Approximation scheme for real-time tasks under fixed-priority scheduling with deferred preemption

T.H.C. Nguyen[1], N.S. Tran[1], V.H. Le[1], P. Richard[2]

(1) VNU, University of Enginnering and Technology, Hanoï, Vietnam
(2) LIAS, ENSMA and University of Poitiers, France

Outlines

Motivations

Problem

Allowing arbitrary preemptions can introduce a high amount of runtime overhead.

Limiting preemptions in real-time tasks helps to :

- Improve I/O scheduling,
- Avoid mutual exclusion synchronizations
- Limit Cache Related Preemption delays (overhead due to cache misses,...)

Limited Preemption Models

Compromise between (arbitrary) preemptive and non-preemptive scheduling models :

- Preemption thresholds : disable preemption up to a specified priority level
- Floating Preemption : maximum interval of non-preemptive regions for each task.
- ***Deferred preemption*** (Co-operative scheduling) : Fixed preemption points (e.g., yield() call in the code)

Ref. *Buttazzo et al., Limited preemptive scheduling for real-time Systems : a survey, IEEE Trans. Industrial Informatics, 2013*

Deferred Preemption Task Model

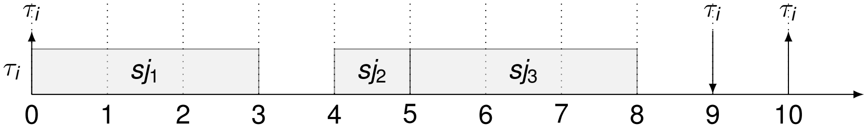
- Platform : Uniprocessor systems.
- Scheduler : Static-priority online scheduling algorithms.
- Task Sets : Sporadic tasks with arbitrary deadlines.
- Priorities : $Prio(\tau_i) < Prio(\tau_j)$ iff $i < j$.

Deferred Preemption Model :

- Every Job of τ_i is a set of m_i non-preemptive subjobs
- preemptions are only allowed at subjob boundaries
- non-preemptive scheduling is a particular case

Example and Notations

Example with 3 subjobs : $\tau_i = \{sj_1, sj_2, sj_3\}$



- $C_i = \sum_{k=1}^{m_i} C_{i,k}$=7 : worst-case execution time of τ_i
- $F_i = 3$ - Computation time final subjob of τ_i.
- $D_i = 9$ - Relative deadline of τ_i
- $T_i = 10$ - Minimum inter-arrival time of τ_i
- B_i - Longest non-preemptive subjob among lower priority tasks
- $U_i = C_i / T_i$ - Utilization factor

Existing Response Time Analysis

- Exact worst-case Response Time Analysis (WR_i)
  - Pseudo-polynomial time algorithm (Bril, et al, RTSJ. 2009)
  - NP-hard in the weak-sense (fixed-point computation)
  - No constant c for approximation :
    $WR_i \leq Approx(WR_i) \leq c \times WR_i$
- Response time upper bound
  - Linear bound (Davis, Burns, RTSS'08) :

$$supD(WR_i) = \frac{B_i + C_i - F_i + \sum_{j<i}(C_j(1 - U_j))}{1 - \sum_{j<i} U_j} + F_i$$

Existing Response Time Analysis

- Exact worst-case Response Time Analysis (WR_i)
  - Pseudo-polynomial time algorithm (Bril, et al, RTSJ. 2009)
  - NP-hard in the weak-sense (fixed-point computation)
  - No constant c for approximation :
    $WR_i \leq Approx(WR_i) \leq c \times WR_i$
- Response time upper bound
  - Linear bound (Davis, Burns, RTSS'08) :

$$supD(WR_i) = \frac{B_i + C_i - F_i + \sum_{j<i}(C_j(1-U_j))}{1-\sum_{j<i}U_j} + F_i$$

Existing Response Time Analysis

- Exact worst-case Response Time Analysis (WR_i)
  - Pseudo-polynomial time algorithm (Bril, et al, RTSJ. 2009)
  - NP-hard in the weak-sense (fixed-point computation)
  - No constant c for approximation : $WR_i \leq Approx(WR_i) \leq c \times WR_i$
- Response time upper bound
  - Linear bound (Davis, Burns, RTSS'08) :

$$supD(WR_i) = \frac{B_i + C_i - F_i + \sum_{j<i} (C_j(1 - U_j))}{1 - \sum_{j<i} U_j} + F_i$$

This work

FPTAS (Fully Polynomial Time Approximation Scheme) :
Response time upper bounds under resource augmentation.

