Escalonamento de Tarefas com Relações Arbitrárias de Precedência em Sistemas Tempo Real Distribuídos

Romulo Silva Oliveira

I I - Univ. Fed. do Rio Grande do Sul
Caixa Postal 15064
Porto Alegre-RS, 91501-970, Brasil
romulo@inf.ufrgs.br

Joni da Silva Fraga

LCMI/DAS - Univ. Fed. de Santa Catarina
Caixa Postal 476
Florianopolis-SC, 88040-900, Brasil
fraga@lcmi.ufsc.br

Resumo

Este artigo considera a análise de escalonabilidade de aplicações tempo real distribuídas onde tarefas podem apesentar relações arbitrárias de precedência. É suposto que as tarefas são periódicas ou esporádicas. Elas possuem prioridade fixa e deadline hard fim-a-fim sempre igual ou menor que o respectivo período. É desenvolvido um método para transformar relações arbitrárias de precedência em jitter de liberação. Após eliminar todas as relações de precedência presentes no conjunto de tarefas é possível aplicar qualquer teste de escalonabilidade disponível para tarefas independentes.

Abstract

This paper considers the schedulability analysis of real-time distributed applications where tasks may present arbitrary precedence relations. It is assumed that tasks are periodic or sporadic and dynamically released. They have fixed priorities and hard end-to-end deadlines that are equal to or less than the respective period. We develop a method to transform arbitrary precedence relations into release jitter. By eliminating all precedence relations in the task set one can apply any available schedulability test that is valid for independent task sets.

1 Introdução

Deadlines são críticos em sistemas de tempo real *hard*. Neste tipo de sistema é necessário ter uma garantia em projeto de que todas as tarefas serão sempre concluídas antes do respectivo deadline. Garantia em projeto pode ser obtida com escalonamento baseado em prioridades fixas [11]. Tarefas recebem prioridades de acordo com alguma política e um teste de escalonabilidade é usado *off-line* para garantir que todas as tarefas cumprirão seus deadlines em um cenário de pior caso. Durante a execução o escalonador seleciona a próxima tarefa baseado em suas prioridades fixas.

O escalonamento tempo real em sistemas distribuídos é usualmente dividido em duas fases: alocação e escalonamento local. Inicialmente cada tarefa é alocada a um processador específico. A alocação original é permanente pois normalmente não é possível migrar tarefas durante a execução. A segunda fase analisa a escalonabilidade de cada processador.

Relações de precedência são comuns em sistemas distribuídos. Elas são criadas pela necessidade de sincronização e/ou transferência de dados entre tarefas. É possível usar *offsets* [2] para implementar relações de precedência. Definindo um *offset* entre as liberações de duas tarefas é possível garantir que a tarefa sucessora iniciará sua execução somente após a predecessora estar concluida. Esta técnica é as vezes chamada "liberação estática de tarefas" ("*static release of tasks*" [14]).

Também é possível implementar relações de precedência fazendo a tarefa predecessora enviar uma mensagem para a tarefa sucessora. Esta mensagem informa ao escalonador que a tarefa predecessora terminou e a tarefa sucessora pode ser liberada. Esta técnica é as vezes chamada "liberação dinâmica de tarefas" ("*dynamic release of tasks*" [14]). A incerteza sobre o momento

da liberação da tarefa sucessora pode ser modelado como um *jitter* na liberação [18] (a palavra inglesa *jitter*, como empregada na literatura de tempo real, poderia ser traduzida como "tremor" ou "variação").

A maioria dos estudos sobre relações de precedência tratam apenas com precedência linear ("*pipelines*"), onde cada tarefa tem no máximo um único predecessor e um único sucessor. Modelos de tarefas que admitem relações arbitrárias de precedência não colocam tal restrição. Estudos que consideram relações arbitrárias de precedência normalmente assumem tarefas liberadas estaticamente. Poucos artigos consideram este problema em ambiente distribuído.

A escalonabilidade de tarefas liberadas estaticamente que apresentam relações arbitrárias de precedência e executam em um único processador é analisada em [3] e [8]. Em [5] relações arbitrárias de precedência são consideradas em ambiente distribuído, também para o caso de tarefas liberadas estaticamente. Relações de precedência lineares são analisadas em [7], [9], [12], [13], [14], [15] e [16].

O trabalho descrito em [2] implementa relações arbitrárias de precedência através da liberação dinâmica de tarefas. Aquele artigo mostra que *jitter* de liberação pode ser usado para modelar as relações de precedência criadas pela comunicação entre tarefas executando em diferentes processadores. Entretanto, aquele artigo não desenvolve esta abordagem de forma detalhada.

O trabalho apresentado em [6] desenvolve uma análise de escalonabilidade apropriada para sistemas distribuídos construidos sobre uma rede de comunicação ponto a ponto. Este modelo de tarefas inclui prioridade fixa e tarefas liberadas dinamicamente. Relações de precedência são transformadas em *jitter* de liberação, para que as tarefas possam ser analisadas como se fossem independentes. O modelo de tarefas suporta relações arbitrárias de precedência.

Similarmente, [18] apresenta uma análise de escalonabilidade para aplicações distribuídas baseadas em redes tipo barramento. São supostas prioridades fixas e liberação dinâmica de tarefas. Relações de precedência lineares são transformadas em *jitter* de liberação, permitindo que a análise considere as tarefas como independentes. Relações arbitrárias de precedência são possíveis através da liberação estática de tarefas.

Este artigo considera o escalonamento de tarefas tempo real *hard* com restrições de precedência arbitrárias em ambiente distribuído. É suposto um modelo de tarefas que inclui prrioridades fixas e liberação dinâmica de tarefas. São desenvolvidas regras de transformação que, a partir do conjunto de tarefas original, criam conjuntos equivalentes de tarefas independentes com *jitter* na liberação. Estes novos conjuntos de tarefas independentes podem ser usados para avaliar a escalonabilidade do conjunto original de tarefas.

O método apresentado neste artigo para transformar relações arbitrárias de precedência em *jitter* de liberação é menos pessimista que uma simples transformação direta, como apresentado em [6]. A eliminação de todas as relações de precedência existentes no conjunto original de tarefas resulta na criação de um conjunto independente de tarefas. Este novo conjunto de tarefas pode ser analisado por qualquer teste de escalonabilidade disponível que seja válido para conjuntos de tarefas independentes. Apesar do método introduzido neste artigo ser aplicável à sistemas com apenas precedência linear, o método foi desenvolvido para o caso mais geral de relações arbitrárias de precedência. O objetivo deste trabalho não é prover uma solução melhor para o caso particular das relações lineares de precedência, mas sim prover um método para o caso geral das relações arbitrárias de precedência.

O restante deste artigo está organizado da seguinte forma: a seção 2 descreve a abordagem adotada neste trabalho e formaliza o problema; na seção 3 é desenvolvido um conjunto de regras de transformação; a seção 4 descreve o algoritmo do teste de escalonabilidade; os resultados das simulações são apresentados na seção 5; finalmente, os comentários finais aparecem na seção 6.

2 Descrição da Abordagem

Existe na literatura diversos testes para analisar a escalonabilidade de conjuntos de tarefas independentes quando prioridades fixas são usadas. Alguns destes testes admitem tarefas com

jitter na liberação, tais como [2] e [17]. Estes dois trabalhos analisam a escalonabilidade de cada tarefa computando seu tempo máximo de resposta, isto é, o tempo entre a chegada da tarefa e sua conclusão, no pior caso. Este valor é então comparado com o respectivo deadline.

É possível fazer uma transformação simples e direta das relações de precedência em *jitter* de liberação [6]. Quando uma tarefa T_i possui várias tarefas predecessoras é suficiente identificar a tarefa predecessora capaz de causar o maior atraso no início de T_i. Este atraso inclui o tempo máximo de resposta da tarefa predecessora mais qualquer atraso na comunicação devido ao envio de mensagens entre diferentes processadores. Este atraso máximo para o início de T_i é transformado no *jitter* de liberação da tarefa T_i. Suas relações de precedência podem então ser ignoradas durante a análise de escalonabilidade [6]. Esta análise é feita através da transformação de um sistema com relações de precedência em um sistema equivalente composto de tarefas com *jitter* na liberação mas sem relações de precedência explícitas. Testes de escalonabilidade são aplicados a este sistema equivalente.

