CB-003

クリティカルの基準変更による 動的スケジューリングアルゴリズムの性能分析
The performance analysis on dynamic scheduling algorithm by critical criteria

佐々木 理成† 兪 明連‡ 横山 孝典††
Risei Sasaki Myungryun Yoo Takanori Yokoyama

1. 研究背景

近年，組込みリアルタイムシステムでは処理能力の向上のためマルチプロセッサ技術の利用が一般的である．従来のシングルプロセッサ環境における最適なリアルタイムスケジューリングアルゴリズム EDF(Earliest Deadline First)[1]は，マルチプロセッサ環境においてはアーキテクチャの並列性を十分に活用できない．マルチプロセッサ環境において，100%で CPU を使用できるアルゴリズムはすでに提案されている[2]ものの，オーバーヘッドが大きく複雑といった問題点をもつ．そのため，実用性の面で EDF を基にしたアルゴリズムが多く研究されている．本研究では，EDF にクリティカルタスクルールを取り入れた EDCL(Earliest Deadline Critical Laxity)[3]に着目し，そのルールの基準を 6 つのパターンに変更する．各パターンにおけるスケジュール成功率やスケジュール可能性解析などの性能分析の結果から，適切なクリティカルな基準について探る．

2. システムモデル

マルチプロセッサでのリアルタイムスケジューリングにおける一般的なシステムモデルを想定する．システムはプロセッサを M 個もち，周期的タスク N 個から構成されるタスクセット$\tau = \{\tau_1, \tau_2, \ldots, \tau_N\}$が与えられる．各タスク$\tau_i$は$\tau_i$の最悪実行時間$C_i$と周期$T_i$の組$\tau_i = (C_i, T_i)$で定義される．また，$U_i = C_i/T_i$をタスク$\tau_i$の利用率とし，システム利用率を$U(\tau)/\mathrm{M}$ と定義する．各タスクτ_iはジョブを周期T_iの間隔で生成し，k 番目に生成されるジョブをτ_{ik}と表す．τ_{ik}は時刻r_{ik}でリリースされ，デッドラインd_{ik}は次のジョブのリリース時刻と一致する．ある時刻tにおけるジョブτ_{ik}の残り実行時間を$C_{ik}(t)$とするとき，τ_{ik}の余裕時間 $\mathrm{L}_{ik}(t)$を $\mathrm{L}_{ik}(t) = d_{ik} - t - C_{ik}(t)$と定義する．

3. 従来研究

3.1 EDF(Earliest Deadline First)

EDF はデッドラインの早いタスクから順に実行していく動的優先度方式のスケジューリングアルゴリズムである．シングルプロセッサ環境下において最適なスケジューリング方法とされているが，マルチプロセッサ環境下においてはスケジュール成功率が極端に低いという問題点をもつ．スケジューラの起動はタスクの起動時と完了時のみ行われ優先度を決定している．図 1 に EDF でのスケジュール例を示す．今回実行に用いたタスクセットは $\tau = \{\tau_1 = (2, 4), \tau_2 = (3, 4), \tau_3 = (5, 8)\}$であり， M = 2 である．

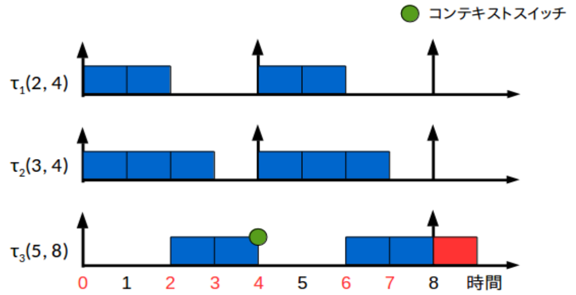

図 1 EDF でのスケジュール例

3.2 EDZL(Earliest Deadline Zero Laxity)[4]

EDZL は EDF において余裕時間が 0 となったタスクから最高優先度を与えるアルゴリズムである．余裕時間が負になる前に優先して実行することでデッドラインミスを防ぐ．スケジューラの起動は毎単位時間行う．図 2 に EDF と同様のタスクセットでの EDZL のスケジュール例を示す．