- Parametric algorithm with input error $0 < \epsilon \leq 1$, $k = \lceil \frac{1}{\epsilon} \rceil - 1$.
- Let WR_i be the exact worst-case response time upon a unit speed processor :
  - $WR_i \leq UB(WR_i)$ for a unit-speed processor.
  - $UB(WR_i) \leq WR_i$ for $(\frac{k}{k+1})$-speed processor.

This processor speedup is an upper bound on the price being paid for using an efficiently computable upper bound on response time !

This work

FPTAS (Fully Polynomial Time Approximation Scheme) : Response time upper bounds under resource augmentation.

- Parametric algorithm with input error $0 < \epsilon \leq 1$, $k = \lceil \frac{1}{\epsilon} \rceil - 1$.
- Let WR_i be the exact worst-case response time upon a unit speed processor :
  - $WR_i \leq UB(WR_i)$ for a unit-speed processor.
  - $UB(WR_i) \leq WR_i$ for $(\frac{k}{k+1})$-speed processor.

This processor speedup is an upper bound on the price being paid for using an efficiently computable upper bound on response time !

How to reduce the computational complexity for analysing τ_i ?

Level-*i* active period : interval of time where only tasks with priority higher of equal to τ_i are running

Exact Analysis	*Approximate Analysis*
solving fixed-point equations	intersection of two linear functions
Pseudo-polynomial number of τ_i's jobs in Level-*i* active period	Polynomial number of τ_i's jobs Level-*i* active period

Exact worst-case response time analysis

- Analysis of all jobs in the level-i active period.
- Request bound functions in $[0, t)$ and $[0, t]$:

$$\mathrm{RBF}(\tau_i, t) \stackrel{\mathrm{def}}{=} \left\lceil \frac{t}{T_i} \right\rceil C_i \qquad \mathrm{RBF'}(\tau_i, t) \stackrel{\mathrm{def}}{=} \left(\left\lfloor \frac{t}{T_i} \right\rfloor + 1 \right) C_i$$

- Cumulative workload functions of tasks having a priority higher or equal to $\tau_{i,l}$ plus a computation of length C :

$$wr_{i,l}(C, t) \stackrel{\mathrm{def}}{=} C + (l+1)C_i + \sum_{i<j} \mathrm{RBF}(\tau_i, t)$$

$$wo_{i,l}(C, t) \stackrel{\mathrm{def}}{=} C + (l+1)C_i + \sum_{i<j} \mathrm{RBF'}(\tau_i, t)$$

Exact worst-case response time analysis

- Analysis of all jobs in the level-i active period.
- Request bound functions in $[0, t)$ and $[0, t]$:

$$\mathrm{RBF}(\tau_i, t) \stackrel{\text{def}}{=} \left\lceil \frac{t}{T_i} \right\rceil C_i \qquad \mathrm{RBF'}(\tau_i, t) \stackrel{\text{def}}{=} \left(\left\lfloor \frac{t}{T_i} \right\rfloor + 1 \right) C_i$$

- Cumulative workload functions of tasks having a priority higher or equal to $\tau_{i,l}$ plus a computation of length C :

$$wr_{i,l}(C, t) \stackrel{\text{def}}{=} C + (l + 1)C_i + \sum_{i<j} \mathrm{RBF}(\tau_i, t)$$

$$wo_{i,l}(C, t) \stackrel{\text{def}}{=} C + (l + 1)C_i + \sum_{i<j} \mathrm{RBF'}(\tau_i, t)$$

Exact worst-case response time analysis

Fixed point equations for the job $\tau_{i,l}$:

- Worst-case Response time $WR_{i,l}(C)$: smallest solution of $wr_{i,l}(C,t) = t$.
- Worst-case Occupied time $WO_{i,l}(C)$: smallest solution of $wo_{i,l}(C,t) = t$

Smallest fixed-point equations of $WR_{i,l}(C)$ and $WO_{i,l}(C)$ are used for computing the <u>*starting time*</u> of the final subjob of $\tau_{i,l}$:

$$R_{i,l} = \begin{cases} WR_{i,l}(B_i - F_i) & \text{for } i < n, \\ WO_{n,l}(-F_n) & \text{for } i = n. \end{cases}$$

