *et al*, 1995), no qual o modelo de análise de Shi e Burns se baseia, considera que o impacto do atraso de liberação de uma tarefa se dá de duas formas: (i)através do chamado "*back to back hit"* de tarefas interferentes e, (ii)impondo um atraso no término da tarefa que sofre do atraso de liberação. Para que tarefas interferidas sofram o "*back to back hit*", ou seja, recebam dois *hits* da tarefa interferente em um intervalo de tempo menor que o seu período, a tarefa interferente deve sofrer o máximo atraso de liberação possível na sua primeira liberação e nenhum atraso na sua liberação seguinte. Porém, o tempo de resposta de uma tarefa não se comporta como o atraso de liberação previsto pela análise de Shi e Burns, pois, para que seu fluxo gerado não sofra atraso na sua segunda liberação, a tarefa não poderia executar a sua segunda liberação (seu tempo de execução deve ser nulo).

Uma vez que o tempo de resposta de uma tarefa não pode ser considerado na sua totalidade como atraso de liberação para um fluxo, deve-se levar em consideração que existirão variações no tempo de execução de uma tarefa no decorrer do funcionamento do sistema, pois o tempo de resposta fornecido pela análise de *response time* considera que a tarefa encontrará as piores condições possíveis durante sua execução e não ocorrerá durante todas liberações da tarefa. O atraso de liberação considerado na análise em (Audsley *et al*, 1995) trata-se da variação entre o maior atraso possível e o menor atraso possível de serem sofridos por uma tarefa. Portanto, o fato que, muito embora uma tarefa nem sempre exiba seu pior tempo de execução, deve ser levado em consideração que ela nunca exibirá um tempo nulo ao se determinar o atraso de liberação efetivo.

Portanto, para determinar o atraso de liberação efetivo sofrido por um fluxo em função do tempo de término da sua tarefa geradora, é necessário não apenas conhecer o pior tempo possível de execução de uma tarefa (C), mas também conhecer seu tempo mínimo de execução (C^{min}). A análise de *response time* utiliza-se do conhecimento do pior tempo de execução possível de uma tarefa sem interferência de outras tarefas (C), normalmente obtido

através de simulações ou análise da estrutura do código da tarefa juntamente com a arquitetura do sistema (Davis e Burns, 2011). Também é possível obter o tempo mínimo de execução de uma tarefa sem interferência que pode ser utilizado como limite inferior para o melhor tempo possível de execução de uma tarefa. Apesar de ser possível a utilização do tempo mínimo de execução da tarefa como um limite inferior para o tempo de resposta da tarefa geradora, a utilização do mesmo introduziria pessimismo na análise de escalonamento de fluxos, pois só é possível que a tarefa exiba este tempo de resposta se a mesma for a única tarefa no processador.

Em (Redell e Sanfridson, 2002), é proposta uma análise exata para o melhor tempo de resposta possível para uma tarefa considerando interferência proveniente de outras tarefas. Através desta análise é possível obter o tempo de resposta de uma tarefa no melhor caso possível. A análise proposta por Redell e Sanfridson utiliza o conceito contrário ao momento crítico, tratando-se de uma janela de tempo entre liberações de tarefas interferentes onde a tarefa sendo analisada possa executar sofrendo a menor interferência possível (idealmente, terminando sua execução antes de receber *hits* de tarefas interferentes). Para simulação de tal janela de tempo é utilizada a Equação (4).

$$r_i^b = C_i^{min} + \sum_{Hp(i)} \left\lceil \frac{r_i^b - T_j - J_j}{T_j} \right\rceil \times C_j^{min} \quad (4)$$

A Equação (4) considera que haverá um *hit* de uma tarefa interferente apenas após a tarefa interferente ter sofrido o máximo atraso de liberação possível (J_j) e necessitará da mínima computação possível para terminar sua execução, também impondo que a tarefa interferente não exercerá *back to back hits*. Através da obtenção do melhor tempo de resposta possível (r_i^b), o atraso de liberação efetivo sofrido por um fluxo de comunicação pode ser calculado através da diferença entre o pior tempo de resposta possível e do menor tempo de resposta possível de uma tarefa, como demonstrado pelo cenário a seguir. Neste cenário, o fluxo F2(9,0,1) é interferido pelo fluxo F1(5,2,1), sendo que o fluxo F1 possui um atraso de liberação mínimo de 1 (Figuras 11 e 12).

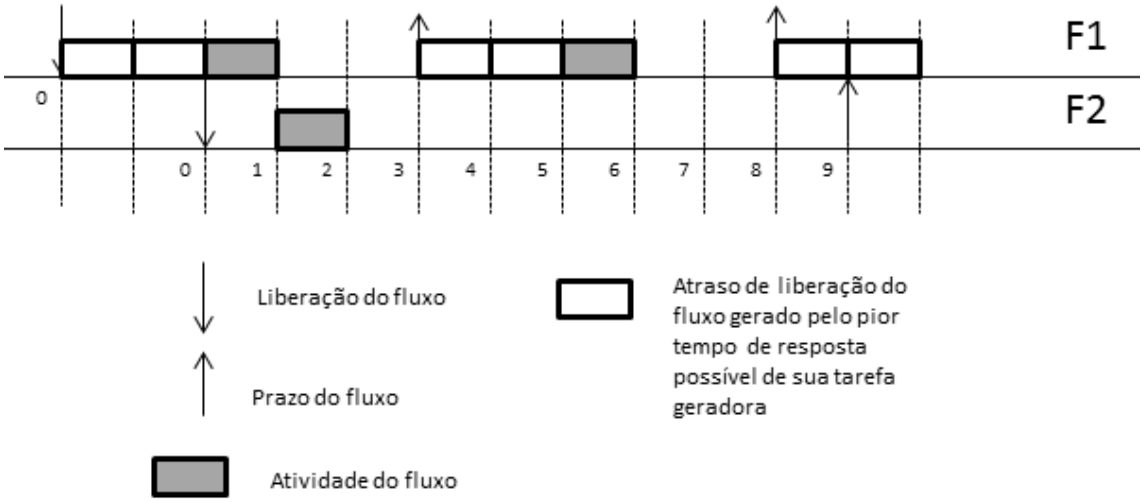

Figura 11: Comportamento temporal assumido por um fluxo sofrendo um atraso de liberação constante.

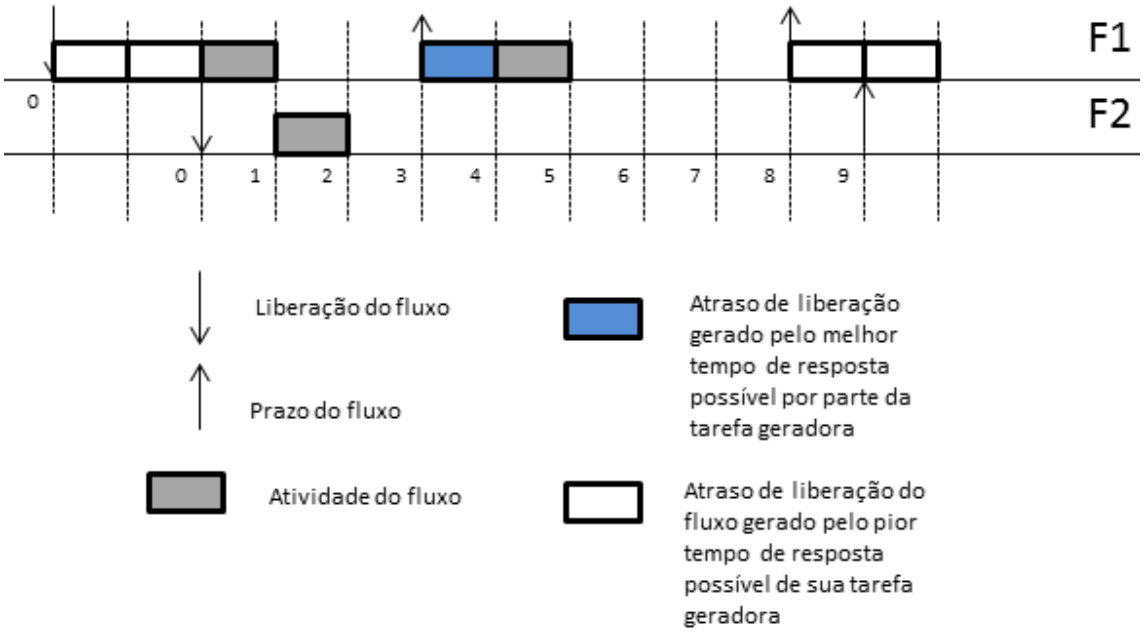

Figura 12: Comportamento temporal assumido por um fluxo sofrendo um atraso de liberação variável.

Na Figura 11, a tarefa geradora do fluxo F1 exibe um tempo de resposta constante (resultando em um atraso de liberação constante), causando com que o fluxo F1 dê seu primeiro *hit* em F2 no tempo 0 e o segundo *hit* no tempo 5. Neste primeiro caso o fluxo F2 recebe dois *hits* do fluxo F1 em um intervalo de 5 unidades de tempo que corresponde ao período do fluxo F1. Porém, na Figura 12, o fluxo F1 sofre um atraso de liberação menor na sua segunda liberação devido ao fato de sua tarefa geradora exibir o melhor tempo de resposta possível. Nestas condições, o fluxo F2 recebe o primeiro *hit* em 0 e o segundo *hit* em 4,

recebendo dois hits de F1 em um intervalo uma unidade de tempo menor que o período do fluxo F1(atraso efetivo).

Apesar de apresentar uma alternativa exata para o cálculo do melhor tempo de resposta de uma tarefa, a abordagem de (Redell e Sanfridson, 2002) considera que todas as tarefas serão estritamente periódicas. Tal pressuposição introduz pessimismo na análise em sistemas que possuem tarefas esporádicas. Entretanto, utilizando o tempo mínimo entre *releases* (*MIT*), pode-se considerar que uma tarefa esporádica funciona como uma tarefa periódica no pior caso (Wilhelm *et al,* 2008). Portanto, é assumido que tarefas esporádicas comportam-se como periódicas, exibindo um período igual ao tempo mínimo entre duas liberações da mesma. Para uma análise de *response time* original, deve-se utilizar tal estratégia, pois trata-se de uma análise pessimista, devendo assumir que o sistema exibirá o pior comportamento temporal possível. Porém, para uma análise otimista como a de Redell e Sanfridson, deve-se assumir que o sistema exibirá o melhor comportamento temporal possível, tornando a pressuposição (tarefas esporádicas se comportarão como tarefas periódicas exibindo sempre o menor tempo entre *releases*) uma ideia divergente do objetivo da análise.

Com o objetivo de possibilitar que a análise de melhor tempo de resposta considere sistemas que possuam tarefas esporádicas, neste trabalho, será introduzido o conceito de tempo máximo entre *releases* de uma tarefa. É assumido que no decorrer do mesmo estudo realizado para se obter o tempo mínimo entre *releases* de uma tarefa esporádica, também é possível derivar o tempo máximo entre *releases* de uma tarefa. Também é considerada a hipótese de o tempo máximo entre *releases* não poder ser derivado para algumas tarefas, nesse caso, devemos assumir que a tarefa esporádica interferente exibirá o melhor comportamento temporal possível, ou seja, ela não sofrerá *release* durante a execução da tarefa sendo analisada.

O segundo comportamento do atraso de liberação (atraso imposto no tempo de execução da tarefa que sofre o atraso de liberação) possui um comportamento diferente e não necessita da análise de melhor tempo de execução para ser definido. O atraso do pior tempo de execução deve ser considerado na sua totalidade no tempo de resposta do fluxo gerado por tal tarefa. Portanto, o tempo de resposta de um fluxo receberá um atraso igual ao pior tempo de resposta de sua tarefa geradora.