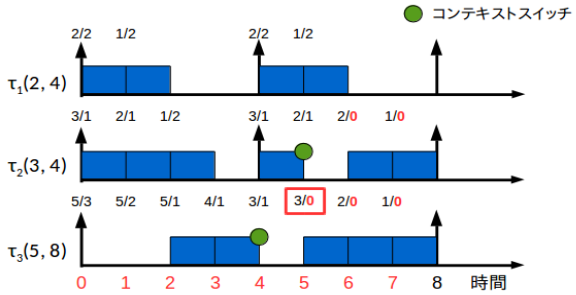

図 2 EDZL でのスケジュール例

図 1 ではデッドラインミスを起こしていたが，図 2 ではスケジュールに成功している．このことから分かるように，EDZL は EDF に比べ，スケジュール成功率が大幅に向上した．しかしながら，余裕時間を計算するためにスケジューラの起動を毎単位時間行う必要があり，オーバーヘッドが大きく実装の面で問題を抱えている．

3.3 EDCL(Earliest Deadline Critical Laxity)

EDCL は EDF にクリティカルタスクルールを取り入れたアルゴリズムである．クリティカルタスクルールとは，EDF スケジューリングを仮定した際に実行されるタスクの最小の残り実行時間よりも余裕時間が小さいタスクに最高優先度を与えるルールである．スケジューラの起動はタスクの起動時と完了時のみである．図 3 に同様のタスクセットにおける EDCL でのスケジュール例を示す．

† 東京都市大学 Tokyo City University
‡ 東京都市大学 Tokyo City University
†† 東京都市大学 Tokyo City University

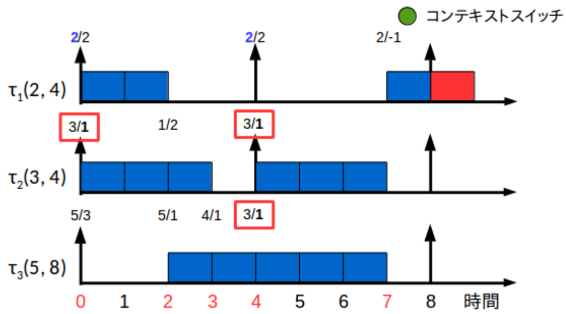

図 3 EDCL でのスケジュール例

EDCL はスケジューラの起動をタスクの起動時と完了時のみに制限したため，スケジューラの起動回数を EDZL の 8 単位時間から 5 単位時間と削減し，オーバーヘッドを抑えることができている．しかし，デッドラインミスを起こしており，EDZL よりもスケジュール成功率が低いという問題点を抱えている．

4. 分析モデル

3.3 における EDCL の問題点は，最高優先度を獲得する条件であるクリティカルの基準を変更することにより改善が期待できる．EDCL におけるクリティカルの基準を変更した 6 つのパターンを以下に示す．ここで，任意のスケジューラの起動時刻t_sにおいて，EDF でスケジュールする際に実行されるタスクにおける最小の残り実行時間を$e_{min}(t_s)$とし，タスクτ_iの残り実行時間を$C_i(t_s)$，余裕時間を$L_i(t_s)$とする．今回は，$e_{min}(t_s) = 2$，$C_i(t_s) = 4$の場合を想定する．

4.1 パターン 1

パターン 1 は元の EDCL である．タスクτ_iは以下の式(1)を満たすとき(図 4)に最高優先度を獲得する．

$$e_{min}(t_s) > L_i(t_s) \qquad (1)$$

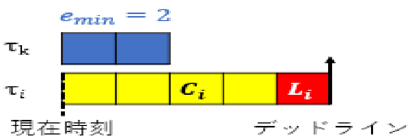

図 4 最高優先度のタイミング(パターン 1)

タスクτ_iは図 4 における現在時刻で最高優先度を獲得する．パターン 1 は$e_{min}(t_s)$の値にもよるが，周期の長いタスクが高負荷でデッドラインミスを起こしやすいという特徴がある．

4.2 パターン 2

パターン 2 は次式(2)を満たすとき(図 5)，タスクτ_iは最高優先度を獲得する．

$$\lceil e_{min}(t_s) \times 1/2 \rceil > L_i(t_s) \qquad (2)$$

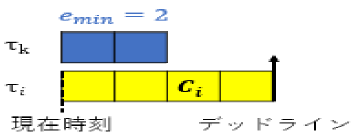

図 5 最高優先度のタイミング(パターン 2)

図 4 と比べ，最高優先度を獲得する時刻がデッドラインに近い．これは，$e_{min}(t_s)$に$1/2$を掛けることで，クリティカルになりにくく設定したためである．