Na verdade, a transfomação descrita acima é pessimista no sentido que o tempo máximo de resposta da tarefa pode ser aumentado no processo. A figura 1 mostra uma aplicação simples que ilustra este pessimismo para o caso de um único pocessador. Tarefas T_0, T_1 e T_2 são caracterizadas pelos seus períodos P_0, P_1 e P_2 e seus respectivos tempos máximos de execução C_0, C_1 e C_2. A tarefa T_0 tem a prioridade mais alta e T_2 tem a prioridade mais baixa.

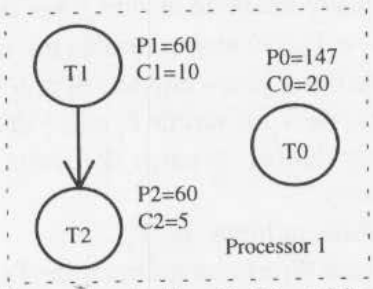

Figura 1 - Aplicação hipotética.

O tempo máximo de resposta da tarefa T_1 é 30. Assim, um *jitter* de liberação igual a 30 é associado com a tarefa T_2 e sua relação de precedência eliminada. No sistema transformado a tarefa T_2 apresenta um tempo máximo de resposta dado pela soma do seu *jitter* de liberação 30 e do seu tempo máximo de execução 5 com a interferência máxima que ela recebe de outras tarefas, a qual é 20. O resultado 55 é muito pessimista pois no cálculo foi considerado que a tarefa T_0 interfere duas vezes com a tarefa T_2. A interferência da tarefa T_0 foi incluida na computação do tempo máximo de resposta de T_1, o qual tornou-se o *jitter* de liberação de T_2. A interferência da tarefa T_0 foi novamente considerada quando o tempo máximo de resposta de T_2 foi calculado. O período 147 da tarefa T_0 torna impossível uma segunda aparição de T_0 dentro de uma única ativação do par de tarefas (T_1,T_2). Nesta aplicação simples é possível percebe que, considerando todas as relações de precedência, o tempo máximo de resposta de T_2 é 35.

O método apresentado neste artigo cria um conjunto de tarefas equivalente para cada tarefa T_i pertencente ao conjunto original. O conjunto equivalente de tarefas é tal que: o tempo máximo de resposta da tarefa T_i no seu conjunto equivalente é maior ou igual ao tempo máximo de resposta desta mesma tarefa no conjunto de tarefas original.

Este trabalho segue a abordagem geral descrita em [6]. É suposto prioridades fixas, liberação dinâmica de tarefas e relações arbitrárias de precedência. Como em [6], todos os deadlines são menores ou iguais ao respectivo período e *jitter* de liberação é usado para modelar parcialmente os efeitos de uma relação de precedência. Diferentemente de [6], este trabalho considera o fato de uma relação de precedência limitar os possíveis padrões de interferência entre tarefas.

2.1 Modelo de Tarefas

Neste artigo é suposto que uma aplicação tempo real executa em um sistema distribuído composto de um conjunto **H** de **h** processadores. O atraso na comunicação remota é limitado e

possui um valor máximo Δ. Os protocolos de comunicação são executados em um processador auxiliar e não competem com as tarefas da aplicação por tempo de processador. O atraso associado com uma comunicação local é suposto zero. Relações de precedência são implementadas por uma mensagem enviada pela tarefa predecessora para a tarefa sucessora.

Uma aplicação é definida por um conjunto **A** de **m** atividades periódicas ou esporádicas. Cada atividade A_X, $A_X \in A$, gera um conjunto possivelmente infinito de chegadas. Uma atividade periódica A_X é caracterizada pelo seu período P_X, isto é, o intervalo de tempo entre duas chegadas sucessivas. É suposto que a primeira chegada de todas as atividades periódicas ocorre no instante zero. Atividades periódicas podem ter várias tarefas iniciais, isto é, tarefas sem predecessores. Uma atividade esporádica A_X é caracterizada pelo seu intervalo de tempo mínimo P_X entre ativações. Atividades esporádicas possuem uma única tarefa inicial. Com um pequeno abuso de notação A_X será usado para representar o conjunto de todas as tarefas que pertencem à atividade A_X.

Cada aplicação é associada com um conjunto **T** de **n** tarefas, isto é, o conjunto de todas as tarefas que pertencem a alguma atividade do conjunto **A**. Qualquer tarefa pode ser suspensa (*preempted*) a qualquer tempo e retomada mais tarde. Cada tarefa T_i é caracterizada por uma prioridade global $\rho(T_i)$, um tempo máximo de execução C_i e um deadline D_i relativo ao instante de chegada da sua atividade. Para todas as tarefas T_i, $1 \le i \le n$, $T_i \in A_X$, temos $D_i \le P_X$. Se a tarefa T_i é uma tarefa inicial de sua atividade ela pode ter um *jitter* de liberação máximo J_i maior que zero. É suposto que se as tarefas T_i e T_k são ambas tarefas iniciais da atividade A_X, então $J_i = J_k$.

Cada tarefa T_i é também caracterizada pelo conjunto $dPred(T_i)$. Este conjunto inclui todas as tarefas que são predecessoras diretas de T_i. A tarefa T_i não é liberada até receber uma mensagem de cada uma das suas predecessoras diretas. A partir do conjunto $dPred(T_i)$ de cada tarefa T_i é possível definir cinco outros conjuntos:

- Conjunto $iPred(T_i)$ das predecessoras indiretas de T_i;
- Conjunto $Pred(T_i)$ das predecessoras diretas ou indiretas de T_i;
- Conjunto $dSuc(T_i)$ das sucessoras diretas de T_i;
- Conjunto $iSuc(T_i)$ das sucessoras indiretas de T_i;
- Conjunto $Suc(T_i)$ das sucessoras diretas ou indiretas de T_i.

Todas as tarefas T_i foram previamente alocadas aos processadores do conjunto H. Algoritmos de alocação estão além do escopo deste artigo. É possível que, como resultado do algoritmo de alocação, tarefas de uma mesma atividade fiquem espalhadas por vários processadores. Uma tarefa não pode migrar durante a execução.

O símbolo R_i^A será usado para denotar o tempo máximo de resposta da tarefa T_i na aplicação **A** e $H(T_i)$ para denotar o processador onde T_i executa. Também, $\rho(A_X)$ representa a mais alta prioridade entre todas as tarefas que pertencem a A_X.

Dada uma aplicação como descrito acima, é necessário determinar se todas as liberações de todas as tarefas estarão sempre concluídas antes do respectivo deadline.

2.2 Atribuição de Prioridades

Quando tarefas com relações de precedência estão distribuídas entre vários processadores, pode existir uma dependência mútua entre o *jitter* de liberação de uma tarefa e o tempo máximo de resposta de outra tarefa. Este problema foi identificado em [18] e resolvido pelo cálculo simultâneo do tempo máximo de resposta de todas as tarefas através de sucessivas iterações.

A abordagem adotada neste trabalho é limitar a forma como prioridades são atribuidas. O problema pode ser eliminado se a política de atribuição de pioridades satisfizer dois requisitos:

- Prioridades locais são definidas a partir de uma ordenação global que inclui todas as tarefas do sistema. Apesar de apenas a ordem das tarefas locais ser importante para o escalonador de cada processador, é suposto que esta ordenação local concorda com a ordenação global.

• A ordenação global é tal que prioridades decrescem ao longo das relações de precedência. Se T_i precede T_j então T_i tem uma prioridade mais alta que T_j. Esta restrição é razoável pois atribui prioridades mais elevadas para as tarefas iniciais das atividades, cujo impacto no tempo de resposta total é maior.

Deadline Monotônico (DM) [1,10] é uma política simples usada para atribuir prioridades que pode satisfazer os dois requisitos definidos antes. DM é uma forma simples e rápida de atribuir prioridades, apesar de não ser ótimo em sistemas com relações de precedência [4]. Entretanto, DM é ótimo na classe das políticas que geram prioridades decrescentes ao longo das relações de precedência [9] e pode ser considerada uma boa heurística de propósito geral. Neste trabalho é assumido que cada tarefa recebe uma prioridade global única de acordo com DM. Deadlines são relativos à chegada da respectiva atividade. A tarefa com o menor deadline relativo recebe a prioridade mais alta. Esta prioridade global única resulta em uma ordenação global das tarefas.