Primeiramente, para que os conceitos discutidos nesse capítulo sejam introduzidos na análise gerada neste capítulo, deve-se modificar o cálculo do atraso de liberação do fluxo para que o atraso infligido por sua tarefa geradora seja levado em consideração. O cálculo do novo atraso de liberação do fluxo é exibido pela equação:

$$J_i^{R'} = J_i^R + r_i - r_i^b \qquad (6)$$

Onde $J_i^{R'}$ é o atraso de liberação ajustado do fluxo de comunicação i, r_i^b é o melhor tempo de resposta da tarefa geradora e r_i é o pior tempo de resposta da tarefa geradora.

O atraso de liberação reajustado deve substituir o atraso de liberação na Equação (6), resultando na seguinte equação:

$$W(i)^k = C_i + \sum_{Hp(i)} \left\lceil \frac{W(i)^{K-1} + J_j^{R\prime} + \sum_{Hpj(i)} \left\lceil \frac{R_j - J_j^{R\prime} + J_k^{R\prime} + J_{kj}^{I}}{T_k} \right\rceil \times C_k}{T_j} \right\rceil \times C_j \qquad (7)$$

Finalmente, o atraso de liberação real causado pela tarefa geradora deve ser introduzido na Equação (2), gerando a equação a seguir:

$$R_i = W(i) + r_i \qquad (8)$$

Onde r_ié o tempo de resposta da tarefa geradora do fluxo de comunicação *i*.

Através das modificações propostas no cálculo do atraso de liberação propostas neste capítulo, é possível a obtenção de garantias temporais para os fluxos do sistema com um pessimismo reduzido em relação à utilização do pior tempo de resposta da tarefa como atraso de liberação. Como a variação máxima entre tempos de execução de uma mesma tarefa é obtida através da diferença entre o pior tempo de resposta e o melhor tempo de resposta de uma tarefa, sua utilização para a previsão ocorrência "*back to back hits*" refletirá o pior caso mais possível (maior atraso efetivo possível) para tal fenômeno.

3.5 Considerações Finais

Neste Capítulo foi feita uma revisão dos conceitos da análise de escalonamento proposta por Shi e Burns. Durante tal revisão, foi identificado que a Premissa (A), utilizada como limite superior para o atraso de interferência dos fluxos, introduzia pessimismo desnecessário na análise. Foi proposta a adoção da Premissa (B) como limite superior para o atraso de interferência dos fluxos gerando uma redução no pessimismo da análise. Também foi definido que mesmo o cálculo do limite superior através da Premissa (B) não resulta em uma análise exata. Um novo cálculo para o atraso de interferência foi introduzido através da Equação (5), visando à consideração de apenas fontes de interferência indireta no cálculo do atraso de interferência. O cálculo da interferência através da Equação (5) gerou uma redução no pessimismo em relação a utilização da Premissa (B) como limite superior. Foi demonstrado que, apesar de uma redução no pessimismo da análise, a análise resultante de todas as modificações ainda não se trata de uma análise exata, porém, para a obtenção de uma análise exata seria necessária agregar grande complexidade a análise de escalonamento. Finalmente, foi proposta uma abordagem para que a análise de escalonamento de fluxos considere os dois comportamentos distintos do atraso de liberação gerador pela tarefa geradora no seu fluxo de comunicação, resultando na Equação (7).

4. Análise de escalonamento dos fluxos de comunicação em sistemas semi-particionados

No Capítulo 3 foi discutido o impacto gerado pelo tempo de execução de uma tarefa no fluxo de comunicação gerado pela mesma. Quando é considerado um modelo de escalonamento de tarefas que permita migrações, surge um novo fator de impacto no comportamento temporal dos fluxos: o comportamento migratório de tarefas geradoras ou destinatárias. A migração de uma tarefa impactará no tempo de resposta de todos os fluxos relacionados a mesma (gerados por ela e destinados a ela), pois modificará a rota que o fluxo deverá percorrer para chegar ao seu destino.

A mudança da rota de um fluxo pode causar alterações no seu tempo de resposta de duas formas, alterando o: (i) número de *hops* pelo qual o fluxo tem que passar até seu destino; (ii) conjunto de fluxos que exercerão interferência direta e indireta no fluxo que teve sua rota modificada. Tais modificações serão regidas pelo padrão de migração das tarefas relacionadas ao fluxo. No caso do escalonamento semi-particionado, o padrão de modificação das rotas de cada fluxo variam de acordo com cada um dos dois modelos de migração previstos por este modelo de escalonamento. Desta forma a próxima seção descreve migração assimétrica e na Seção 4.2 é apresentada a migração simétrica.

4.1 Migração Assimétrica

No padrão de migração assimétrica, assumindo que cada tarefa migratória esteja dividida entre dois processadores, uma tarefa executará sua primeira parcela, migrará para o próximo processador, executará sua parcela final e, então, retornará para o processador original antes de sua próxima liberação. Dessa forma, o processador de início da execução e de término da execução de cada tarefa migratória serão sempre os mesmos.

Considerando-se o padrão de comunicações discutido no Capítulo 3, onde um fluxo de comunicação será enviado sempre após o término de sua tarefa geradora, é seguro assumir que a tarefa geradora não provocará mudanças na rota de seu fluxo de comunicação, pois a mesma sempre terminará sua execução no mesmo processador. Apesar da análise não tratar de garantias de sincronia entre duas tarefas, como é assumido que o fluxo sempre se destinará ao processador de início de execução da tarefa receptora, as rotas dos fluxos permanecerão inalteradas pela migração da tarefa receptora. Devido ao fato de a rota de cada fluxo permanecer inalterada por migrações, tanto da tarefa geradora quanto da tarefa receptora, o número de *hops* que o fluxo necessitará para chegar ao seu destino e o conjunto de fluxos interferentes permanecerão inalterados no curso de execução do sistema. Tendo em vista as particularidades do modelo de migração assimétrica, o escalonamento de fluxos de

comunicação e tarefas em um sistema semi-particionado com migração assimétrica será realizado de acordo com o algoritmo proposto a seguir.

Modelagem:

1. Distribuir tarefas nos processadores e selecionar tarefas migratórias e suas rotas.

Análise:

1. Realizar o escalonamento das tarefas por *response time*. No caso das tarefas migratórias, é necessária apenas a análise de escalonamento da segunda parcela da tarefa, pois, a primeira parcela executará com prioridade maior a todas as outras tarefas do seu respectivo processador.

2. Reajustar o atraso de liberação de fluxos de comunicação gerados por tarefas migratórias de acordo com a equação:

$$J_i^R = r_{j1,1} + r_{j2,1}$$

Onde J_i^R é o atraso de liberação de um fluxo *ci* gerado pela tarefa migratória t.

3. Realizar a análise de escalonamento dos fluxos de comunicação de acordo com o modelo de análise de escalonamento de comunicações definida na Seção 3.4, substituindo os atrasos de liberação dos fluxos gerados por tarefas migratórias de acordo com a equação acima.

Algoritmo 1. Algoritmo para o escalonamento de tarefas e fluxos de comunicação para sistemas semi-particionados com migração assimétrica.

4.2 Migração Simétrica

No padrão de migração simétrico, cada tarefa migrará apenas uma vez por liberação, ou seja, considerando que tarefas serão divididas em duas partes, cada tarefa migrará para o processador seguinte do ciclo de migração após a execução da primeira parcela, terminando sua execução neste processador e permanecendo no mesmo até sua próxima liberação. Como na migração simétrica a tarefa não retorna para o processador de origem após o término de sua execução, os processadores de término e início da execução da tarefa serão alternados a cada liberação da tarefa.

A alternância entre processadores de início e final de execução das tarefas causa a variação da rota de cada fluxo relacionado a tarefas migratórias. Essa alteração se dá devido à alteração no número de *hops* na rota do fluxo e variação dos conjuntos de interferência que afetam o fluxo. Devido ao caráter assíncrono do relacionamento entre a tarefa geradora do fluxo de comunicação e a tarefa receptora, o comportamento migratório de ambas as tarefas afetará a rota do fluxo de comunicação . O relacionamento de um fluxo com apenas uma tarefa migratória produzirá duas rotas possíveis para o fluxo, enquanto o relacionamento com duas tarefas migratórias resultará em quatro variações de rota (Figura 13).

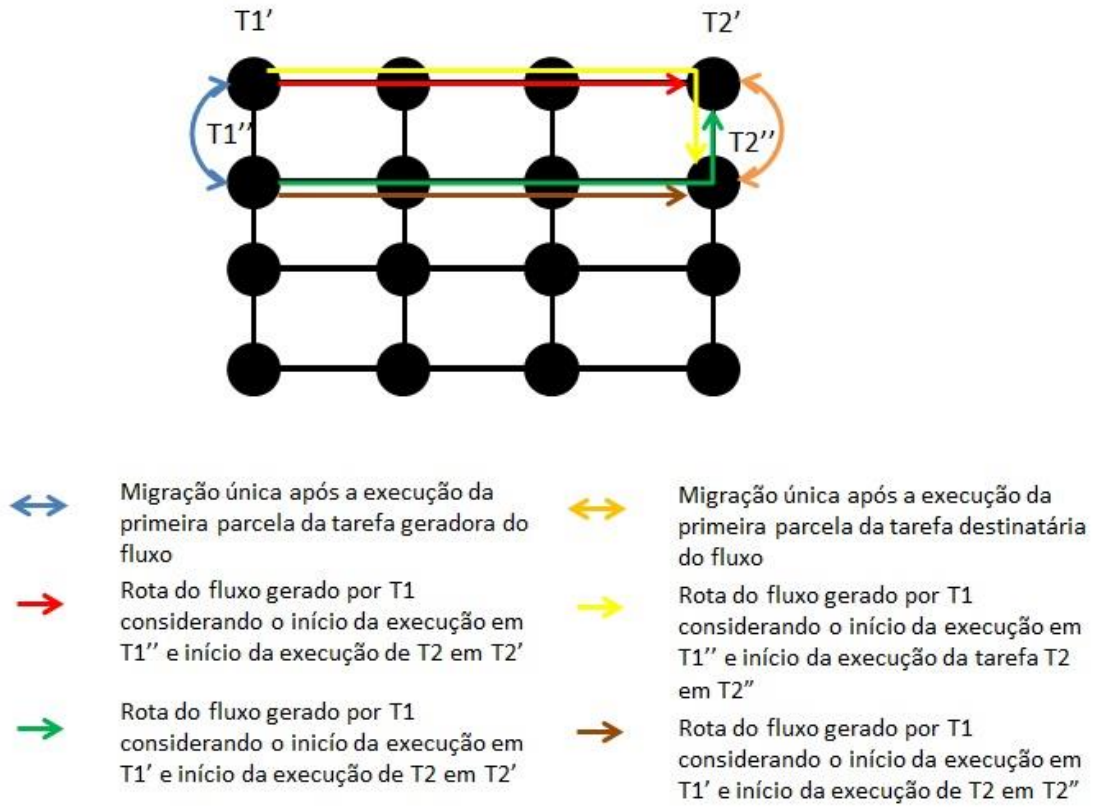

Figura 13: Possibilidades de rota de um fluxo relacionado a duas tarefas migratórias.

As alterações na rota dos fluxos de comunicação devido à migração de tarefas pode interferir no tempo de resposta dos fluxos do sistema de forma direta ou indireta. O impacto direto se dá devido às alterações na rota do próprio fluxo sendo analisado, que causarão possível variação na latência do fluxo (devido à variação do número de *hops*) e a possível variação do conjunto de fluxos que geram interferências diretas e indiretas no fluxo sendo analisado (Figura 14). Fluxos não migratórios podem ser impactados de forma indireta, pois a variação das rotas nos fluxos migratórios podem causar variações no conjunto de fluxos interferentes mesmo para fluxos não migratórios. Para que o impacto da migração de tarefas em fluxos migratórios seja avaliado de forma a prover garantias temporais para o sistema, deve-se levar em conta ambas as formas de impacto.