4.3 パターン 3

パターン 3 は次式(3)を満たすとき(図 6)，タスクτ_iは最高優先度を獲得する．

$$\lceil e_{min}(t_s) \times 3/2 \rceil > L_i(t_s) \qquad (3)$$

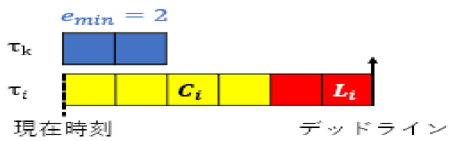

図 6 最高優先度のタイミング(パターン 3)

図 4 と比べ，デッドラインから余裕を持って最高優先度を獲得している．これは，パターン 2 とは反対に，$e_{min}(t_s)$に$3/2$を掛けることで，クリティカルになりやすく設定したためである．

4.4 パターン 4

パターン 4 以降は$e_{min}(t_s)$を用いるのではなく，タスクτ_iの残り実行時間$C_i(t_s)$と余裕時間$L_i(t_s)$とを比較する．そのため，自タスク内の比較のみで最高優先度を決定できる．

パターン 4 は次式(4)を満たすとき(図 7)，タスクτ_iは最高優先度を獲得する．

$$\lceil C_i(t_s) \times 1/2 \rceil > L_i(t_s) \qquad (4)$$

図 7 最高優先度のタイミング(パターン 4)

パターン 4 は自タスク内で比較するため、周期の長いタスクが高負荷でデッドラインミスを起こしやすいパターン 1 とは異なり，そのようなタスクでも比較的最高優先度を獲得しやすいという特徴をもつ．

4.5 パターン 5

パターン 5 は次式(5)を満たすとき(図 8)，タスクτ_iは最高優先度を獲得する．

$$\lceil C_i(t_s) \times 1/4 \rceil > L_i(t_s) \qquad (5)$$

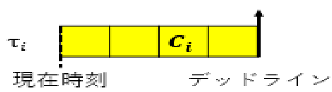

図 8 最高優先度のタイミング(パターン 5)

図 7 と比べ，最高優先度を獲得する時刻がデッドラインに近い．これは，パターン 4 では$C_i(t_s)$に$1/2$を掛けていたのに対し，$1/4$を掛けることでクリティカルになりにくく設定したためである．

4.6 パターン 6

パターン 6 は次式(6)を満たすとき(図 9)，タスクτ_iは最高優先度を獲得する．

$$\lceil C_i(t_s) \times 3/4 \rceil > L_i(t_s) \qquad (6)$$

図 9 最高優先度のタイミング(パターン 6)

図 7 と比べ，デッドラインから余裕を持って最高優先度を獲得している．これは，パターン 5 とは反対に，$C_i(t_s)$に3/4を掛けることでクリティカルになりやすく設定したためである．

5. スケジュール可能性解析

本研究では，反応時間解析(Response-time Analysis：RTA)[5]を用いてスケジュール可能性解析を行う．

5.1 用語の定義と補題

まず，解析に使用する用語を 2 つ定義する．

定義 1 (干渉長)． 区間$[a,b)$におけるタスクτ_kへの干渉長$I_k[a,b)$は，τ_kがM個以上の高優先度タスクによってブロックされ実行できない区間の合計長を表す．また，この区間におけるタスクτ_kへのτ_iによる干渉長$I_k^i[a,b)$は，τ_kがτ_iによりブロックされて実行できない区間の合計長を表す．

定義 2 (仕事量)． 区間$[a,b)$におけるタスクτ_iの仕事量$W_i[a,b)$は与えられたスケジュールにおいてτ_iがこの区間で実行しなければならない実行量を表す．

次に，干渉長に関する補題を以下に示す．

補題 1. すべてのグローバルスケジューリングアルゴリズムに対して以下の式(7) が成り立つ[6].

$$I_k[a,b) \geq x \Leftrightarrow \sum_{i \neq k} \min\left(I_k^i[a,b), x\right) \geq Mx \quad (7)$$

解析の手順を以下に示す．各タスクτ_kの反応時間が最大となる状況を考える．そのときのジョブをJ_k^*とし，最大反応時間をR_k^{ub}とする．ジョブJ_k^*について，J_k^*のリリース時刻r_k^*から応答までの区間$[r_k^*, r_k^* + R_k^{ub})$における$\tau_k$への干渉長$I_k$の上限値$I_k^{ub}$を求める．そして，$I_k^{ub}$と$\tau_k$の実行時間から$\tau_k$の反応時間の上限値$R_k^{ub}$を求める．$R_k^{ub}$を用いることで，$\tau k$の余裕時間の下限値$L_k^{lb} = T_k - R_k^{ub}$を求めることができる．余裕時間の下限値$L_k^{lb}$からスケジュール可能性判定条件を導く．