Para satisfazer o requisito de prioridades decrescentes ao longo das relações de precedência é necessário adicionar duas novas regras ao DM:

• Se uma tarefa T_i é predecessora direta da tarefa T_j, então é necessário ter $D_i \leq D_j$, onde D_i e D_j são os respectivos deadlines relativos à chegada da atividade que inclui T_i e T_j. Esta regra não reduz a escalonabilidade do sistema. Não faz sentido a tarefa T_j ter um deadline menor que T_i uma vez que T_j somente pode iniciar sua execução após T_i estar concluída.

• Quando duas tarefas possuem o mesmo deadline, DM não especifica qual delas deve receber a prioridade mais elevada. Neste tabalho é suposto que, quando existe uma relação de precedência entre duas tarefas, a tarefa predecessora recebe a prioridade mais alta.

Combinando DM com estas duas regras fica assegurado que as prioridades das tarefas produzem uma ordenação global e que as prioridades são sempre decrescentes ao longo das relações de precedência. Um efeito colateral desta política de atribuição de prioridades é uma análise de escalonabilidade mais simples. Um efeito similar é mostrado em [9] para tarefas com relações de precedência que são liberadas dinamicamente e executam em um único processador.

Com relação ao modelo de tarefas apresentado na seção 2.1, temos agora que cada tarefa recebe uma prioridade individual única no sistema de acordo com o Deadline Monotônico. É suposto que os deadlines aumentam ao longo das relações de precedência, isto é, se $T_i \in Pred(T_j)$ então $D_i \leq D_j$ e $\rho(T_i) > \rho(T_j)$. Sem perda de generalidade as tarefas serão denominadas na ordem decrescente de suas prioridade. Assim, $i<j$ implica que $\rho(T_i) > \rho(T_j)$.

3 Regras de Transformação

Nesta seção são desenvolvidas regras de transformação capazes de criar um conjunto de tarefas independentes com *jitter* de liberação que é equivalente ao conjunto original mas não inclui relações de precedência. Cada regra é apresentada como um teorema.

Teoremas 1 a 5 descrevem regras para transformar um sistema onde uma tarefa T_i, a única tarefa da atividade A_X, recebe interferência de tarefas das outras atividades. Inicialmente considere a interferência sobre T_i causada por uma atividade completamente local A_y. Uma vez que prioridades decrescem ao longo das relações de precedência existem 3 casos possíveis: todas as tarefas de A_y possuem uma prioridade menor que $\rho(T_i)$; todas as tarefas de A_y possuem uma prioridade maior que $\rho(T_i)$; algumas tarefas de A_y possuem prioridade maior que $\rho(T_i)$ enquanto outras tarefas de A_y possuem uma prioridade menor que $\rho(T_i)$. Estes 3 casos originam, respectivamente, teoremas 1, 2 e 3. A prova destes teoremas é trivial e não será apresentada aqui.

Teorema 1

Seja **A** uma aplicação onde $A_X \in A$, $A_y \in A$, $A_X \neq A_y$ são atividades. A tarefa T_i é a única tarefa da atividade A_X. Suponha também que $\forall T_j \in A_y$, $H(T_j)=H(T_i) \wedge \rho(T_j)<\rho(T_i)$ e que **B** é uma aplicação tal que $B = A - \{A_y\}$.

Nas condições acima temos $R_i^A = R_i^B$. □

Neste ponto é introduzido #dPred(T_i) para denotar o número de elementos no conjunto dPred(T_i) dos predecessores diretos da tarefa T_i. Similarmente, serão usados #dSuc(T_i), #iPred(T_i) e #iSuc(T_i) para respectivamente denotar o número de sucessores diretos, predecessores indiretos e sucessores indiretos da tarefa T_i.

Teorema 2

Seja **A** uma aplicação onde $A_x \in A$, $A_y \in A$, $A_x \neq A_y$ são atividades. A atividade A_x tem uma única tarefa T_i e ainda $\forall T_j \in A_y$, $H(T_j)=H(T_i) \wedge \rho(T_j)>\rho(T_i)$.

Também, existe uma única tarefa T_k tal que $T_k \in A_y \wedge$ #dPred(T_k)=0.

Finalmente, $B = (A-\{A_y\}) \cup \{A_e\}$ é uma aplicação onde A_e é uma atividade com o mesmo período da atividade A_y, $P_e=P_y$, e A_e é composta por uma única tarefa T_e tal que:

$J_e = J_k$, $H(T_e) = H(T_i)$, $\rho(T_e) = \rho(A_y)$, e

$$Ce = \sum_{\forall Tj.Tj \in Ay} Cj .$$

Nas condições acima temos $R_i^A = R_i^B$. □

Teorema 3

Seja **A** uma aplicação onde $A_x \in A$, $A_y \in A$, $A_x \neq A_y$ são atividades. A atividade A_x é composta por uma única tarefa T_i. A atividade A_y é tal que:

$\forall T_j \in A_y . H(T_j) = H(T_i)$,

$\exists T_j \in A_y . \rho(T_j)>\rho(T_i) \wedge \exists T_j \in A_y . \rho(T_j)<\rho(T_i)$.

Também, a atividade A_y tem uma única tarefa inicial T_k, isto é, $T_k \in A_y \wedge$ #dPred(T_k)=0.

Finalmente, suponha $B = (A - \{ A_y \}) \cup \{ A_e \}$ uma aplicação onde A_e é uma atividade composta por uma única tarefa T_e tal que: $P_e >> P_i$, $J_e=J_k$, $H(T_e)=H(T_i)$, $\rho(T_e) = \rho(A_y)$, e

$$Ce = \sum_{\forall Tj.Tj \in Ay \wedge \rho(Tj)>\rho(Ti)} C_j .$$

Nas condições acima temos $R_i^A = R_i^B$. □

Neste ponto é definido $\delta_{i,j}$ como o atraso associado com o envio de uma mensagem da tarefa T_i para a tarefa T_j. Logo, $\delta_{i,j} = \Delta$ quando $H(T_i) \neq H(T_j)$, e $\delta_{i,j} = 0$ quando $H(T_i)=H(T_j)$.

Teorema 4 supõe que a tarefa T_j é executada no mesmo processador que a tarefa T_i e que T_j possui vários predecessores diretos. Entre os predecessores de T_j existe a tarefa T_k, aquela capaz de gerar o maior atraso possível na liberação de T_j. É mostrado que é possível transformar a aplicação de tal forma que T_j passará a ter T_k como único predecessor direto.

Teorema 4

Seja **A** é uma aplicação onde $A_x \in A$, $A_y \in A$, $A_x \neq A_y$ são atividades. A atividade A_x possui uma única tarefa T_i e a atividade A_y é tal que $\exists T_j \in A_y$, $H(T_i)=H(T_j) \wedge$ #dPred(T_j)≥ 2.

Ainda, a tarefa $T_k \in A_y$ é tal que $T_k \in$ dPred(T_j) e

$$R_k^A + \delta_{k,j} = \max_{\forall T_l \in dPred(T_j)} (R_l^A + \delta_{l,j}) .$$

Finalmente, $B = (A - \{A_y\}) \cup \{A_e\}$ é uma aplicação onde A_e é uma atividade composta das mesmas tarefas, relações de precedência e período da atividade A_y, exceto por T_j que possui dPred(T_j)={ T_k }.

Nas condições acima temos $R_i^A \leq R_i^B$.

Prova

Aplicação **B** é o resultado da eliminação de todas as relações de precedência em **A** onde T_j aparece como a tarefa sucessora, exceto pela relação de precedência onde T_k é o predecessor.

Seja Γ_i^A o conjunto de todos as possíveis escalas de execução geradas pela aplicação **A** no processador $H(T_i)$. Observe que R_i^A é definido por uma escala de execução que pertence a Γ_i^A e onde existe a maior distância temporal entre a chegada e a conclusão de T_i.