Worst-case response time of $\tau_{i,l}$: $WR_{i,l} = R_{i,l} + F_i - l \times T_i$

Exact worst-case response time analysis

Fixed point equations for the job $\tau_{i,l}$:

- Worst-case Response time $WR_{i,l}(C)$: smallest solution of $wr_{i,l}(C,t) = t$.
- Worst-case Occupied time $WO_{i,l}(C)$: smallest solution of $wo_{i,l}(C,t) = t$

Smallest fixed-point equations of $WR_{i,l}(C)$ and $WO_{i,l}(C)$ are used for computing the *starting time* of the final subjob of $\tau_{i,l}$:

$$R_{i,l} = \begin{cases} WR_{i,l}(B_i - F_i) & \text{for } i < n, \\ WO_{n,l}(-F_n) & \text{for } i = n. \end{cases}$$

Worst-case response time of $\tau_{i,l}$: $WR_{i,l} = R_{i,l} + F_i - l \times T_i$

Exact worst-case response time analysis

Fixed point equations for the job $\tau_{i,l}$:

- Worst-case Response time $WR_{i,l}(C)$: smallest solution of $wr_{i,l}(C, t) = t$.
- Worst-case Occupied time $WO_{i,l}(C)$: smallest solution of $wo_{i,l}(C, t) = t$

Smallest fixed-point equations of $WR_{i,l}(C)$ and $WO_{i,l}(C)$ are used for computing the *starting time* of the final subjob of $\tau_{i,l}$:

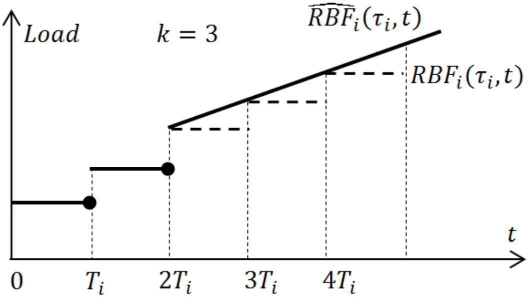
$$R_{i,l} = \begin{cases} WR_{i,l}(B_i - F_i) & \text{for } i < n, \\ WO_{n,l}(-F_n) & \text{for } i = n. \end{cases}$$

Worst-case response time of $\tau_{i,l}$: $WR_{i,l} = R_{i,l} + F_i - l \times T_i$

Approximation scheme technique

k : number of steps (scheduling points) to consider before the linear approximation.

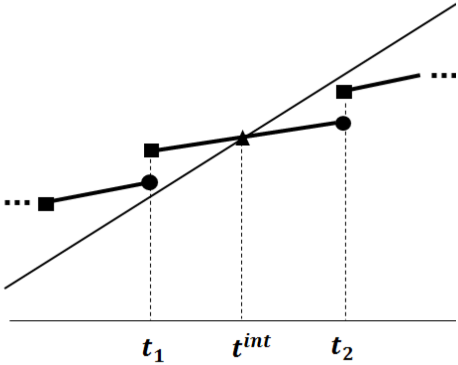
$$\widehat{\mathrm{RBF}}(\tau_i, t) = \begin{cases} \mathrm{RBF}_i(t) & \text{for } t \leq (k-1)T_i, \\ (t + T_i)\frac{C_i}{T_i} & \text{otherwise.} \end{cases}$$



Approximate starting time of final subjobs

Between two subsequent job releases, compute the intersection between :

- the processor capacity function $f(t) = t$
- approximate cumulative workload (linear)



Approximate Workload Function and Testing Set

Scheduling Points (Testing set) :

$$\widehat{S}_i \stackrel{\text{def}}{=} \{t = aT_b \mid a = 1, \ldots, k-1; b = 1, \ldots, i-1\} \bigcup \{0\}$$

- Let A denote the maximum instant in $\widehat{S}_i$:
  - $(0, A]$: $\forall j \leq i$, approx. workload a step function.
  - (A, ∞) : $\forall j \leq i$, approx. workload is a linear continuous function.

$\Longrightarrow$ corresponding to 2 testing stages.