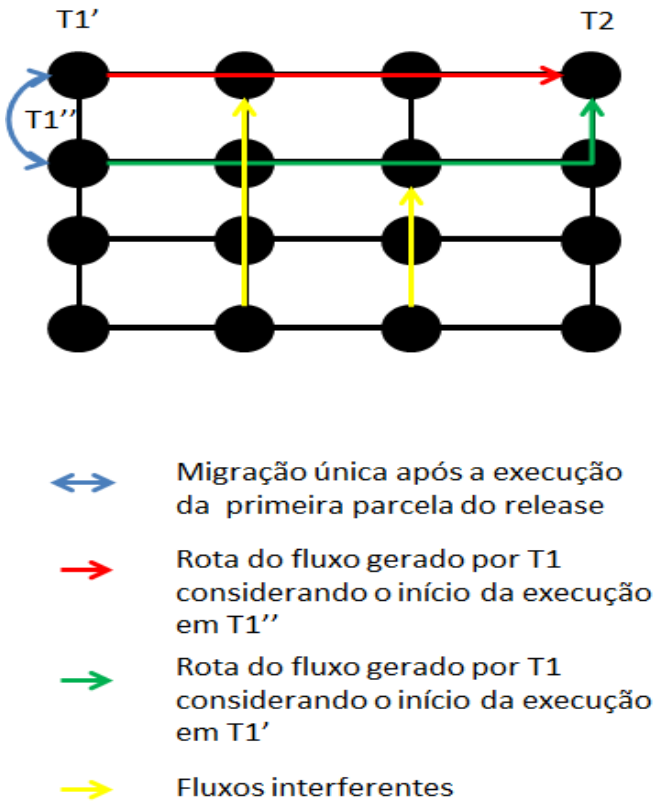

Figura 14: Demonstração da variação nos padrões de interferência gerados pela variação de rotas de um fluxo.

Como mostrado na Figura 14, caso a tarefa T1 inicie sua execução pela sua parcela T1’’ (terminando sua execução em T1’), o fluxo de comunicação gerado (marcado em vermelho) sofrerá interferência de apenas um dos possíveis fluxos interferentes. Quando sua execução inicia pela parcela T1’, o mesmo fluxo (com uma nova rota) sofre interferência de dois fluxos.

A análise de escalonamento de fluxos deve ser capaz de analisar o pior impacto possível da variação de rotas por parte dos fluxos. Para que esse fator seja levado em consideração, serão propostas modificações na análise de escalonamento definida na Seção 3.4 (Equação(7)). Primeiramente, deve-se contrair o cálculo do atraso de interferência indireta da equação final como mostrado a seguir:

$$W(i)^k = C_i + \sum_{Hp(i)} \left\lceil \frac{W(i)^{K-1} + J_j^{R\prime} + J_{ji}^I}{T_j} \right\rceil \times C_j \qquad (10)$$

$$J_{ji}^I = \sum_{Hpj(i)} \left\lceil \frac{R_j - J_j^{R\prime} + J_k^{R\prime} + J_{kj}^I}{T_k} \right\rceil \times C_k \qquad (11)$$

Considerando que cada possível rota de um fluxo apresentará potencialmente um conjunto de interferência distinto e variações na latência do fluxo, rotas distintas poderão

apresentar *response times* e atrasos de interferência direta distintos. Portanto, para que a análise de escalonamento possa fornecer garantias temporais, deve-se calcular cada um dos possíveis *response times e* atrasos de interferência para cada fluxo migratório. Deve ser ressaltado que para que seja calculado o *response time* de um fluxo que seja migratório ou sofra interferência de um, é necessário que cada um dos seus possíveis atrasos de interferência sejam conhecidos, portanto, é imperativo que esta análise de escalonamento seja feita em ordem decrescente de prioridade de fluxos. Dessa forma, os possíveis atrasos de interferência de cada fluxo interferente serão calculados durante as análises posteriores.

As alterações necessárias para o cálculo do *response time* de um fluxo considerando migrações de tarefas serão representadas através de alterações em variáveis na equação original da análise de escalonamento. No cálculo da janela de prioridade para o fluxo *i* (Equação (11)), deverá ser calculado uma janela de prioridade ($W(i)^k$) para cada possível rota do fluxo i. Portanto o termo $W(i)^k$ será substituído por $W(i,n)^k$ onde o *n* indexará as possíveis rotas do fluxo *i*/ Juntamente com esta modificação será feita a substituição do termo J_{ji}^R pelo termo ${J_{ji,n}^I}^{MAX}$. Tal termo representa o atraso de interferência máximo apresentando pelo fluxo *j* na variação de rota *n* do fluxo *i* , considerando todas as variações de rota do fluxo k que interceptam a rota *n* do fluxo *i*. Tais modificações resultarão na Equação (12) que deverá substituir a Equação (9).

$$W(i,n)^k = C_i + \sum_{Hp(i)} \left\lceil \frac{W(i)^{K-1} + J_j^{R\prime} + {J_{ji,n}^I}^{MAX}}{T_j} \right\rceil \times C_j \qquad (12)$$

O cálculo do atraso de interferência (Equação (11)), também deverá ser adaptado, pois deve ser calculado o atraso de interferência de cada uma das variações de rotas do fluxo para que seja definido o atraso de interferência máximo $({J_{ji,n}^I}^{MAX})$. Portanto, o termo J_{ji}^I será substituído por $J_{ji,mn}^I$, onde *m* indexa a variação de rota do fluxo *j* e *n* indexa a variação de rota do fluxo *i*. Para que o atraso de interferência seja calculado para o fluxo j, deve ser considerado que o fluxo sofrerá o máximo atraso possível de fluxos de interferência indireta. Deve ser assumido que os fluxos de interferência indireta *k* deverão apresentar o máximo atraso de interferência possível. Portanto, o termo J_{kj}^I deverá ser substituído por ${J_{kj,n}^I}^{MAX}$, onde *n* indexará a variação de rota do fluxo j que está sendo analisada. Após as modificações discutidas será gerada a Equação (13) que deverá substituir a Equação (11).

$$J_{ji,mn}^I = \sum_{Hpj(i)} \left\lceil \frac{R_j - J_j^{R\prime} + J_k^{R\prime} + {J_{kj,n}^I}^{MAX},}{T_k} \right\rceil \times C_k \qquad (13)$$

Finalmente, no cálculo do *response time* do fluxo, deve-se considerar que ele exibiu o pior comportamento temporal possível. Portanto, deve ser considerada a pior das janelas de prioridade dentre suas possíveis rotas. Considerando-se a equação final do cálculo de *response time* (Equação (8)), deve-se substituir o termo $W(i)$ por $W(i)^{MAX}$, representando a

maior das janelas de priopridade para o fluxo *i*. Tal modificação resultará na equação (14) que deve substituir a Equação (8).

$$R_i = W(i)^{MAX} + r_i \quad (14)$$

Considerando as modificações realizadas na análise de escalonamento de fluxos de comunicação, é proposto a seguir um algoritmo para a análise de escalonamento das tarefas e fluxos de comunicação em um sistema semi-particionado.

Modelagem:

1. Distribuir tarefas nos processadores e selecionar tarefas migratórias e suas rotas de migração.
2. Identificar os fluxos que possuam relacionamento com tarefas migratórias (se originem ou se destinem a uma tarefa migratória). Criar um cenário de fluxo para cada variação de rota (sendo todos os cenários atrelados a um único fluxo).
3. Identificar as variações de conjuntos de interferência direta para cada um dos cenários de cada fluxo migratório.

Análise:

1. Realizar a análise de escalonamento de tarefas através da análise de *response time*, considerando-se as divisões de tarefas migratórias realizadas de acordo com o modelo do método de escalonamento semi-particionado de prioridade fixa selecionado. É feito o calculo das latências dos fluxos de comunicação remodelados para que ela reflita a alteração na rota seguida pelo fluxo causada pela migração de tarefas. Se as novas posições das tarefas possuírem uma maior distância entre si que as posições originais, a fluxo necessitará de mais *hops* para chegar ao seu destino, aumentando sua latência;
2. Reajustar o atraso de liberação dos fluxos migratórios, pois o padrão de interferência sofrido pela parte final da tarefa geradora do fluxo mudará no modelo de comunicação simétrico. O motivo da mudança do padrão de interferência é o fato de que a tarefa alterna seu término entre dois processadores distintos (cada um possuindo o seu conjunto de tarefas). O cálculo do atraso de liberação dos sub-fluxos é dado por:

$$J^R_{i,1} = r_{i1,1} + r_{i2,2}$$

$$J^R_{i,1} = r_{i1,2} + r_{i2,1}$$

Para toda tarefa *i*, *ri1,1* é o *response time* de sua primeira parcela sendo executada no processador inicial do padrão de migração, *ri2,2* é o *response time* de sua segunda parcela da tarefa executando no segundo processador do padrão de migração, *ri1,2* é o *response time* de sua primeira parcela executando no segundo processador do padrão de migração e *ri2,1* é o *response time* de sua segunda parcela sendo executada no processador inicial do padrão de migração.

3. Realizar a análise de escalonamento dos fluxos de comunicação de acordo com o modelo de análise de escalonamento de comunicações discutido previamente nesta seção.

Algoritmo 2. Algoritmo para o escalonamento de tarefas e comunicações em um sistema semi-particionado com migração simétrica.

4.3 Considerações Finais

Deve ser ressaltado que a análise de escalonamento de fluxos migratórios proposta neste capítulo, com o objetivo de prover garantias temporais, considera que cada fluxo sofrerá *hits* através da pior rota possível para cada um de seus fluxos interferentes. Esse cenário trata-se de um cenário pessimista, pois, na maioria dos casos, um fluxo não poderá dar dois *hits* consecutivos em outro fluxo através da mesma rota, tendo em vista que haverá uma migração entre os *hits* e, portanto, uma alteração na rota do segundo *hit*. Um fluxo de comunicação terá sua rota alterada por migrações da tarefa geradora do fluxo e da tarefa receptora. Caso o fluxo possua uma tarefa migratória como tarefa geradora, ele apresentará sempre uma variação de rota entre liberações, pois a tarefa geradora e o fluxo gerado compartilham do mesmo período.

No entanto, a tarefa receptora de um fluxo de comunicação pode não apresentar um período igual ao do fluxo de comunicação a ser recebido. Portanto, quando apenas a tarefa receptora é uma tarefa migratória, um fluxo poderá ser liberado mais de uma vez por uma mesma rota. Porém, isso só poderá ocorrer se o período da tarefa receptora for maior do que o período da tarefa geradora adicionado ao tempo de resposta exibido pela tarefa geradora na segunda liberação de seu fluxo. Caso a tarefa receptora possua um período maior que o período da tarefa geradora adicionado de seu melhor tempo de resposta possível e menor que o período da tarefa geradora adicionado de seu pior tempo de resposta possível, não é possível garantir qual das duas rotas será utilizada pela segunda liberação do fluxo, pois a rota poderá variar a cada liberação da tarefa. É apenas possível a previsão garantida da segunda rota do fluxo caso o período da tarefa receptora seja maior que o período da tarefa geradora adicionado do pior tempo de resposta da mesma. Neste caso, poderá ser determinada a quantidade de liberações (*n*) que o fluxo poderá se manter em uma mesma rota através da Equação (15).

$$n = \left\lceil \frac{T_{geradora} + J^{r}_{fluxo}}{T_{receptora}} \right\rceil \qquad (15)$$

Caso tanto a tarefa geradora do fluxo quanto a receptora forem tarefas migratórias, o padrão de alterações de rota do fluxo torna-se mais complexo. O fluxo sempre alterará de rota em cada liberação, porém, em cada liberação ele possuirá duas rotas possíveis (determinadas pela migração de sua tarefa receptora). Entretanto, o processador da próxima liberação da tarefa receptora pode ser inferido, utilizando-se a noção de proporcionalidade de períodos determinada pela Equação (15). Apesar desta possibilidade de inferência do próximo processador de liberação da tarefa receptora, ela dependerá do momento da primeira liberação das duas tarefas, algo que, atualmente, não pode ser determinado, pois seria necessário analisar qual combinação de tempos de liberação geraria o pior padrão de variação de rotas.