O *jitter* de liberação máximo da tarefa T_j na aplicação **A** á dado por $R_k^A+\delta_{k,j}$. Como a relação de precedência entre T_k e T_j está também presente na aplicação **B**, o mesmo acontece com o seu *jitter* de liberação máximo em **B**. A única consequência da remoção das demais relações de precedência é o possível aumento do conjunto de todas as possíveis escalas de execução, isto é, $\Gamma_i^A \subseteq \Gamma_i^B$.

Como $\Gamma_i^A \subseteq \Gamma_i^B$, qualquer escala de execução possível em Γ_i^A é também possível em Γ_i^B, inclusive a escala de execução que define R_i^A. Não é possível afirmar qualquer coisa sobre a existência ou não em Γ_i^B de escalas de execução que definem um tempo de resposta maior para T_i. Assim, temos que $R_i^A \leq R_i^B$. □

Teorema 5 supõe uma tarefa T_j que possui um único predecessor direto e executa no mesmo processador que a tarefa T_i executa. Este teorema mostra que a interferência de T_j sobre T_i não é reduzida quando a relação de precedência envolvendo T_j é transformada em *jitter* de liberação.

Teorema 5

Seja **A** uma aplicação onde $A_x \in A$, $A_y \in A$, $A_x \neq A_y$ são atividades. A atividade A_x é composta por uma única tarefa T_i e a atividade A_y é tal que:

$\exists T_j \in A_y, \exists T_k \in A_y , H(T_i)=H(T_j) \wedge dPred(T_j)=\{T_k\}$.

Finalmente, $B = (A - \{ A_y \}) \cup \{ A_e \}$ é uma aplicação onde A_e é uma atividade composta das mesmas tarefas, relações de precedência e período da atividade A_y, exceto pela tarefa T_j que tem $dPred(T_j)=\{ \}$ e $J_j = R_k^A + \delta_{k,j}$.

Nas condições acima temos $R_i^A \leq R_i^B$.

Prova

Aplicação **B** é o resultado da eliminação da única relação de precedência em **A** onde T_j aparece como a tarefa sucessora. Seja novamente Γ_i^A o conjunto de todas as escalas de execução possíveis geradas pela aplicação **A** no processador $H(T_i)$.

O *jitter* de liberação máximo T_j na aplicação **A** é dado por $R_k^A+\delta_{k,j}$. O mesmo *jitter* de liberação máximo está presente na aplicação **B**. A única consequência de remover aquela relação de precedência está no possível aumento do conjunto de todas as escalas de execução possíveis, isto é, $\Gamma_i^A \subseteq \Gamma_i^B$.

Como $\Gamma_i^A \subseteq \Gamma_i^B$, qualquer escala de execução possível em Γ_i^A é também uma escala de execução possível em Γ_i^B, inclusive a escala de execução que define R_i^A. Não é possível fazer afirmações a respeito da existência ou não em Γ_i^B de escalas de execução que poderiam definir um tempo de resposta maior para T_i. Assim, temos que $R_i^A \leq R_i^B$. □

Teoremas 6 a 11 tratam com situações onde uma tarefa T_i, sujeita a relações de precedência dentro de sua atividade A_x, recebe interferência de tarefas pertencentes a outras atividades. Estes teoremas transformam a atividade da tarefa T_i até o ponto onde T_i passa a ser uma tarefa independente. Teorema 6 mostra que, quando a tarefa T_i não possui predecessores, é possível eliminar as relações de precedência que conectam a tarefa T_i as outras tarefas da atividade A_x.

Teorema 6

Seja **A** uma aplicação, $A_x \in A$ uma atividade e $T_i \in A_x$ uma tarefa tal que $dPred(T_i)=\{ \}$. Suponha que $B = (A-\{A_x\}) \cup \{A_i, A_e\}$ é uma aplicação onde A_i é uma atividade, $P_i=P_x$, composta exclusivamente pela tarefa T_i. A_e é uma atividade composta pelas mesmas tarefas e

relações de precedência da atividade A_X, exceto pela tarefa T_i que é eliminada e pelo seu período $P_e >> P_X$.

Nas condições acima temos que $R_i^A = R_i^B$.

Prova

Dentro de uma execução de T_i em **A** todas as demais tarefas de A_X podem ser liberadas no máximo uma única vez. Uma segunda liberação implicaria em uma segunda chegada de A_X e, portanto, uma segunda chegada de T_i. Quando a aplicação **B** é considerada, o período de A_e garante que suas tarefas podem ser liberadas no máximo uma vez dentro de uma única execução de T_i. Assim, a quantidade de interferência sobre T_i em **A** devido a todas as outras tarefas de A_X será igual a quantidade de interferência sobre T_i em **B** devido as tarefas de A_e.

Também, a eliminação das relações de precedência que incluem T_i como predecessor não afeta a interferência sobre T_i causada por tarefas que pertencem as outras atividades e possuem prioridade superior com relação a T_i.

Como as diferenças entre **A** e **B** não afetam a interferência sobre T_i, temos $R_i^A = R_i^B$. □

Teorema 7 trata da situação onde a tarefa T_i possui uma única tarefa predecessora T_k que executa no mesmo processador que T_i executa. Neste caso, é possível juntar as duas tarefas.

Teorema 7

Seja **A** uma aplicação, $A_X \in A$ uma atividade e $T_i \in A_X$ uma tarefa tal que $dPred(T_i)=\{T_k\}$ e $H(T_i)=H(T_k)$.

Suponha $B = (A-\{A_X\}) \cup \{A_e\}$ uma aplicação onde A_e é uma atividade composta pelas mesmas tarefas, relações de precedência e período da atividade A_X, exceto pelas tarefas T_k e T_i que são substituidas pela tarefa T_e. A tarefa T_e é caracterizada por:

$H(T_e)=H(T_i)$, $\rho(T_e)=\rho(T_i)$, $J_e = J_k$,

$C_e=C_k+C_i$, $dPred(T_e)=dPred(T_k)$, $dSuc(T_e)=dSuc(T_i)$.

Nas condições acima temos que $R_i^A = R_e^B$.

Prova

Vamos transformar a aplicação **A** em **B** passo a passo e mostrar que o tempo máximo de resposta de T_i em **A** é iqual ao tempo máximo de resposta da tarefa T_e em **B**.

É possível que algumas tarefas em **A** sucedam T_k mas não sucedam T_i. Estas tarefas possuem uma prioridade menor que $\rho(T_k)$, mas podem ter uma prioridade maior que $\rho(T_i)$ e interferir com T_i. Este fato não depende das relações de precedência de T_k. Assim, estas relações podem ser eliminadas sem afetar R_i^A.

Suponha o intervalo de tempo [t,t') definido pela liberação de T_k e pela conclusão de T_i no cenário de pior caso para T_i. Vamos atribuir a T_k a prioridade $\rho(T_i)$. Existe suficiente tempo de processador dentro do intervalo [t, t') para executar completamente as tarefas T_k, T_i e toda tarefa T_j tal que $\rho(T_j)>\rho(T_i) \wedge H(T_j)=H(T_i)$, que foram liberadas antes de **t'** mas não terminadas antes de **t**. Assim, atribuindo a T_k a prioridade $\rho(T_i)$, temos uma nova ordenação de tarefas dentro daquele intervalo de tempo, de forma que T_k poderá ser executada no final do intervalo, perto da execução de T_i. De qualquer forma, T_i novamente estará finalizada em **t'** no pior caso.

Vamos agora incluir em $dPred(T_i)$ todas as tarefas que pertencem ao conjunto $dPred(T_k)$. Esta mudança não altera o instante de liberação T_i, pois todas as tarefas que pertencem a $dPred(T_k)$ eram originalmente elementos do conjunto $iPred(T_i)$ e já estarão concluídas quando T_i é liberada.

Vamos agora eliminar a relação de precedência entre T_k e T_i logo a tarefa T_i é liberada ao mesmo instante que a tarefa T_k é liberada. Após esta transformação a tarefa T_i apresenta o mesmo *jitter* de liberação máximo da tarefa T_k, pois ambas possuem os mesmos instantes de chegada e de liberação. Novamente, temos uma reordenação das tarefas executadas no intervalo de tempo [t,t'), tal que T_i pode executar antes da tarefa T_k estar finalizada. De qualquer forma, as tarefas T_k e T_i estarão concluídas ao final do intervalo [t,t') mesmo no pior caso, pois a

demanda por tempo de processador não foi alterada e prioridades iguais implicam em uma ordem arbitrária de execução.