Stage 1 : Primitive interval properties

$\Longrightarrow$ There might be more than one job of τ_i to consider in a primitive interval $(t_1, t_2]$, but :

1. To check all jobs terminated against their deadlines : Check only the first job whose final subjob has started in $(t_1, t_2]$.
2. To check the end of the level-i active period : Check if the last active period completes before the next job release.

Stage 1 : Primitive interval properties

$\Longrightarrow$ There might be more than one job of τ_i to consider in a primitive interval $(t_1, t_2]$, but :

1. To check all jobs terminated against their deadlines : Check only the first job whose final subjob has started in $(t_1, t_2]$.
2. To check the end of the level-i active period : Check if the last active period completes before the next job release.

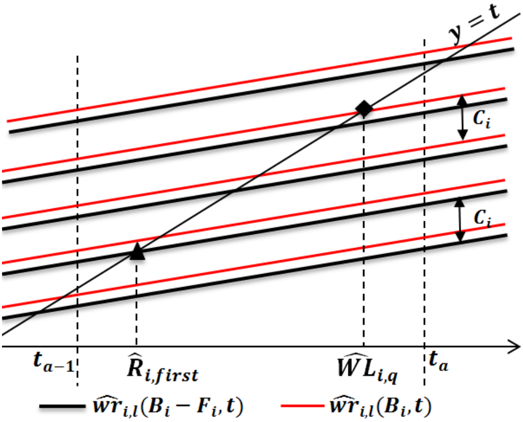
Stage 1 : Primitive interval properties

$\Longrightarrow$ There might be more than one job of τ_i to consider in a primitive interval $(t_1, t_2]$, but :

1. To check all jobs terminated against their deadlines : Check only the first job whose final subjob has started in $(t_1, t_2]$.
2. To check the end of the level-i active period : Check if the last active period completes before the next job release.

Stage 1 : Approximate Intersection Point

First Stage : Find an approximate intersection point in a primitive interval (two subsequent scheduling points).



Stage 2 : Linear approximation bound

Stage 2 analyses the primitive interval (A, ∞) if level-i is not completed before the last scheduling point of the Stage 1.

- Define the index of the first job to complete in the interval (A, ∞)
- Compute the intersection point between its approximate workload and the processor capacity

Property

The greatest upper bound computed during the two stages defines the approximate response time upper bound.

Stage 2 : Linear approximation bound

Stage 2 analyses the primitive interval (A, ∞) if level-i is not completed before the last scheduling point of the Stage 1.

- Define the index of the first job to complete in the interval (A, ∞)
- Compute the intersection point between its approximate workload and the processor capacity

Property

The greatest upper bound computed during the two stages defines the approximate response time upper bound.

Worst-case performance guarantee

Main properties of the algorithm :

- Performance guarantees :

Lemma

Let $s = \frac{k}{k+1}$*. If* $(l+1)C_i + C \geq 0$ *then :*
a. $WR_{i,l}(C) \leq \widehat{WR}_{i,l}(C) \leq WR^s_{i,l}(C)$*.*
b. $WO_{i,l}(C) \leq \widehat{WO}_{i,l}(C) \leq WO^s_{i,l}(C)$*.*

where $k = \left\lceil \frac{1}{\epsilon} \right\rceil - 1$

- Worst-case speedup factor : $(1 + \frac{k}{k+1})$
- Complexity of the algorithm : $\mathcal{O}(kn^2)$ (This is an FTPAS)

Worst-case performance guarantee

Main properties of the algorithm :

- Performance guarantees :

Lemma

Let $s = \frac{k}{k+1}$. *If* $(l+1)C_i + C \geq 0$ *then :*
a. $WR_{i,l}(C) \leq \widehat{WR}_{i,l}(C) \leq WR^s_{i,l}(C)$.
b. $WO_{i,l}(C) \leq \widehat{WO}_{i,l}(C) \leq WO^s_{i,l}(C)$.

where $k = \left\lceil \frac{1}{\epsilon} \right\rceil - 1$

- Worst-case speedup factor : $(1 + \frac{k}{k+1})$
- Complexity of the algorithm : $\mathcal{O}(kn^2)$ (This is an FTPAS)

Worst-case performance guarantee

Main properties of the algorithm :

- Performance guarantees :

Lemma

Let $s = \frac{k}{k+1}$. *If* $(l+1)C_i + C \geq 0$ *then :*

a. $WR_{i,l}(C) \leq \widehat{WR}_{i,l}(C) \leq WR^s_{i,l}(C)$.

b. $WO_{i,l}(C) \leq \widehat{WO}_{i,l}(C) \leq WO^s_{i,l}(C)$.

where $k = \left\lceil \frac{1}{\epsilon} \right\rceil - 1$

- Worst-case speedup factor : $(1 + \frac{k}{k+1})$
- Complexity of the algorithm : $\mathcal{O}(kn^2)$ (This is an FTPAS)

Experimentations

Comparison of FPTAS and SupD (Davis,Burns, 2008) on randomly generated task sets.