5. Escalonamento semi-particionado de tarefas com análise de comunicações

Este capítulo descreve a definição de um algoritmo de distribuição semi-particionada de tarefas, sendo apresentado nesta seção. Na Seção 5.1 é apresentado um algoritmo para semi-particionamento de um conjunto de tarefas com necessidades de comunicação. A seção 5.2 apresenta uma avaliação do algoritmo proposto na Seção 5.1.

Em (Kato e Yamasaki, 2009), foi proposto um modelo de escalonamento semi-particionado baseado no modelo de escalonamento particionado DM (*deadline monotonic*) chamado DM-PM. Juntamente com o modelo de escalonamento, foi proposto um algoritmo para alocação de tarefas, seleção, divisão e realocação de tarefas migratórias baseado em conceitos da análise de *response time*.

Entretanto, este algoritmo não prevê a presença de comunicações no sistema e utiliza-se de um padrão fixo de distribuição de tarefas. O comportamento temporal de fluxos de comunicação está fortemente atrelado à posição de suas tarefas geradoras e receptoras, pois as mesmas definem seu conjunto de interferência e latência. Portanto a distribuição de um conjunto específico de tarefas sempre no mesmo posicionamento dificulta o escalonamento de fluxos de comunicação.

Outro fator desconsiderado pelo algoritmo de distribuição de tarefas migratórias proposto por Kato e Yamasaki é a presença de atrasos de liberação nas parcelas resultantes da divisão de tarefas migratórias. Para que seja garantido que ambas as parcelas de uma tarefa migratória sejam escalonáveis, deve-se considerar que as mesmas não serão executadas simultaneamente. A condição da execução não simultânea é forçada através da introdução de um atraso de liberação na segunda parcela da tarefa a ser executada igual ao tempo de execução da primeira parcela da tarefa.

Para que a condição de execução não simultânea seja introduzida no algoritmo de divisão de tarefas proposto por Kato e Yamasaki, o cálculo de parcelas proposto no algoritmo deve levar em consideração a possível ocorrência de *back to back hits* causados pelo atraso de liberação da segunda parcela da tarefa. Também deve ser considerado que o atraso de liberação se comportará de forma distinta para os dois tipos de migração (simétrica e assimétrica). No caso da migração assimétrica, o atraso de liberação sempre ocorrerá na mesma parcela, pois a tarefa sempre terminará sua execução no mesmo processador. A migração simétrica fará com que o atraso de liberação reveze entre as duas parcelas de uma tarefa, pois a tarefa revezará o seu término entre os processadores.

Cada tarefa migratória t_i do sistema será dividida em duas parcelas (t'_i e t''_i), possuindo, respectivamente, parcelas $C'_i \; e \; C''_i$ da computação total da tarefa t_i. Assumindo a premissa de que as tarefas migratórias sempre executarão com prioridade máxima, pode-se considerar que o pior tempo de execução da primeira parcela de uma tarefa será sempre fixo e

igual a parcela de computação C'_i. Entretanto, para a análise de possíveis *back to back hits*, considerar toda a parcela C'_i como atraso de liberação para a segunda parcela da tarefa aumentará o pessimismo da análise de escalonamento. O atraso efetivo é dado pela variação máxima do tempo de execução de uma tarefa, considerando seu pior e melhor tempo de execução, pois a diferença entre os dois limita o desvio máximo entre tempos de execução da parcela.

É possível considerar-se que o melhor tempo de execução da primeira parcela da tarefa como sendo a mesma proporção do melhor tempo de resposta total que sua parcela de computação C'_i é do tempo total de computação C_i. Entretanto, em um cenário real partes trechos específicos do código de uma tarefa podem contribuir com maior intensidade para o pior tempo de computação possível (trechos críticos) de uma tarefa, tal relação de proporcionalidade pode não proceder para certos casos, pois não pode ser previsto qual das parcelas ficara com o trecho crítico.

Além de o cálculo de divisão de parcelas proposto em (Kato e Yamasaki, 2009) não levar em consideração o atraso no inicio da execução de uma parcela gerado pela execução da outra parcela da tarefa, o cálculo também não leva em consideração a interferência total sofrida por cada tarefa presente no processador. O algoritmo de divisão de tarefas do DM-PM calcula o quanto cada tarefa do processador suportaria de interferência (oriunda de uma parcela de uma tarefa migratória) sem perder prazo através da Equação (16):

$$C'_j = \frac{D_i - R_i}{\left\lceil \frac{T_i + J_j^R}{T_j} \right\rceil} \qquad (16)$$

A Equação (16) considera o *response time* de uma tarefa como o máximo de computação executada durante o prazo da tarefa, subtraindo este valor do próprio prazo da tarefa para medir a disponibilidade de tempo para execução de outra tarefa de maior prioridade em relação à tarefa interferida (t_i). Porém, como está sendo considerada a introdução de uma nova fonte de interferência, o *response time* pode não fornecer uma medida precisa da interferência máxima sofrida pela tarefa sendo analisada. Isso ocorre, pois uma tarefa pode terminar sua execução antes de sofrer a totalidade da interferência exercida durante seu período. Essa possibilidade é ilustrada pela Figura 15.

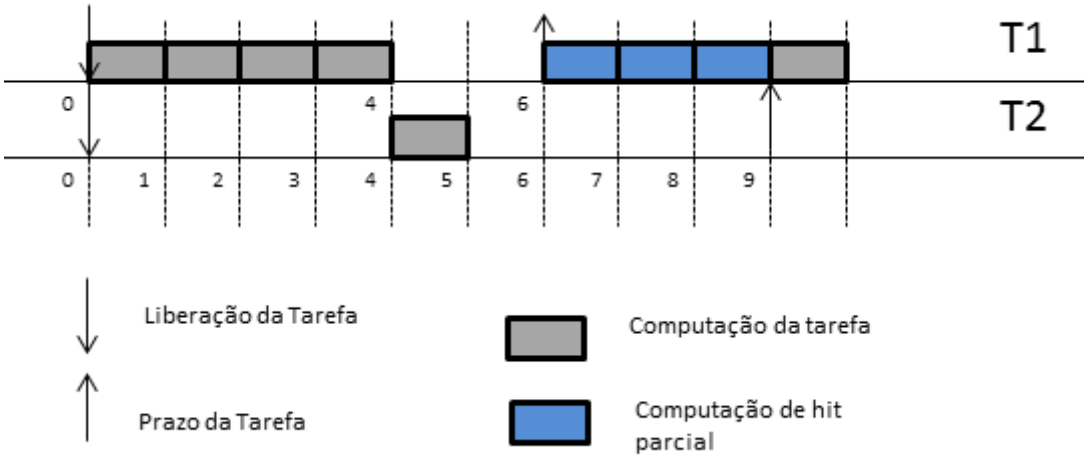

Figura 15:
Demonstração da vaga temporal real para a parcela de uma tarefa migratória em relação à tarefa t_2.

Na Figura 15, a tarefa $t_2(9,0,1)$ termina sua execução 1 unidade de tempo antes do segundo *hit* da tarefa $t_1(6,0,4)$ que ocupa todo o restante do período da tarefa t_2. Neste caso, $D_2 - R_2$ resultaria em 4 unidades de tempo, não representando corretamente o tempo disponível para execução de outras tarefas em relação à tarefa t_2. No exemplo representado pela Figura 15 qualquer nova fonte de interferência a gerar uma interferência maior do que 1 unidade de tempo, resultará na perda de prazo por parte da tarefa t_2. Portanto, para determinar-se o tempo disponível para execução de novas tarefas em relação a uma tarefa qualquer, deve ser considerada a interferência total sofrida pela mesma durante todo seu período e não apenas até o término de sua execução. Deve ser considerado apenas o tempo de execução da tarefa interferente que ocorra durante o período de *release* da tarefa interferida. No caso do exemplo anterior, a tarefa t_1 exerce 7 unidades de tempo de interferência na tarefa t_2 ao invés de 8 (que seria o total de computação de dois *hits* da tarefa t_1).

É importante ser ressaltado que os intervalos de interferência gerados devido a *hits* parciais podem ser interrompidos por *hits* parciais oriundos de tarefas de mais alta prioridade, podendo causar a redução da interferência exercida pelo *hit* parcial. Tal comportamento é demonstrado pela Figura 16.

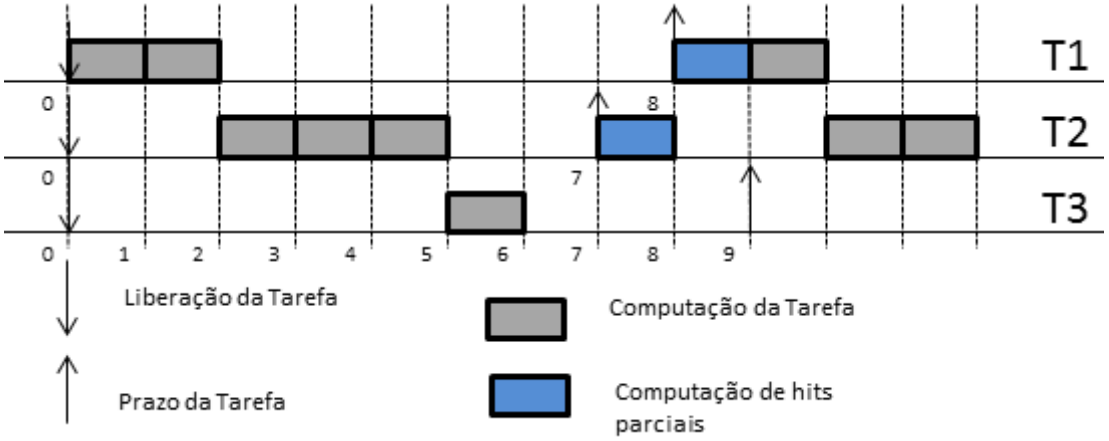

Figura 16: Ilustração da interação entre hits parciais das tarefas t_2 e t_1.

Como a Figura 16 demonstra, a interferência da tarefa $t_2(7,0,3)$ é interrompida pela segunda liberação da tarefa $t_1(8,0,2)$. Portanto, para determinar-se a interferência oriunda dos *hits* parciais deve ser realizada uma análise da interação entre os *hits* parciais de cada tarefa (que se manifestarão na forma de intersecções entre intervalos de interferência). Considerando esses fatores, é proposto o seguinte algoritmo (Algoritmo 3) para análise de *hits* parciais em relação a tarefas presentes em cada processador.