Uma vez que, após todas as transformações, T_k e T_i possuem o mesmo instante de liberação, o mesmo *jitter* máximo de liberação, a mesma prioridade, executam no mesmo processador, possuem os mesmos predecessores e os mesmos sucessores, podemos considera-las como uma única tarefa. A tarefa equivalente T_e tem um tempo máximo de execução igual a soma dos tempos máximos de execução de T_i e T_k e estará concluida no instante **t'**, no pior caso. Assim, temos $R_i^A = R_e^B$. □

Teorema 8 supõe uma tarefa T_i com uma única tarefa T_k como predecessora, e T_k executa em outro processador. É possível substituir a relação de precedência por *jitter* de liberação associado com a tarefa T_i.

Teorema 8

Seja **A** uma aplicação, $A_x \in A$ uma atividade e $T_i \in A_x$ uma tarefa tal que $dPred(T_i)=\{T_k\}$, onde $H(T_i) \neq H(T_k)$. Também $B=(A-\{A_x\}) \cup \{A_e\}$ é uma aplicação onde A_e é uma atividade composta pelas mesmas tarefas, relações de precedência e período da atividade A_x, exceto pela tarefa T_i que é substituida pela tarefa T_e, e por todas as tarefas do conjunto $Pred(T_i)$ em **A** que, por definição, não causam interferência sobre T_e em **B**. A tarefa T_e é definida por:

$H(T_e)=H(T_i)$, $\rho(T_e)=\rho(T_i)$, $C_e=C_i$, $J_e=R_k^A+\Delta$, $dPred(T_e)=\{\ \}$.

Nas condições acima temos $R_i^A = R_e^B$.

Prova

Uma relação de precedência cria *jitter* de liberação para a tarefa sucessora ao mesmo tempo em que reduz o conjunto de escalas de execução possíveis para a aplicação. Predecessores podem atrasar a liberação da tarefa sucessora mas eles não causam interferência sobre ela. Vamos comparar a tarefa T_i em **A** com a tarefa T_e em **B** com respeito ao *jitter* de liberação máximo e à interferência que elas recebem.

O *jitter* de liberação de T_i causado por T_k em **A** é máximo quando T_k tem o seu tempo de resposta máximo e a comunicação entre T_k e T_i exibe seu atraso máximo, isto é, $R_k^A+\Delta$. Por definição, a tarefa T_e tem um *jitter* de liberação máximo em **B** que é igual ao *jitter* de liberação máximo de T_i em **A**.

No momento que a tarefa T_i é liberada em **A**, todos os seus predecessores indiretos estão, necessariamente, já concluidos. Eles não serão liberados novamente antes da conclusão da tarefa T_i e, portanto, não interferem com a execução de T_i além da sua contribuição para o *jitter* de liberação de T_i. Pela forma como T_e foi definida, esta tarefa não recebe qualquer interferência em **B** das tarefas que precedem indiretamente T_i em **A**.

Como os efeitos da relação de precedência entre T_k e T_i em **A** são compensados pelas características de T_e em **B**, podemos dizer que $R_i^A = R_e^B$. □

Teoremas 9, 10 e 11 consideram uma tarefa que possui vários predecessores diretos. Os teoremas mostram que, com exceção de uma, todas as demais relações de precedência podem ser eliminadas. Como definido antes, Δ representa um limite para atrasos associados com comunicação remota.

Teorema 9

Seja **A** uma aplicação, $A_x \in A$ uma atividade, $T_i \in A_x$ uma tarefa tal que $\#dPred(T_i) \geq 2$ e todos os predecessores diretos de T_i são locais, isto é, $\forall T_j \in dPred(T_i)$, $H(T_j) = H(T_i)$.

A tarefa T_k, predecessor crítico de T_i em **A**, é definida por:

$$R_k^A = \max_{\forall T_l . T_l \in dPred(T_i)} R_l^A .$$

Finalmente, suponha $B = (A - \{A_X\}) \cup \{A_e\}$ é uma aplicação onde A_e é uma atividade composta pelas mesmas tarefas, relações de precedência e período da atividade A_X, exceto pela tarefa T_i que em A_e tem $dPred(T_i)=\{ T_k \}$.

Nas condições acima temos $R_i^A = R_i^B$.

Prova

No pior caso, a tarefa T_k é a última entre os predecessores diretos de T_i a terminar sua execução e enviar uma mensagem para T_i. Como todos os predecessores diretos de T_i estão no mesmo processador é garantido que, quando T_k termina no pior caso, todas as outras tarefas que são predecessores diretos de T_i também estarão terminadas. Ao manter a relação de precedência entre T_k e T_i também é mantido o tempo máximo de resposta da tarefa T_i. □

Teorema 10

Seja **A** uma aplicação, $A_X \in A$ uma atividade, $T_i \in A_X$ uma tarefa tal que $\#dPred(T_i) \geq 2$ e todos os predecessores diretos de T_i são remotos, isto é, $\forall T_j \in dPred(T_i)$, $H(T_j) \neq H(T_i)$.

A tarefa T_k, predecessor crítico de T_i em **A**, é definida por:

$$R_k^A + \Delta = \max_{\forall T_l . T_l \in dPred(T_i)} R_l^A + \Delta .$$

Finalmente, suponha $B = (A - \{A_X\}) \cup \{A_e\}$ uma aplicação onde A_e é uma atividade composta pelas mesmas tarefas, relações de precedência e período da atividade A_X, exceto pela tarefa T_i que em A_e tem $dPred(T_i)=\{ T_k \}$ e exceto também que toda tarefa T_j em **A** do tipo $T_j \in Pred(T_i) \wedge H(T_j)=H(T_i)$, por definição, não causa interferência sobre T_i em **B**.

Nas condições acima temos $R_i^A = R_i^B$.

Prova

No pior caso, a tarefa T_k é a última entre os predecessores diretos de T_i a terminar sua execução e enviar uma mensagem para T_i. É garantido que, no momento que T_k terminar sua execução no pior caso, todas as outras tarefas que são predecessores diretos ou indiretos de T_i já terão também terminado. Ao manter a relação de precedência entre T_k e T_i também estará sendo mantido o tempo máximo de resposta de T_i.

No momento que a tarefa T_i está pronta para executar em **A**, todos os predecessores indiretos locais já estão concluidos e não serão liberados novamente antes da conclusão de T_i. Assim, eles não causam interferência sobre T_i em **A**. O mesmo acontece em **B** por definição. □

Teorema 11

Seja **A** uma aplicação, $A_X \in A$ uma atividade e $T_i \in A_X$ uma tarefa tal que $\#dPred(T_i) \geq 2$. Suponha também que $\exists T_j \in dPred(T_i)$, $H(T_j)=H(T_i)$ e que $\exists T_j \in dPred(T_i)$, $H(T_j) \neq H(T_i)$.

A tarefa $T_{loc} \in A_X$ é uma tarefa tal que $T_{loc} \in dPred(T_i)$ e $H(T_{loc})=H(T_i)$, onde

$$R_{loc}^A = \max_{\forall T_l . T_l \in dPred(T_i) \wedge H(T_l)=H(T_i)} R_l^A .$$

Similarmente, suponha $T_{rem} \in A_X$ uma tarefa tal que $T_{rem} \in dPred(T_i)$ e $H(T_{rem}) \neq H(T_i)$, onde

$$R_{rem}^A + \Delta = \max_{\forall T_l . T_l \in dPred(T_i) \wedge H(T_l) \neq H(T_i)} R_l^A + \Delta .$$

A tarefa T_k, predecessora crítica de T_i em **A**, é definida pelas seguintes regras:

[a] Caso $R^A_{rem}+\Delta \geq R^A_{loc}$ então $T_k = T_{rem}$,

[b] Caso $R^A_{rem}+\Delta < R^A_{loc} - I^A_{loc}$ então $T_k=T_{loc}$,

onde I^A_{loc} é a máxima interferência recebida por T_{loc} em **A**.