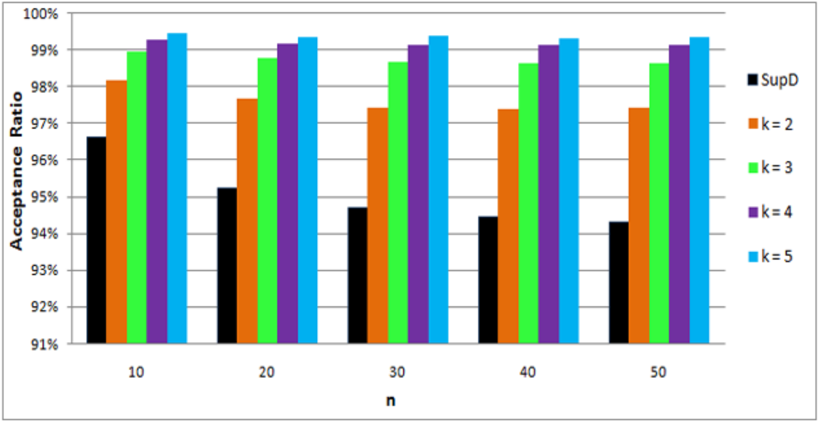
Monitored metrics :

- Acceptance ratio : rate of tasks stated ”feasible” by the considered upper bound ($ub_i \leq D_i$) for feasible tasks ($WR_i \leq D_i$).
- Average Error : $\frac{ub_i - WR_i}{WR_i}$

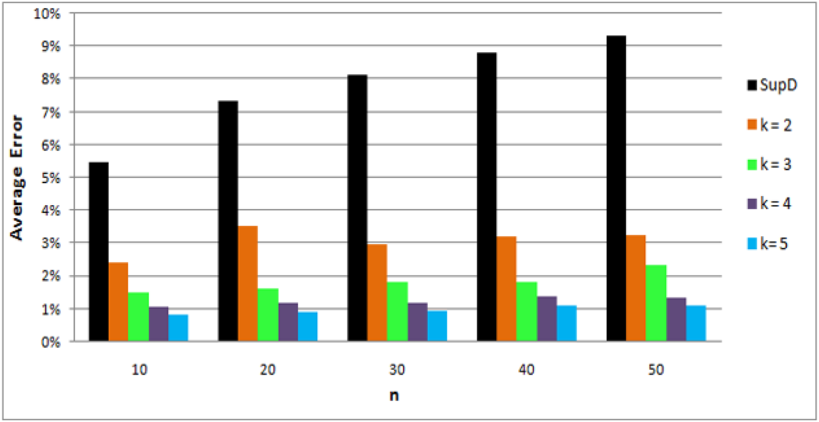
Plots :

- black : linear bound SupD (Davis, Burns 2008).
- others : our algorithm for $k = 2$ to $k = 5$.

Acceptance ratio



Average Error



Conclusion and Perspectives

Approximate Worst-case Response Time analysis of FPDS :

- Polynomial Time Algorithm for response times upper bounds with high accuracy (FTPAS)
- Worst-case performance guarantee under resource augmentation analysis (i.e., speed up factor)

Perspectives :

- Release jitters and network analysis
- Analysis of large scale distributed systems under resource augmentation with a worst-case performance guarantee.

Conclusion and Perspectives

Approximate Worst-case Response Time analysis of FPDS :

- Polynomial Time Algorithm for response times upper bounds with high accuracy (FTPAS)
- Worst-case performance guarantee under resource augmentation analysis (i.e., speed up factor)

Perspectives :

- Release jitters and network analysis
- Analysis of large scale distributed systems under resource augmentation with a worst-case performance guarantee.