O algoritmo proposto percorre as tarefas interferentes em ordem decrescente de prioridade, verificando a ocorrência de *hits* parciais. Caso um *hit* parcial ocorra (limite superior do número de *hits* é maior que o limite inferior), o algoritmo insere o intervalo que o *hit* ocupará em uma lista, considerando os intervalos inseridos anteriormente. Caso existam intervalos na lista que apresentem intersecção com o intervalo a ser inserido, mais de o intervalo será dividido em segmentos de forma a ocupar a máxima computação disponível dentro de suas necessidades computacionais.

```
1   Interferência_total = 0
2   Intervalos = []
3   Para cada tarefa tj que interfere em ti
4   {
5       Interferência_total = Interferência_total + ⌊(Ti + Jj^R) / Tj⌋ * Cj
6       Se (⌊(Ti + Jj^R) / Tj⌋ > ⌈(Ti + Jj^R) / Tj⌉)
7       {
8           Inicio = ⌊(Ti + Jj^R) / Tj⌋ * Tj
9           Se (Intervalos == [])
10          {
11              Se (Inicio + Cj > Ti) Intervalos.adiciona(Inicio,Ti)
12              Senão Intervalos.adiciona(Inicio,Inicio + Cj)
13          }
14          Senão
15          {
16              Para todo intervalo em Intervalos
17              {
18                  (x,y) ← Intervalos[n]
19                  Se (x ≤ Inicio e y > Inicio)
20                      Inicio = x
21              }
22              resto = Cj
23              Para todo intervalo em Intervalos
24              {
25                  (x,y) ← Intervalos[n]
26                  Se(y == Inicio)
27                  {
28                      (z,k) ← Intervalos[n+1]
29                      Se (Inicio + resto ≤ z)
30                          Intervalos.adiciona(Inicio,Inicio+resto)
31                      Senão
32                      {
33                          resto = resto − (z − Inicio)
34                          Intervalos.adiciona(Inicio,Inicio+z)
35                          Inicio = k
36                      }
37                  }
38                  Senão
39                  {
40                      Se (Inicio + resto ≤ x)
41                          Intervalos.adiciona(Inicio,Inicio+resto)
42                      Senão
43                      {
44                          resto = resto − (x − Inicio)
45                          Intervalos.adiciona(Inicio,Inicio+x)
46                          Inicio = y
47                      }
48
49                  }
50              }
51              Se (Inicio < Ti e resto > 0)
52              {
53                  Se (Inicio + Cj > Ti) Intervalos.adiciona(Inicio,Ti)
54                  Senão Intervalos.adiciona(Inicio,Inicio + Cj)
55              }
56          }
57      }
58  }
59  Para cada intervalo em Intervalos
60  {
61      (x,y) ← Intervalos[n]
62      Interferência_total = Interferência_total + y − x
63  }
```

Algoritmo 3. Análise de interação entre hits parciais.

Após a realização da análise de *hits* parciais, a interferência total sofrida pela tarefa t_i será a soma da interferência total dos *hits* integrais (já contida na variável $Interferência_{total}$) com o somatório da computação contida nos intervalos de *hits* parciais. O tempo vago total no processador em relação à tarefa t_i poderá ser calculado através da subtração do prazo da tarefa t_i pela sua interferência total proveniente de análise. Tal resultado pode ser substituído pelo termo $D_i - R_i$ no cálculo de parcelas da análise proposta por Kato e Yamasaki. Porém, o resultado da análise de Kato e Yamasaki após a substituição pode causar um sub-aproveitamento do tempo livre disponível no processador. Esse sub-aproveitamento pode ocorrer quando o último *hit* da tarefa migratória possui um prazo que se estende além do período da tarefa interferida. Neste caso, a utilização do intervalo de tempo livre total no período da tarefa considerará que o *hit* da tarefa migratória não poderá se estender além do período da tarefa interferida, desconsiderando qualquer computação adicional que poderia ser utilizada pela tarefa migratória.

Para o cálculo da computação disponível para a parcela migratória, deve-se considerar que uma tarefa não pode sofrer uma interferência maior que a dada pelo seu momento crítico, o qual é utilizado para realizar-se a análise de interferência total. Portanto, mesmo que o total de computação de uma tarefa interferente não tenha sido atendido durante o período da tarefa interferida, o fato de sua computação restante ser exercida como interferência na próxima liberação da mesma não poderá gerar uma interferência maior que a interferência oriunda do momento crítico. Sendo assim, se o total de computação de um *hit* parcial da parcela da tarefa migratória (a ser inserida no processador) não exercer toda sua interferência até o final do período da tarefa interferida, não poderá ocorrer uma interferência maior que a estimada através dos seus *hits* integrais. Se o *hit* parcial de uma tarefa migratória dispor de um intervalo de tempo livre menor que o intervalo que seria ocupado por cada um dos hits integrais, é seguro assumir que o hit parcial pode usufruir de uma vaga temporal idêntica aos *hits* anteriores. Para se estimar a capacidade de computação disponível para os *hits* integrais da parcela de uma tarefa migratória, deve ser considerado o tempo livre presente no intervalo de tempo entre o inicio do *hit* parcial da parcela da tarefa migratória e o final do período da tarefa interferida.

A seguir, é proposto um algoritmo para o cálculo da interferência gerada pelos *hits* parciais das tarefas analisadas pelo Algoritmo 3 e estimativa da máxima computação possível para cada *hit* da tarefa migratória.

```
1   t_k←tarefa_migratória
2   J'^R_k = 0
3   hits = ⌈(T_i + J'^R_k) / T_k⌉
4   hit_parcial = ((hits − 1) * T_k) − J'^R_k
5   ordena(Intervalos)
6   Para cada intervalo em Intervalos
7   {
8           (x,y)←Intervalos[n]
9           Se (x < hit_parcial e y < hit_parcial)
10          {
11                  interferência_total = interferência_total + y − x
12                  Se (n é o ultimo intervalo da lista)
13                  {
14                          C'_k = (T_i − (interf + C_i)) / T_k
15                  }
16          }
17          Senão
18          {
19                  Se (interferência_total + C_i < hit_parcial)
20                  {
21                          Se (x< hit_parcial)
22                          {
23                              interferência_total= interferência_total+( hit_parcial − x)
24                              interferência_parcial = interferência_parcial+( y − hit_parcial)
25                          }
26                          Senão interferência_parcial = y − x
27                          Para todo intervalo n+1...n+m
28                          {
29                                  (z,k)← Intervalos
30                                  interferência_parcial = interferência_parcial + k − z
31                          }
32                          hit_integral = (T_i + interferência_total) / (hits − 1)
33                          Se (hit_integral > (T_i − hit_parcial) − interferência_parcial)
34                          {
35                                  C'_k = hit_integral
36                                  termina_loop
37                          }
38                          Senão
39                          {
40                                   interf = interferência_total + interferência_parcial
41                                  C'_k = (T_i − (interf + C_i)) / T_k
42                                  termina_loop
43                          }
44                  }
45                  Senão
46                  {
47                          interferência_parcial = y − x
48                          Para todo intervalo n+1...n+m
49                          {
50                                  (z,k)← Intervalos
51                                  interferência_parcial = interferência_parcial + k − z
52                          }
                              nterf = interferência_total + interferência_parcial
53                                  C'_k = (T_i − (interf + C_i)) / T_k
54                                  termina_loop
55                  }
56          }
57  }
```

Algoritmo 4. Algoritmo de estimativa da capacidade computacional disponível para cada hit de uma parcela da tarefa migratória.

Com a introdução do Algoritmo 4 no final do Algoritmo 3, o algoritmo resultante fornecerá uma estimativa do total de computação possível de ser atribuído para a parcela da

tarefa migratória. Porém, tal estimativa não leva em consideração o atraso de liberação imposto pela primeira parcela a ser gerada na segunda parcela da tarefa sendo calculada. Esse atraso de liberação pode impactar esta estimativa através do fenômeno de *back to back hit* ou através do deslocamento do inicio do *hit* parcial da parcela migratória para um novo ponto na linha de tempo como demonstrado pelas Figuras 15 e 16.

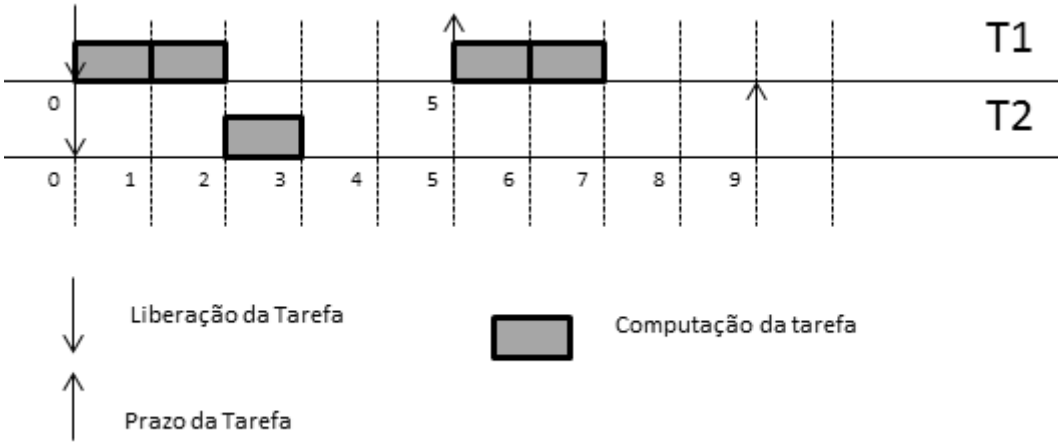

Figura 17: Comportamento temporal da tarefa t_1 sem atraso de liberação.

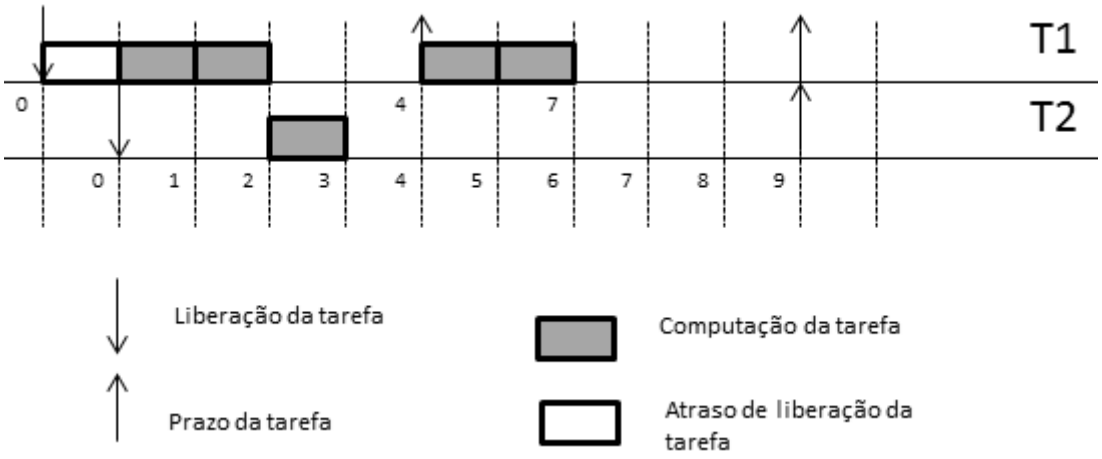

Figura 18: Comportamento temporal da tarefa t_1 após a introdução de atraso de liberação.

A Figura 17 demonstra a interpretação gráfica da interferência da tarefa $t_1(2,5)$ na tarefa $t_2(1,9)$. Como pode ser observado, a tarefa t_1 termina o seu *hit* integral e inicia seu ultimo *hit* em 5. Após a inserção de 1 unidade de tempo como atraso de liberação para t_1, apesar da não ocorrência de um *back to back hit*, o ponto de início do ultimo *hit* de t_1 é deslocado para 4 (*hit* anterior é deslocado $-J_1^R$ unidades de tempo), como ilustrado pela Figura 14. Finalmente, com a inserção de 2 unidades de tempo como atraso de liberação para

t_1, um *back to back hit* ocorre fazendo com que t_1 inflija um terceiro *hit* em 8 (um *hit é* adicionado e o ultimo *hit* é deslocado $T_1 - J_1^R$ unidades de tempo), durante o período de t_2 (Figura 15).