Finalmente, suponha $B = (A - \{A_x\}) \cup \{A_e\}$ uma aplicação onde A_e é uma atividade composta pelas mesmas tarefas, relações de precedência e período da atividade A_x, exceto pela tarefa T_i que em A_e possui $dPred(T_i)=\{ T_k \}$ e exceto também que, se $H(T_k)\neq H(T_i)$ então todas as tarefas T_j tais que $T_j \in Pred(T_i) \wedge H(T_j)=H(T_i)$ em **A**, por definição, não causam interferência sobre T_i em **B**.

Nas condições acima temos $R_i^A = R_i^B$.

Prova

Caso [a]

No momento que a mensagem da tarefa T_{rem} ficar disponível para a tarefa T_i, todos os predecessores diretos e indiretos de T_i estarão concluidos e suas mensagens já estarão disponíveis para T_i. Assim, o tempo máximo de resposta de T_i é definido pela precedência que inclui T_{rem}.

Quando a tarefa T_i está pronta para executar em **A**, todos os predecessores locais já estão concluidos. Eles não serão liberados novamente antes da conclusão de T_i e, portanto, não causam interferência sobre T_i em **A**. O mesmo acontece em **B** por definição.

Caso [b]

O pior cenário para T_i pode ser diferente do pior cenário para T_{loc}. Por exemplo, suponha outra tarefa no mesmo processador de T_i com uma prioridade superior e um período muito maior que P_x. No pior caso para T_i esta outra tarefa é liberada exatamente no mesmo instante que T_i é liberada, após T_{loc} estar terminada. Este certamente não é o pior caso para T_{loc}. Assim, I^A_{loc} é calculado na suposição o pior caso possível para T_{loc}, mas este não é necessariamente o pior caso possível para T_i. Como $R^A_{rem}+\Delta<R^A_{loc}$ a mensagem de T_{rem} vai chegar antes que T_{loc} termine no pior caso para T_{loc}. Mas isto não garante que a mensagem de T_{rem} chegará antes que T_{loc} termine no pior caso para T_i. Assim, não é suficiente ter $R^A_{rem}+\Delta < R^A_{loc}$ para afirmar que T_{loc} é o predecessor crítico de T_i.

A suposição que $R^A_{rem} + \Delta < R^A_{loc} - I^A_{loc}$ garante que, independentemente dos padrões de interferência sobre T_{loc} e T_i, a mensagem de T_{rem} estará disponível para T_i antes que T_{loc} termine. Assim, o pior caso para T_i estará associado com a conclusão de T_{loc} e não com a chegada da mensagem enviada por T_{rem}. Ao preservar a relação de precedência entre T_{loc} e T_i também estará preservado o pior caso para T_i e seu tempo máximo de resposta. □

Teorema 11 não define T_k quando: $R^A_{loc} - I^A_{loc} \leq R^A_{rem}+\Delta < R^A_{loc}$. Neste caso, é possível determinar um limite superior para o tempo máximo de resposta de T_i através do aumento artificial de Δ entre T_{rem} e T_i. A figura 2 mostra uma aplicação composta de duas atividades. Uma atividade tem três tarefas enquanto a outra possui uma única tarefa. Dois processadores são usados. A figura mostra também os períodos (P0, P1, P2), o *jitter* de liberação máximo (J0, J1, J2), o tempo máximo de execução (C0, C1, C2, C3) e as relações de precedência.

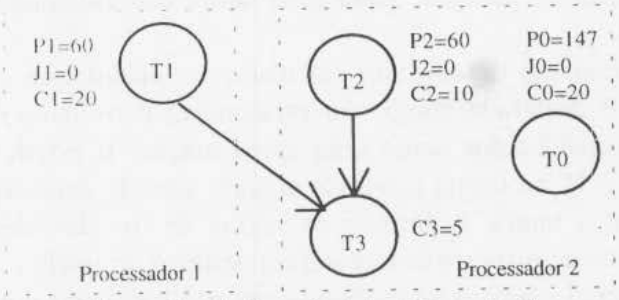

Figura 2 - Aplicação hipotética, Δ=7.

Como a tarefa T_3 possui dois predecessores diretos devemos aplicar o teorema 11 para determinar a precedência crítica. Existe um único predecessor direto remoto, logo $T_{rem}=T_1$. Existe também um único predecessor direto local, $T_{loc}=T_2$. Temos que

$R^A_{rem}+\Delta=R^A_1+\Delta=20+7=27$,

e que

$I^A_{loc}=I^A_2=20$ and $R^A_{loc}=R^A_2=10+20=30$.

Uma vez que

$R^A_{rem}+\Delta < R^A_{loc}$, e

$R^A_{rem}+\Delta \geq R^A_{loc} - I^A_{loc}$,

o teorema 11 não define T_k. Como esta é uma aplicação pequena, é possível testar todas as possibilidades. Caso $T_k=T_{rem}=T_1$, o tempo máximo de resposta de T_3 seria dado por 27+5+20= 52. Caso $T_k=T_{loc}=T_2$, o tempo máximo de resposta de T_3 seria dado por 10+5+20=35. Logo, o pior caso é $T_k=T_{rem}$.

Em geral esta análise possui uma complexidade exponencial e não é viável na prática para sistemas muito maiores que aquele da figura 2. Nestes casos é possível aumentar artificialmente o valor de Δ especificamente para a comunicação entre T_1 e T_3. Isto é feito de tal forma que a condição [a] do teorema 11 é satisfeita. No caso do exemplo, ao supor $\Delta=10$ teremos:

$R^A_{rem}+\Delta =30 = R^A_{loc}$ e então a tarefa T_1 será a precedência crítica de T_3.

Este aumento artificial no valor de Δ introduz uma certa quantidade de pessimismo na análise. Isto acontece quando a precedência crítica é originalmente a precedência remota e o aumento de Δ é, na verdade, desnecessário. É importante notar que apenas um atraso é aumentado, ou seja, o atraso associado com a comunicação entre T_{rem} e T_i. O valor aumentado é usado apenas para calcular o tempo máximo de resposta da tarefa T_i. O tempo máximo de resposta das demais tarefas será computado usando o valor original de Δ.

Para reconhecer esta situação de pessimismo exagerado seria necessário computar o tempo de resposta de T_i considerando primeiramente $T_k=T_{loc}$ e então $T_k=T_{rem}$. A complexidade desta análise depende do grafo de precedência em consideração. Para alguns grafos de precedência ela pode apresentar complexidade 2^n, onde **n** é o número de tarefas na aplicação.

Vamos agora analisar melhor alguns casos particulares que podem ocorrer quando os teoremas 3, 6, 8, 9, 10 e 11 são aplicados. Estes teoremas implicitamente assumem que o tempo máximo de resposta de cada tarefa é menor ou igual ao período da sua atividade. Esta suposição permite ignorar a segunda chegada de uma atividade enquanto as tarefas finais de sua chegada anterior ainda não estão concluídas. Pode ser mostrado que se uma tarefa T_i na aplicação **A** tem um tempo máximo de resposta maior que o seu período, então o tempo máximo de resposta de uma tarefa T_k em **B** pode ser menor que em **A**, mas ainda assim maior que seu próprio período. Uma vez que todos os deadlines são menores ou iguais aos respectivos períodos, a transformação descrita nestes teoremas é tal que: se T_k é escalonável **A**, ela também é escalonável em **B**; se T_k não é escalonável em **A**, ela também não é escalonável em **B**, mas poderá apresentar um tempo máximo de resposta que é menor do que aquele em **A**. Neste último caso a transformação é otimista, mas não otimista ao ponto de fazer uma tarefa não escalonável em **A** tornar-se uma tarefa escalonável em **B**.

Se qualquer tempo máximo de resposta calculado for maior que o período da respectiva atividade, a aplicação é declarada como não escalonável e os tempos máximos de resposta calculados devem ser considerados como uma aproximação. É possível resumir o efeito dos teoremas 3, 6, 8, 9, 10 e 11 na forma mostrada abaixo, onde R_i representa o tempo máximo de resposta calculado para a tarefa T_i usando as regras de transformação e R_i^A representa o verdadeiro tempo máximo de resposta da tarefa T_i no sistema original:

- Se $\exists T_i \in A_x$, $R_i > P_x$ então a aplicação é considerada não escalonável e os tempos máximos de resposta calculados para todas as tarefas são aproximações;
- Se $\forall T_i \in A_x$, $R_i \leq P_x \wedge \exists T_j \in A_y$, $R_j > D_j$ então a aplicação é considerada não escalonável e os tempos máximos de resposta calculados para todas as tarefas são aproximações pessimistas, isto é, $\forall T_k$, $R_k \geq R_k^A$;

• Se $\forall T_i \in A_X$, $R_i \leq P_X \wedge \forall T_i \in A_X$, $R_i \leq D_i$ então a aplicação é escalonável e os tempos máximos de resposta calculados para todas as tarefas são aproximações pessimistas, isto é, $\forall T_k, R_k \geq R_k^A$.

4 Algoritmo de Transformação e Análise

Esta seção apresenta um algoritmo que calcula o tempo máximo de resposta de cada tarefa. Uma vez calculado, o tempo máximo de resposta pode ser comparado com o deadline da tarefa. Basicamente, são feitas transformações sucessivas da aplicação até todas as relações de precedência estarem eliminadas. Estas transformações são feitas de tal foma que o tempo máximo de resposta da tarefa T_i na aplicação transformada é maior ou igual ao seu tempo máximo de resposta na aplicação original. As transformações levam em consideração como a tarefa T_i é afetada pelas demais tarefas. Assim, para cada tarefa T_i é necessário aplicar as transformações a partir da aplicação original.

O algoritmo é composto de três partes. A primeira parte corresponde a um laço cujo corpo é repetido para cada tarefa T_i. Primeiramente as relações de precedência dentro da atividade que inclui T_i são eliminadas. Em seguida o tempo máximo de resposta desta tarefa independente é calculado. Estes dois passos são realizados pelas segunda e terceira partes.

Tabela 1 apresenta a parte principal do algoritmo. Tarefas com prioridade superior são analisadas antes. Assim, quando R_i^A estiver sendo calculado, já será conhecido o valor R_j^A para toda tarefa T_j tal que $\rho(T_j)>\rho(T_i)$. As linhas 2 e 3 são apenas chamadas para as segunda e terceira partes, respectivamente.

```
[1] Para cada T_i, i de 1 a n, faça
[2]   Compute uma aplicação equivalente onde T_i é uma tarefa independente
[3]   Compute o tempo máximo de resposta de T_i na aplicação equivalente
```

Tabela 1 - Primeira parte do algoritmo: parte principal.

Tabela 2 apresenta a segunda parte do algoritmo. A atividade que inclui T_i passa por sucessivas transformações até que uma tarefa independente seja obtida. Quando T_i tem um único predecessor direto que executa no mesmo processador as duas tarefas são unificadas (linhas 5 e 6). Caso T_i possua um único predecessor direto, que executa em outro processador, a relação de precedência é transformada em *jitter* de liberação (linhas 7 e 8). Finalmente, quando T_i possui vários predecessores diretos, todas as relações de precedência com exceção de uma são eliminadas (linhas 9 e 10).

É importante notar que o laço definido pelas linhas de 5 a 10 é executado até T_i não mais possuir qualquer predecessor (linha 4). Neste momento, a linha 11 elimina todas as relações de precedência que ainda conectam T_i às suas tarefas sucessoras na atividade. Ao final da segunda parte temos uma tarefa independente.

```
[4]  Enquanto #dPred(T_i) > 0
[5]    Caso #dPred(T_i)=1 ∧ H(T_i)=H(T_k), T_k∈ dPred(T_i)
[6]      Aplica teorema 7, junta T_k e T_i
[7]    Caso #dPred(T_i)=1 ∧ H(T_i)≠H(T_k), T_k∈ dPred(T_i)
[8]      Aplica teorema 8, reduz para #dPred(T_i)=0
[9]    Caso #dPred(T_i) > 1
[10]     Aplica teoremas 9,10,11, reduz para #dPred(T_i)=1
[11] Aplica teorema 6, separa T_i de A_x
```

Tabela 2- Segunda parte: torna T_i uma tarefa independente.

Tabela 3 mostra a terceira parte do algoritmo. É um laço que elimina todas as relações de precedência em todas as atividades que tem pelo menos uma tarefa no mesmo processador onde T_i executa (linhas 12 e 13). Entre estas atividades estão incluidas o que sobrou da atividade original da tarefa T_i, após a remoção de T_i.

Cada atividade é decomposta em fragmentos. Cada fragmento é definido por uma única tarefa inicial. Esta decomposição é alcançada através da eliminação das relações de precedência múltiplas (linhas 14 a 16) e da eliminação das relações de precedência que incluem outro processador (linhas 17 a 19).

Finalmente, cada fragmento (linha 20) é reduzido a uma tarefa independente capaz de gerar uma interferência equivalente sobre T_i. Se a prioridade de cada tarefa do fragmento for inferior a prioridade de T_i, o fragmento é eliminado (linhas 21 e 22). Se a prioridade de cada tarefa do fragmento for superior a prioridade de T_i, o fragmento é transformado em uma tarefa única que gera a mesma interferência (linhas 23 e 24). No caso de algumas tarefas possuirem uma prioridade inferior enquanto outras tarefas possuem uma prioridade superior a T_i, o fragmento também é transformado em uma tarefa equivalente (linhas 25 e 26). Ao final (linha 27), é possível aplicar um algoritmo que calcula o tempo máximo de resposta de uma tarefa em um sistema composto por tarefas independentes. Por exemplo, o algoritmo descrito em [2].

```
[12] Para cada A_y∈A, T_i∉A_y, faça
[13]    Se ∃T_j∈T,T_j∈A_y ∧ H(T_j)=H(T_i) então
[14]       Para cada T_j∈T, T_j∈A_y ∧ H(T_j)=H(T_i) ∧ ρ(T_j)>ρ(T_i)
[15]          Se #dPred(T_j)≥2 então
[16]                Aplica teorema 4, reduz para #dPred(T_j)=1
[17]       Para cada T_j∈T , Tj∈A_y ∧ H(T_j)=H(T_i) ∧ ρ(T_j)>ρ(T_i)
[18]          Se T_k∈dPred(T_j) ∧ H(T_k)≠H(T_i) então
[19]                Aplica teorema 5, reduz para #dPred(T_j)=0
[20]       Para cada fragmento independente A_z de A_y em H(T_i)
[21]          Caso ∀T_j∈A_z , ρ(T_j)<ρ(T_i)
[22]                Aplica teorema 1, elimina fragmento A_z
[23]          Caso ∀T_j∈A_z , ρ(T_j)>ρ(T_i)
[24]                Aplica teorema 2, reduz A_z para uma única tarefa
[25]          Outros casos
[26]                Aplica teorema 3, reduz A_z para uma única tarefa
[27] Computa o tempo máximo de resposta de T_i
```

Tabela 3 - Terceira parte: calcula o tempo máximo de resposta.

Através de uma simples inspeção do algoritmo é possível notar que qualquer grafo de tarefas acíclico pode ser analisado. Vamos agora considerar a complexidade do algoritmo apresentado nesta seção. A primeira parte é um laço executado **n** vezes. A segunda parte pode ser executada no máximo **n** vezes, quando T_i é a última tarefa de uma atividade linear composta por **n** tarefas. O laço principal da terceira parte (linha 12) pode ser executado no máximo **n-1** vezes, quando não existem relações de precedência na aplicação. Neste caso, os laços internos (linhas 14, 17 e 20) são limitados a uma única iteração. Em outra situação extrema, T_i recebe interferência de uma atividade que inclui todas as tarefas restantes da aplicação. Neste caso o laço principal executa uma única vez e o número de iterações dos laços internos é limitado por **n** (nenhuma atividade pode ter mais que **n** tarefas). Suponha que a linha 27 possui complexidade **E**. Este valor depende do teste de escalonabilidade usado para tarefas independentes. O algoritmo como um todo possui uma complexidade dada por n(n+(n+E)), ou seja, $\mathbf{O(n^2+n.E)}$.