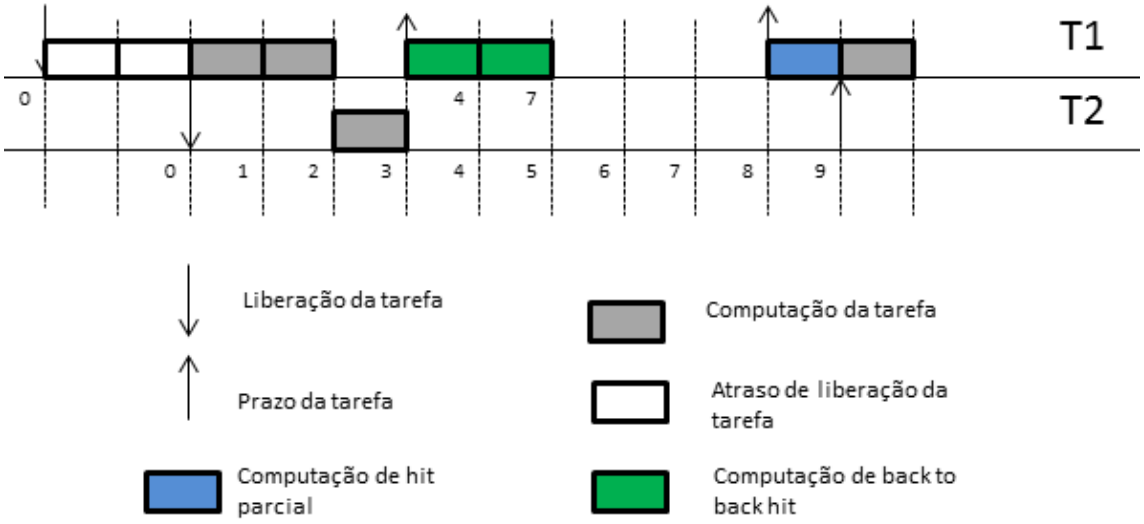

Figura 19: Comportamento temporal da tarefa t_1 durante a ocorrência de um back to back hit.

Tendo em vista as condições demonstradas pelas figuras anteriores, uma nova estimava tem que ser feita utilizando-se o atraso de liberação que seria gerado pela primeira estimativa ($J_k'^R = C_k - C_k' = C_k''$). O novo valor de atraso de liberação deve ser substituído no Algoritmo 2 e uma nova estimativa deve ser calculada, sendo esta a estimativa definitiva. A nova estimativa não se trata de um valor exato da capacidade de computação disponível, pois o novo valor de C_k' gerará um novo valor para $J_k'^R$, porém, com o objetivo de simplificar o cálculo, não será calculada uma nova estimativa com o atraso de liberação reajustado.

O algoritmo proposto por Kato e Yamasaki leva em consideração que a tarefa migratória possui um período menor que todas as tarefas presentes no processador. Entretanto, este trabalho trata com a possibilidade do mapeamento heurístico das tarefas considerando a comunicação, o que torna esta premissa restritiva. Para que o algoritmo de mapeamento não seja restringido por esta premissa, o algoritmo de estimativa da computação das parcelas proposto anteriormente deve ser modificado para levar em consideração a possibilidade de existirem tarefas com um período menor que a tarefa migratória no processador sendo analisado.

Quando é calculada a computação possível para uma parcela de uma tarefa migratória em relação a tarefas com um período maior que o da tarefa migratória, é levado em consideração que haverá vários *hits* da tarefa migratória e a computação disponível no

intervalo é dividida de acordo. Porém, quando se trata de uma tarefa com um período menor que o da tarefa migratória, não se pode considerar a disponibilidade de tempo em várias liberações da tarefa interferida. Como a tarefa migratória será executada com a maior prioridade possível no processador, ela exercerá toda sua interferência no processador ininterruptamente. Portanto, se a interferência exercida pela tarefa migratória for maior que o intervalo de tempo livre na primeira liberação da tarefa interferida, ocorrerá a perda de prazo da tarefa interferida. Para que esta condição seja levada em consideração, deve-se considerar o total de tempo livre em apenas uma liberação da tarefa interferida para o cálculo da computação disponível para a parcela da tarefa migratória. Portanto, se a tarefa analisada possuir um período menor que o da tarefa migratória, ao final do Algoritmo 1, devem ser somados a interferência total dos *hits* integrais, o somatório da computação contida nos *hits* parciais e a computação da tarefa interferida. A computação disponível para a parcela da tarefa migratória será igual à subtração do prazo da tarefa interferida pelo resultado da soma previamente mencionada. Novamente deve ser analisado o atraso gerado proveniente da segunda parcela, se este atraso gerar um *back to back hit* por parte da tarefa migratória, deve-se recalcular a computação disponível para cada parcela da tarefa migratória através do Algoritmo 4, caso contrário, a computação previamente calculada poderá ser mantida.

O algoritmo proposto neste capítulo deve ser realizado para cada par de processadores vizinhos e para cada tarefa migratória. Caso uma tarefa migratória não possa ser alocada em nenhum par de processadores, o algoritmo falha e uma nova alocação de tarefas ou um a nova seleção de tarefas migratórias deve ser realizada.

5.1 Semi-particionamento de tarefas com necessidades de comunicação

O algoritmo de divisão de tarefas proposto no início do capítulo trata-se estritamente de um algoritmo de divisão e mapeamento de tarefas, não levando em conta a presença de comunicação. A necessidade de comunicação em tarefas migratórias representa um novo desafio, pois suas possibilidades de posicionamento são limitadas pelo fato de que são necessários dois processadores vizinhos com capacidade suficiente para suportar a computação integral da tarefa. Outro fator a ser levado em consideração na seleção de uma tarefa migratória é o tamanho de seu período. Idealmente, a tarefa migratória terá um período menor que todas outras tarefas no processador devido a limitações impostas pela condição contrária que foram discutidas no início do capítulo.

Se estes fatores forem considerados em conjunto com o fato que tarefas migratórias modificarão os padrões de interferência entre comunicações no sistema, a escolha de uma tarefa migratória ideal torna-se um processo complexo. Este processo pode ser realizado de uma forma mais simples através de uma abordagem de tentativa e erro, realizando a divisão e

alocação da tarefa sem levar em consideração a comunicação e checando o escalonamento dos fluxos de comunicação (através da análise proposta no Capítulo 4) no estado final do sistema. Porém, a falha na divisão de tarefas leva a uma série de opções para a tentativa da correção do problema: (i) tentar realocar a tarefa migratória; (ii) selecionar uma nova tarefa como tarefa migratória; (iii) tentar realizar um reposicionamento das tarefas que possuem fluxos problemáticos; (iv) realocar todo o conjunto novamente.

A tentativa de realocar a tarefa é a alternativa menos custosa das alternativas mencionadas, porém, as possibilidades de alocação para tarefas migratórias são limitadas pela necessidade de vizinhança entre os processadores. O processo de seleção de uma nova tarefa migratória exige a alocação da antiga tarefa migratória e a seleção de uma nova tarefa migratória o que pode ser um processo complexo e pouco recompensador. Realizar a realocação das tarefas com fluxos problemáticos também pode ser um processo complexo pois pode envolver o deslocamento de outras tarefas que por sua vez poderão causar o deslocamento de fluxos e novos problemas com fluxos de comunicação. Assumindo que a alocação do conjunto é feita através de um processo heurístico como em (Bonilha, 2011), a tentativa de realocação de todas as tarefas pode gerar mapeamentos diferentes e, portanto, novas possibilidades de tarefas migratórias.

Com o objetivo da realização de testes preliminares para o possível desenvolvimento de heurísticas de seleção e alocação de tarefas migratórias focadas em comunicação, abaixo é proposto um algoritmo para o mapeamento de tarefas e divisão de tarefas migratórias levando em conta comunicação. Este algoritmo é composto por uma série de passos que são enumerados no Algoritmo 5.

1) Realizar a alocação do conjunto com um algoritmo heurístico de alocação de tarefas focado em computação e comunicação como o proposto em (Bonilha, 2011).
2) Realizar a análise de escalonamento das tarefas e verificar se algum processador possui tarefas que perderam prazo.
3) Caso alguma tarefa tenha perdido prazo, para cada processador, selecionar e marcar como migratória a tarefa com o menor período possível no processador que forneça uma vaga temporal suficiente para tarefa que perdeu prazo não perder prazo. Caso tal tarefa não exista, marcar a tarefa que perdeu prazo como tarefa migratória.
4) Realizar a divisão e alocação das tarefas migratórias de acordo com o algoritmo proposto no início deste capítulo. Se o algoritmo falhar retornar ao passo 1.
5) Realizar a análise de escalonamento proposta no Capítulo 4 para verificar a escalonabilidade dos fluxos de comunicação.
6) Caso algum fluxo de comunicação não atenda seu prazo, retornar ao passo 1.

Algoritmo 5. Algoritmo de alocação de tarefas para sistemas semi-particionados com comunicação.

5.2 Avaliação do algoritmo

Com o objetivo de avaliar a capacidade de alocação do Algoritmo 5, foi realizada uma série de testes. Tais testes consistiram na execução do algoritmo para o mapeamento de conjuntos

de tarefas e fluxos gerados aleatoriamente considerando uma NoC 4x3. Foram gerados conjuntos aleatórios de tarefas com 75%, 80%, 85% e 90% de utilização de processador, com o objetivo de exercer diferentes graus de dificuldade ao algoritmo. Para cada conjunto de tarefas, foram criados fluxos de comunicação partindo de 75% das tarefas. Para cada fluxo foi atribuída uma latência entre 5% e 10% do período não utilizado pela computação de sua tarefa geradora. Dessa forma, prevenindo que um par de tarefa geradora e fluxo gerado não fosse não-escalonável sem sofrer interferência de outros fluxos.

Os testes foram realizados para duas abordagens distintas: (i) execução do algoritmo de divisão de tarefas sem remapeamento caso o algoritmo falhe em dividir as tarefas; (ii) execução do algoritmo de divisão de tarefas com 10 tentativas de remapeamento em caso de falha. Para cada uma dessas abordagens e para cada um dos marcos de utilização de processador previamente mencionados, foram gerados 1000 conjuntos de tarefas aleatórios.

Para cada um dos testes, foram realizadas medições de: (i) número de sucessos da heurística de mapeamento (sem divisão de tarefas); (ii) número de sucessos do algoritmo de divisão (recuperação de conjuntos não escalonáveis produzidos pela heurística de mapeamento); (iii) número de ocorrências de aumento no número de fluxos não escalonáveis após a divisão de tarefas; (iv) número de redução no número de erros de fluxos após a divisão de tarefas.

5.2.1 Divisão de tarefas sem remapeamento de tarefas

Para os testes do algoritmo de divisão de tarefas, o algoritmo de divisão de tarefas foi executado apenas uma vez (sem remapeamento em caso de falha) para cada um dos 1000 conjuntos de tarefas e fluxos aleatórios com 75%, 80%, 85% e 90% de utilização de processador. Para cada execução do algoritmo, foram medidas a taxa de sucesso da heurística particionada (sem necessidade de divisão de tarefas), a taxa de recuperação de um conjunto não escalonável por parte do algoritmo de divisão de tarefas e a taxa total de sucesso do algoritmo (conjuntos escalonáveis produzidos por qualquer meio). Os resultados destes testes são demonstrados nos Gráficos 1, 2 e 3.

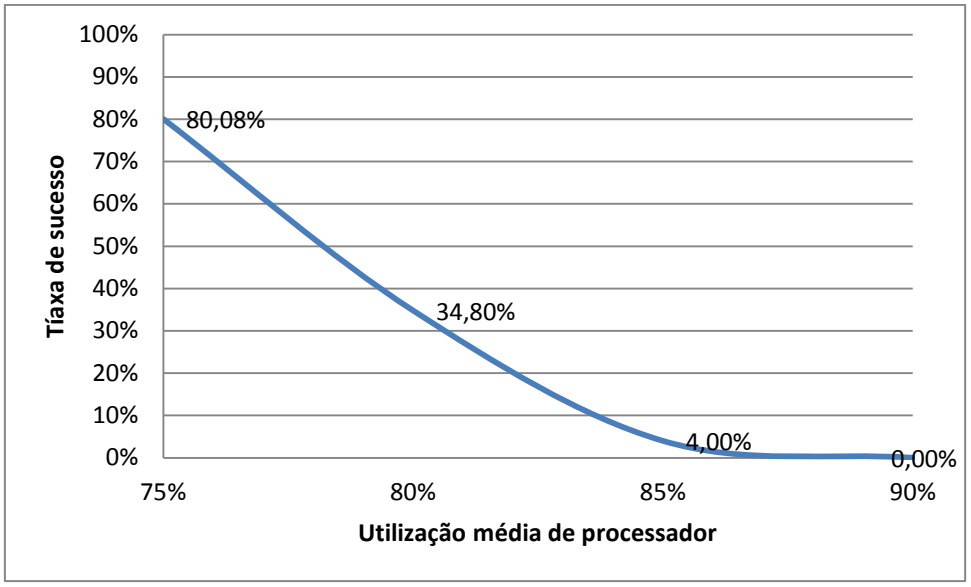

Gráfico 1. Taxa de sucesso do mapeamento heurístico de tarefas (sem divisão).

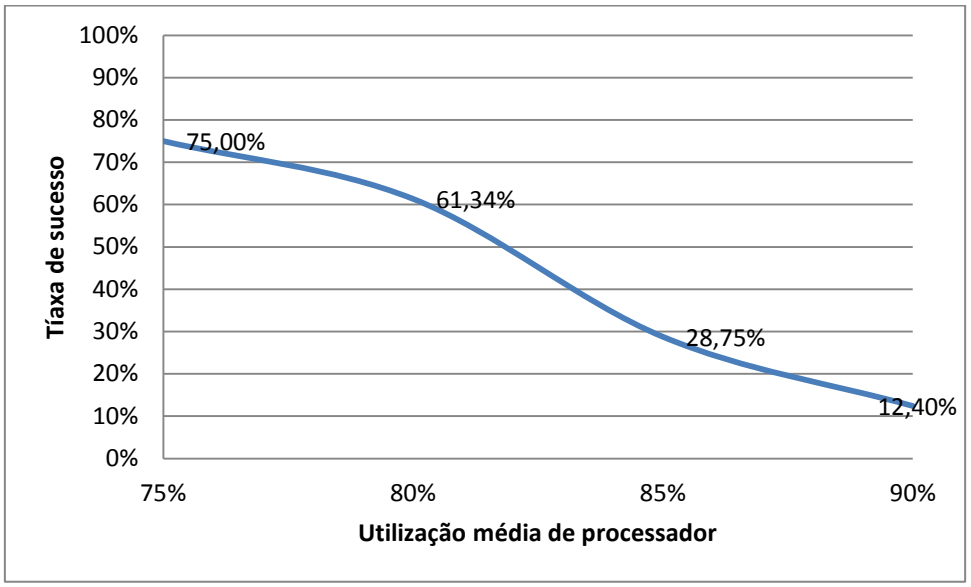

Gráfico 2. Taxa de sucesso na recuperação de mapeamentos não escalonáveis por parte do algoritmo de divisão de tarefas.

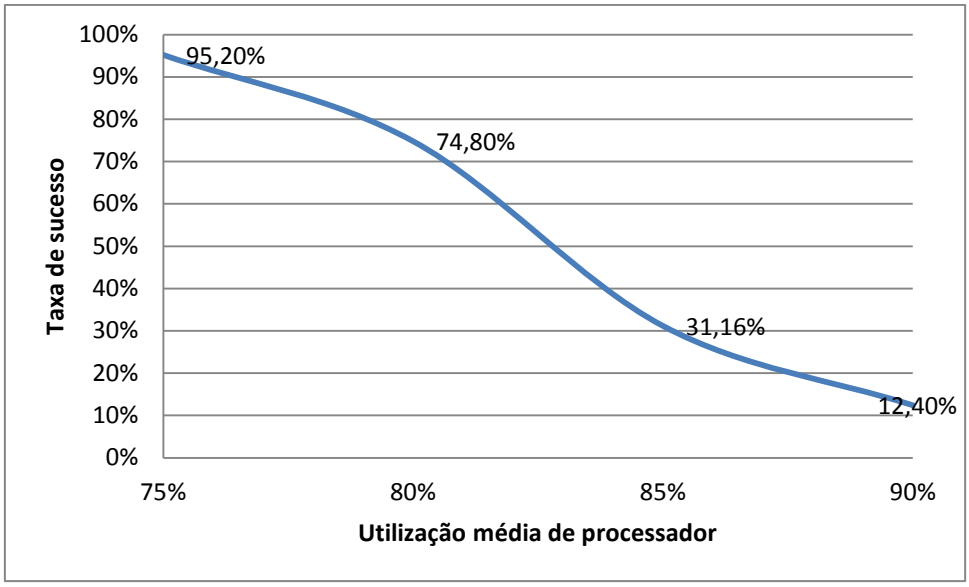

Gráfico 3. Taxa de sucesso total do algoritmo.

Como pode ser observado no Gráfico 1, o mapeamento heurístico de tarefa apresentou uma taxa de sucesso de 80,08% no mapeamento de conjuntos de tarefa com 75% de utilização de processado, uma taxa de sucesso alta se considerado que algoritmos semi-particionados apresentaram garantia de escalonamento para conjunto com no máximo 69% de utilização, como discutido na Seção 2.1.1. Entretanto, a taxa de sucesso do mapeamento heurístico decresceu rapidamente com o aumento da utilização de processador, apresentando uma taxa de sucesso de 34,80% para conjuntos com 80% de utilização de processador, 4% para conjuntos com 85% e 0% para conjuntos com 90%.

O algoritmo de divisão de tarefas apresentou uma taxa de sucesso na recuperação de mapeamentos não escalonáveis de 75% para conjuntos com 75% de utilização, uma taxa de sucesso mais baixa que a do algoritmo de mapeamento heurístico para mesma categoria de utilização de processador. Porém, com o aumento da dificuldade de escalonamento dos conjuntos, o algoritmo de divisão de tarefas apresentou resultados melhores que o mapeamento heurístico. O Algoritmo de divisão de tarefas apresentou uma taxa de sucesso de 61,34% na recuperação de conjuntos com 80% de utilização de processador, 28,75% na recuperação de conjuntos com 85% e 12,40% na recuperação de conjuntos com 90% de utilização. A criação de tarefas migratórias conseguiu uma taxa de sucesso consideravelmente maior na criação de mapeamentos escalonáveis para conjuntos de tarefas com grande utilização de processador, mostrando a necessidade da divisão de tarefas para o escalonamento de tais conjuntos.

A taxa de sucesso total do algoritmo (considerando obtenção de mapeamentos escalonáveis com e sem migração) demostra claramente que o mapeamento heurístico e a divisão de tarefas se complementam. Obtendo uma taxa de sucesso de 95,20% no mapeamento de tarefas com 75% de utilização de processador, 74,80% no mapeamento de tarefas com 80%, 31,16% no mapeamento de tarefas com 85% e 12,40% no mapeamento de tarefas com 90%.

No entanto, deve ser ressaltado que o algoritmo de divisão de tarefas, apesar de partir de um mapeamento favorável para comunicação das tarefas, não leva em consideração tal fator durante a divisão e remapeamento de tarefas. Devido a esse fator, o algoritmo apresentou aumento do número de perdas de prazo por parte de fluxos após a divisão de alguns conjuntos, como demonstrado pelo Gráfico 4.

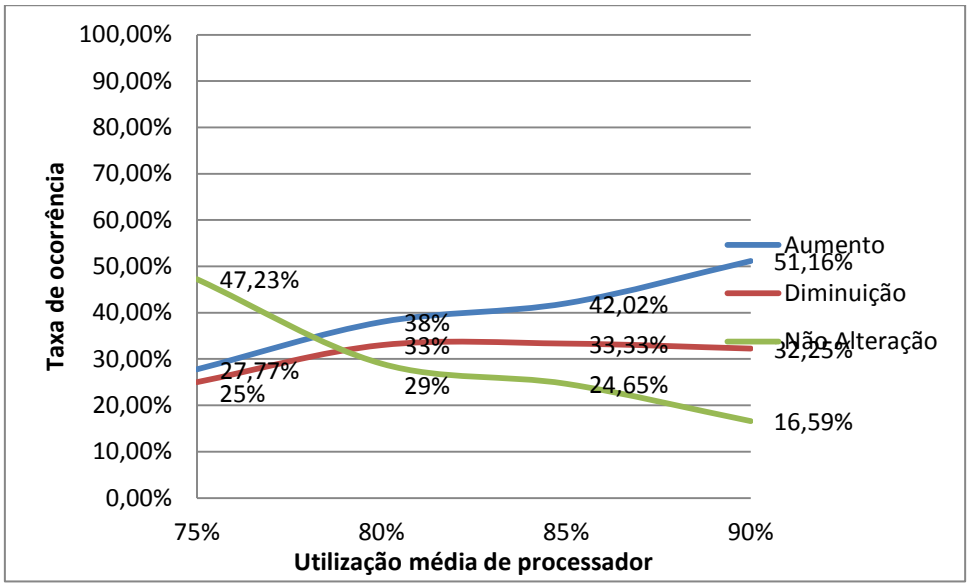

Gráfico 4. Taxa ocorrência de alterações no número de prazos perdidos por parte de fluxos.

O algoritmo de divisão de tarefas apresentou uma taxa crescente de ocorrência de aumento no número de perdas de prazo por parte dos fluxos com o aumento da utilização de processador dos conjuntos de tarefa. A ocorrência de aumento nos erros de tarefas alcançou uma taxa de ocorrência acima de 50% em conjuntos com 90% de utilização. Também foi constatado um índice de ocorrência de redução no número de perdas de prazos de fluxos relativamente alto (se mantendo próximo aos 33% para conjuntos com 80%, 85% e 90% de utilização de processador). Tais resultados demonstram a necessidade da introdução de heurísticas de divisão de tarefas focadas na comunicação de tarefas. A presença de uma taxa mínima de não aumento no número de perdas de prazos de 48,84% sem utilização de heurísticas de divisão também demonstra que tais heurísticas possuem grande potencial de sucesso.

5.2.2 Divisão de tarefas com remapeamento de tarefas

Foram realizados testes com o algoritmo de divisão de tarefas com remapeamentos heurísticos em caso de falha, como sugerido na Seção 5.1, sendo imposto um limite de 10 tentativas de remapeamento em caso de falha. O algoritmo foi testado em 1000 conjuntos de tarefas aleatórios para 75%, 80%, 85% e 90% de utilização, como os testes anteriores. O Gráfico 5 ilustra os resultados dos testes.

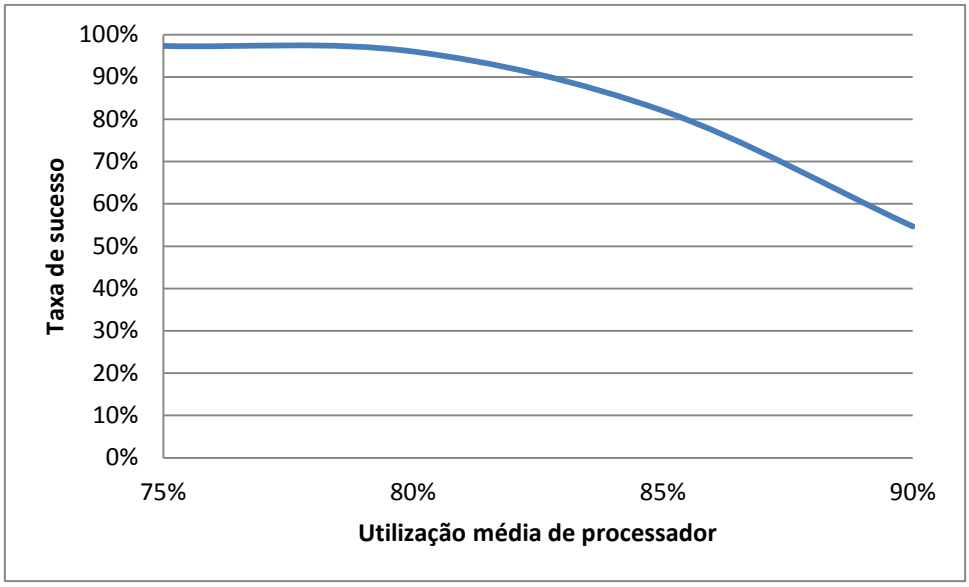

Gráfico 5. Taxa de sucesso do algoritmo de divisão de tarefas com 10 remapeamentos em caso de falha.