5 Simulação

O método descrito na seção anterior para transformar relações arbitrárias de precedência em *jitter* de liberação pode também ser avaliado com simulação. Seu desempenho é comparado com o desempenho de uma transformação simples e direta, onde cada relação de precedência é transformada em *jitter* de liberação. Ambos os métodos transformam o conjunto original de tarefas em um conjunto de tarefas independentes que pode ser analisado, por exemplo, com o algoritmo apresentado em [2].

A geração da carga foi baseada no método usado em [15]. Cada aplicação é composta de 5 atividades com **T** tarefas por atividade, mais 5×T tarefas independentes. Neste contexto, uma tarefa independente pode ser vista como uma atividade composta por uma única tarefa. Foram simuladas atividades com **T** igual a 3, 5 e 7. Todas as atividades são periódicas, com período entre 100 e 10000, de acordo com uma distribuição exponencial. Todas as tarefas possuem deadlines iguais ao período da atividade.

O sistema simulado possui Δ=20. Tarefas são alocadas aleatoriamente a quatro processadores. O tempo máximo de execução de cada tarefa é baseado em um número escolhido aleatoriamente, com uma distribuição uniforme, no intervalo 0.01 a 1. O tempo máximo de execução de cada tarefa é igual ao seu período multiplicado pela utilização do seu processador multiplicado ainda pelo seu número aleatório e dividido pela soma dos números aleatórios de todas as tarefas alocadas ao mesmo processador. Em cada aplicação todos os processadores possuem a mesma utilização. Prioridades são atribuidas às tarefas baseadas no deadline e relações de precedência.

A tabela 4 resume os resultados das simulações. A maior preocupação das simulações é o número de aplicações declaradas escalonáveis pelo algoritmo deste artigo comparado com o número de aplicações declaradas escalonáveis pelo método simples mencionado antes. Por exemplo, um valor de 45% significa que o método simples foi capaz de declarar escalonável um número de aplicações que corresponde a 45% do número de aplicações declaradas escalonáveis pelo algoritmo deste artigo.

Tarefa/Atividade	3	5	7
Utilização			
10%	100%	100%	100%
20%	100%	100%	100%
30%	100%	100%	100%
40%	100%	100%	100%
50%	100%	99%	97%
60%	99%	94%	92%
70%	94%	85%	80%
80%	81%	65%	54%
90%	53%	40%	34%

Tabela 4 - Resultado das simulações.

A tabela 4 mostra os resultados quando a utilização nos processadores varia de 10% a 90% e o número de tarefas por atividade varia de 3 a 7. O número de conjuntos de tarefas gerado foi tal que nosso algoritmo foi capaz de declarar escalonável pelo menos 1000 aplicações. O mesmo conjunto de aplicações foi fornecido como entrada para ambos os algoritmos. Uma vez que muitas caracteristicas das tarefas são escolhidas aleatoriamente, muitas das aplicações geradas não são mesmo escalonáveis.

Os testes apresentam desempenho igual quando a utilização dos processadores é 40% ou menor. Quando a utilização dos processadoes é 50% ou maior nosso teste é menos pessimista. Por exemplo, quando existem 3 tarefas por atividade e a utilização do processador é 90%, o método simples é capaz de declarar escalonável apenas 53% das aplicações que nosso método identificou como escalonáveis.

A diferença entre os métodos aumenta quando aumenta o número de tarefas por atividade. Por exemplo, o resultado para 90% de utilização vai de 53% até 34% quando o número de tarefas por

atividade aumenta de 3 para 7. A tabela 4 mostra que o algoritmo apresentado aqui é menos pessimista que uma simples transformação das relações de precedência em *jitter* de liberação.

6 Conclusões

Este artigo apresentou um novo método para transformar relações arbitrárias de precedência em *jitter* de liberação em aplicações tempo real distribuídas. É suposto que as tarefas são liberadas dinamicamente e executadas conforme suas prioridades fixas. Foram consideradas tarefas periódicas e esporádicas que possuem um deadline igual ou menor que o seu período. Após eliminar todas as relações de precedência presentes no conjunto de tarefas é possível aplicar qualquer teste de escalonabilidade válido para conjuntos de tarefas independentes.

A corretude do método apresentado foi demonstrada através do desenvolvimento de vários teoremas. Estes teoremas são os blocos básicos usados na construção do algoritmo apresentado na seção 4. Simulações foram realizadas para destacar as vantagens de nosso método quando comparado com uma simples transformação direta das relações de precedência em *jitter* de liberação, como apresentado em [6].

O método apresentado resulta em um teste de escalonabilidade suficiente mas não necessário. Apesar disto, este método é menos pessimista que uma simples transformação direta de cada relação de precedência no correspondente *jitter* de liberação. Os autores não conhecem nenhum outro método capaz de analisar a escalonabilidade de conjuntos de tarefas com relações arbitrárias de precedência e liberação dinâmica de tarefas em ambiente distribuído.

Agradecimentos

Este trabalho foi parcialmente financiado pela FAPERGS e pelo CNPq.

Referências

[1] N. C. Audsley, A. Burns, A. J. Wellings. Deadline Monotonic Scheduling Theory and Application. Control Engineering Practice, Vol. 1, No. 1, pp. 71-78, feb. 1993.

[2] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, A. J. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. Software Engineering Journal, Vol. 8, No. 5, pp.284-292, 1993.

[3] N. Audsley, K. Tindell, A. Burns. The End of the Line for Static Cyclic Scheduling? Proc. of the Fifth Euromicro Workshop on Real-Time Syst., pp.36-41, 1993.

[4] N. C. Audsley. Flexible Scheduling of Hard Real-Time Systems. Department of Computer Science Thesis, University of York, 1994.

[5] R. Bettati, J. W.-S. Liu. End-to-End Scheduling to Meet Deadlines in Distributed Systems. Proc. of the 12th Intern. Conf. on Distributed Computing Systems, pp. 452-459, june 1992.

[6] A. Burns, M. Nicholson, K. Tindell, N. Zhang. Allocating and Scheduling Hard Real-Time Tasks on a Point-to-Point Distributed System. Proceedings of the Workshop on Parallel and Distributed Real-Time Systems,pp.11-20, 1993.

[7] S. Chatterjee, J. Strosnider. Distributed Pipeline Scheduling: End-to-End Analysis of Heterogeneous, Multi-Resource Real-Time Systems. Proceedings of the 15th International Conf. on Dist. Computing Systems, may 1995.

[8] R. Gerber, S. Hong, M. Saksena. Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes. Proc. of the IEEE Real-Time Systems Symp., dec. 1994.

[9] M. G. Harbour, M. H. Klein, J. P. Lehoczky. Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems. IEEE Trans. on Soft. Eng., Vol.20, No.1, pp.13-28, jan. 1994.

[10] J. Y. T. Leung, J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. Performance Evaluation, 2 (4), pp. 237-250, december 1982.

[11] C. L. Liu, J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM, Vol. 20, No. 1, pp. 46-61, january 1973.

[12] L. Sha, R. Rajkumar, S. S. Sathaye. Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems. Proceedings of the IEEE, Vol. 82, No. 1, pp. 68-82, january 1994.

[13] J. Sun, R. Bettati, J. W.-S. Liu. An End-to-End Approach to Scheduling Periodic Tasks with Shared Resources in Multiprocessor Systems. Proc. of the IEEE Workshop on Real-Time Operating Syst. and Software, pp. 18-22, 1994.

[14] J. Sun, J. W.-S. Liu. Bounding the End-to-End Response Time in Multiprocessor Real-Time Systems. Proc. Workshop on Par. Dist. Real-Time Sys., pp91-98, Santa Barbara,april 1995.

[15] J. Sun, J. W.-S. Liu. Synchronization Protocols in Distributed Real-Time Systems. Proc. of 16th Intern. Conf. on Dist. Comp. Syst., may 1996.

[16] J. Sun, J. W.-S. Liu. Bounding the End-to-End Response Times of Tasks in a Distributed Real-Time System Using the Direct Synchronization Protocol. Tech. Report UIUC-DCS-R-96-1949, Univ. of Ill. at Urbana-Champaign, june 1996.

[17] K. W. Tindell, A. Burns, A. J. Wellings. An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks. Real-Time Systems, pp. 133-151, 1994.

[18] K. Tindell, J. Clark. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. Microproc. and Microprog., 40, 1994.