A introdução de remapeamentos em caso de falha provocou um aumento significativo na taxa de sucesso do algoritmo de divisão de tarefas, indicando que a combinação de períodos de tarefas em cada um dos processadores é um fator de grande impacto na eficácia do algoritmo de divisão de tarefas. O algoritmo de divisão de tarefas com remapeamentos em caso de falha apresentou uma taxa de sucesso de 97,31% para conjuntos com 75% de utilização, 96% para conjuntos com 80%, 82% para conjuntos com 85% e 54,66% para conjuntos com 90% de utilização.

A grande melhora na taxa de sucesso do algoritmo, provocada pelo remapeamento, sugere a existência de diversas possibilidades de mapeamentos de tarefas escalonáveis para um conjunto. Esta possibilidade pode ser explorada através do remapeamento arbitrário (independente de falha) para geração de vários mapeamentos escalonáveis, podendo ser escolhido o mapeamento mais benéfico para os fluxos de comunicação do sistema.

6. Considerações Finais

Atualmente, no desenvolvimento industrial de SoCs (System on Chip) multiprocessados de tempo real, utiliza-se o modelo de escalonamento semi-particionado. Tal modelo de escalonamento garante a escalonabilidade de conjuntos de tarefas com no máximo 69% de utilização de processador, aumentando o custo de tais sistemas devido ao baixo aproveitamento de recursos. Apesar da presença de modelos de escalonamento mais eficientes na literatura atual, tais modelos não são adotados devido à falta de estudos e ferramentas para obtenção de garantias temporais para tais modelos em condições reais de execução. Este trabalho teve como objetivo o desenvolvimento de uma análise de escalonamento capaz de prover garantias temporais para um conjunto de fluxos e tarefas considerando fatores como o impacto da migração de uma tarefa nos seus fluxos de comunicação, algo que é abstraído nas análises de escalonamento atuais, tornando a análise mais representativa das condições reais de um sistema.

O trabalho foi iniciado com uma análise do pessimismo presente na análise de escalonamento para fluxos de comunicação proposta por (Shi e Burns, 2010) onde fontes de pessimismo desnecessário foram identificadas. Através da identificação de fontes de pessimismo desnecessário na análise, foram feitas modificações na análise de escalonamento de Shi e Burns, gerando uma nova análise de escalonamento menos pessimista mas ainda capaz de fornecer garantias temporais para os fluxos de comunicação. Foi proposta uma nova modificação na análise resultante para que a mesma leve em consideração os atrasos impostos pela tarefa geradora no seu fluxo de comunicação.

Utilizando-se da nova análise de escalonamento proposta na Seção 3.3, foi proposta uma análise de escalonamento para sistemas com escalonamento semi-particionado. Tal análise foi criada com o intuito de considerar-se o impacto da migração de tarefas no comportamento temporal dos fluxos de comunicação do sistema. Foi levantada uma discussão a respeito de como cada um dos modelos de migração interfere no comportamento temporal dos fluxos de comunicação e foi proposta uma análise de escalonamento para cada um destes modelos de migração.

Levando em consideração o fato de que os algoritmos de divisão de tarefas migratórias encontrados na literatura atual, não só consideram comunicação entre tarefas, mas utilizam-se de mapeamentos iniciais, na maioria dos casos, determinísticos. Foi proposto um novo algoritmo de divisão e remapeamento de tarefas migratórias que é capaz de reduzir as restrições impostas no mapeamento prévio por algoritmos como o de (Kato e Yamasaki, 2009). Também foi demonstrado que o algoritmo proposto por Kato e Yamasaki pode produzir resultados equivocados devido a não consideração da interferência exercida após o término da execução de uma tarefa.

Finalmente, foram realizados testes do novo algoritmo de divisão de tarefas para identificar-se a possibilidade da junção deste algoritmo com heurísticas da mapeamento de tarefas dirigidas a comunicação entre tarefas. Os resultados dos testes sugeriram uma boa

sinergia entre o algoritmo de mapeamento de tarefas focado em comunicação e o algoritmo de divisão de tarefas para o mapeamento de conjuntos de tarefa com um alto nível de utilização de processador. Os testes também indicaram a necessidade de heurísticas adicionais capazes de dirigir a divisão e remapeamento de tarefas migratórias de forma a beneficiar o comportamento temporal dos fluxos de comunicação do sistema.

6.1 Trabalhos Futuros

Os testes realizados com o algoritmo de divisão de tarefas demonstram a necessidade de heurísticas capazes de guiar o algoritmo de forma a beneficiar o comportamento temporal dos fluxos de comunicação. Além disso, estes testes sugeriram que tais heurísticas possuam um certo grau de liberdade para interagir com o algoritmo de remapeamento de tarefas migratórias, tendo em vista o alto índice de não aumento do número de perdas de prazos por parte de fluxos obtidos pelo algoritmo sem a utilização das mesmas.

O algoritmo de criação e divisão de tarefas migratórias possui 2 pontos de tomada de decisão cruciais: (i) qual tarefas serão as tarefas migratórias; (ii) onde essas tarefas migratórias devem ser mapeadas. Introdução de heurísticas no ponto de decisão (ii) poderiam tornar não só o remapeamento de tarefas migratórias mais benéfico para os fluxos de comunicação do sistema como também torná-lo mais eficiente. Entretanto, a introdução de heurísticas na tomada de decisão (i) poderia prover melhores candidatas à divisão e à relocação, tarefas com fluxos de comunicação menos impactantes ou com períodos mais propícios para o algoritmo de divisão. São necessários estudos a respeito de possíveis heurísticas e o impacto das mesmas para ambos pontos de tomada de decisão.

Outra melhoria possível de ser feita é a introdução de novos fatores no modelo de análise de escalonamento para sistemas semi-particionados, como atrasos de transferência de estado entre processadores para tarefas migratórias e a modificação de rotas de fluxos interferentes entre *hits*. Porém, para a introdução do segundo, seria necessário um estudo a respeito da definição da pior sequência de migrações possíveis para a interferência de um fluxo.

Finalmente, podem ser realizados estudos sobre a utilização de novos esquemas de prioridade para o modelo de escalonamento semi-particionado, como o EDF (*Earliest Deadline First*). Algoritmos de divisão de tarefas baseados em EDF provaram-se mais eficientes na divisão de tarefas migratórias, porém, tal esquema de prioridade torna mais difícil a obtenção de garantias como o momento de geração do fluxo de comunicação no sistema (imprevisibilidade).

7. Bibliografia

C. L. Liu e J. W. Layland. Scheduling Algorithms for Multiprogramming in Hard-Real-Time Enviroment, Journal of the ACM, vol. 10, issue 1, 1973.

R. I. Davis e A. Burns. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems, ACM Computing Surveys, vol. 43, no. 4, 35-35, 2011.

Z. Shi e A. Burns. Schedulability Analysis and task mapping for real-time on-chip communication, Real-Time System, vol. 46, issue 3, 360-385, 2010.

N. C. Audsley, A. Burns, R. I. Davis, K. Tindell e A. J. Wellings. Fixed Priority Pre-emptive Scheduling: An Historical Perspective, Real-Time Systems, vol. 8, issue 2-3, 173-198, 1995.

J. D. Ullman. NP-Complete Scheduling Problems, Journal of Computer and System Sciences, vol. 10, issue 3, 384-393, 1975.

A. Burchard, J. Liebeherr, Y. Oh e S.H. Son. New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems, IEEE Transactions on Computers, vol. 44, no. 12, 1429-1442, 1995.

I. Bonilha, Alocação de Tarefas para Aplicações de Tempo Real em Arquiteturas Multi-core. UFSM, 2011.

L. M. Ni e P. K. McKinley. A survey of wormhole routing techniques in direct networks, IEE Computer, vol. 26, issue 2, 62-76, 1993.

J. M. Lopez, J. L. Diaz e D. F. Garcia. Minimum and maximum utilization bounds for multiprocessor RM scheduling. EuroMicro Conference on Real-Time Systems, pages 67–75, Delft, The Netherlands, June 13–15 2001. 12, 13, 27, 41, 55

J. M. Lopez, J. L. Diaz e D. F. Garcia. Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems. Real-Time Systems, vol. 28, issue 1, 39-68, 2004.

J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. H. Anderson e S. K. Bariah. A categorization of real-time multiprocessor scheduling problems and algorithms, Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Chapman and Hall, 2004.

X. Qi, D. Zhu e H. Aydin. Global scheduling based reliability-aware power management for multiprocessor real-time systems. Real-Time Systems, vol. 47, issue 2, 109-142, 2011.

T.P. Baker. Multiprocessor edf and deadline monotonic schedulability analysis. 24th IEEE Real-Time Systems Symposium, pages 120–129, Dec. 2003.

B. Andersson, S. Baruah, J. Jonsson. Static-Priority Scheduling on Multiprocessors. 22nd IEEE Real-Time Systems Symposium, 193-202, 2001.

J. Goossens , S. Funk, S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. Real-Time Systems, Srpringer, vol 25, no 2-3, 187–205, 2003.

J. Goossens, P. Yomsi. Exact Schedulability Test for global-EDF Scheduling of Periodic Hard Real-Time Tasks on Identical Multiprocessors. Cornell University Library, arXiv:1012.5929, 2010.

A. Srinivasan e J. Anderson. Efficient scheduling of soft real-time applications on multiprocessors. 15th Euromicro Conference on Real-Time Systems, pages 51–59, July 2003.

S. Baruah, N. Cohen, C. Plaxton, D. Varvel. Proportionate Progress: A notion of fairness in resource allocation. Algorithmica, Springer, vol. 15, no. 6, 600-625, 1996.

D. Zhu, D. Mossé e R. Melhem. Multiple-resource periodic scheduling problem: how much fairness is necessary?. 24th IEEE Real-Time Systems Symposium, 142-151, 2003.

B. Andersson, E. Tovar. Multiprocessor Scheduling with Few Preemptions. 12th IEEE Conference on Embedded and Real-Time Computing Systems and Applications, 322-334, 2006.

G. Levin, S. Funk, C. Sadowski, I. Pye, e S. Brandt. Dp-fair: A simple model for understanding optimal multiprocessor scheduling. Euromicro Conference on Real-Time Systems, pages 3–13, 2010.

S. Kato, N. Yamasaki. Semi-Partitioned Fixed-Priority Scheduling on Multiprocessors. 15th IEEE Real-Time and Embedded Technology and Applications Symposium, 23-32, 2009.

S. Kato, N. Yamasaki, Y. Ishikawa. Semi-partitioned Scheduling of Sporadic Task Systems on Multiprocessors. 21st Euromicro Conference on Real-Time Systems, IEEE, 249-258, 2009.

F. Dorin, P. Yomsi, J. Goossens, P. Richard. Semi-Partitioned Hard Real-Time Scheduling with Restricted Migrations upon Identical Multiprocessor Platforms. Cornell University Library, arXiv:1006.2637, 2010.

X. Qi, D. Zhu e H. Aydin. A Study of Utilization Bound and Run-Time Overhead for Cluster Scheduling in Multiprocessor Real-Time Systems. 16th International Conference on Embedded and Real-Time Computing Systems and Applications , 3-12, 2010.

N. Audsley, A. Burns, M. Richardson, K. Tindell e A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal, vol. 8, issue 5, 284-292, 1993.

O. Redell e M. Sanfridson. Exact Best-Case Response Time Analysis of Fixed Priority Scheduled Tasks. 14th Euromicro Conference on Real-Time Systems, 165-172, 2002.