ここで，干渉長の上限値I_k^{ub}に関する補題を以下に示す．

補題 2. 区間$[a,b)$におけるタスクτ_iのτ_kへの干渉長$I_k^i[a,b)$は，この区間でのτ_iの仕事量$W_i[a,b)$を超えない．

ゆえに，干渉するタスクτ_iの仕事量の上限値W_i^{ub}を求めることにより干渉長の上限値I_k^{ub}が得られる．

また，マルチプロセッサシステムでτ_iがデッドラインミスを起こさない場合の最悪の仕事量の上限値は，式(8)と(9)を用いることで計算できる．

$$N_i\left(R_k^{ub}\right) = \left\lfloor \frac{R_k^{ub} + T_i - C_i}{T_i} \right\rfloor \quad (8)$$

$$\omega_i\left(R_k^{ub}\right) = N_i\left(R_k^{ub}\right)C_i + \min\left(C_i, R_k^{ub} + T_i - C_i - N_i\left(R_k^{ub}\right)T_i\right) \quad (9)$$

5.2 パターン 1~6 のスケジュール可能性解析

パターン 1~6 はクリティカルタスクがあるかどうかで仕事量の最大値が異なる．ない場合は常に EDF で動作するため，同じ解析結果になる[6]．ある場合，τ_kよりもデッドラインの遅いタスクτ_iがクリティカルタスクとなり，τ_kが実行終了したと同時にτ_iも実行時間C_i分実行完了したときが仕事量の最大値となる．具体的な状況を図 10 に示す．

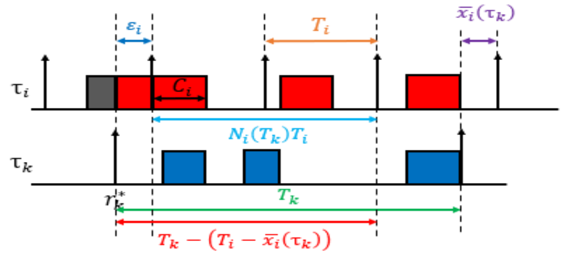

図 10 仕事量が最大となる状況

図 10 より，区間$[r_k^*, r_k^* + T_k)$におけるτ_iの仕事量は，最悪実行時間すべて実行できているジョブの実行時間分とリリース 時刻が区間より前でデッドラインが区間の中にあるジョ ブ(carry-in ジョブ)の実行時間分の和で求められる．ここで，$N_i(T_k)$をリリース時刻とデッドラインが共に区間内に収まっているジョブの数とすると式(10)で表される．

$$N_i(T_k) = \left\lfloor \frac{T_k - \left(T_i - \overline{x_i}(\tau_k)\right)}{T_i} \right\rfloor \quad (10)$$

また，τ_iがクリティカルになるときの余裕時間$\overline{x_i}(\tau_k)$は式(11)で与えられる．

$$\overline{x_i}(\tau_k) = \min\left(T_i - C_i,\ T_k - C_i, ※\right) \quad (11)$$

ただし，※はパターンにより下記の通り変化する．

パターン 1： $\min(C_* | \tau_* \in \tau,\ * \neq i)$
パターン 2： $\min(\lceil 1/2 \cdot C_* \rceil | \tau_* \in \tau,\ * \neq i)$
パターン 3： $\min(\lceil 3/2 \cdot C_* \rceil | \tau_* \in \tau,\ * \neq i)$
パターン 4： $\lceil 1/2 \cdot C_i \rceil$
パターン 5： $\lceil 1/4 \cdot C_i \rceil$
パターン 6： $\lceil 3/4 \cdot C_i \rceil$

さらに，carry-in ジョブの仕事量ε_iは式(12)で計算できる．

$$\varepsilon_i = \min\left(C_i,\ \max\left(0, T_k - N_i(T_k)T_i - \left(T_i - \overline{x_i}(\tau_k)\right)\right)\right) \quad (12)$$

よって，τ_iの仕事量の最大値$W_i^{ub}(T_k)$は次式(13)で表される．

$$W_i^{ub}(T_k) = min\left(T_k,\ (N_i(T_k) + 1)C_i + min\left(C_i,\ max\left(0, T_k - N_i(T_k)T_i - \left(T_i - \overline{x_i}(\tau_k)\right)\right)\right)\right) \quad (13)$$

以上より，それぞれのパターンにおける反応時間の上限値が求められる．

定理 1. パターン 1~6 でスケジュールされたマルチプロセッサシステムのタスクτ_kの反応時間の上限値は，次の式(14)のR_k^{ub}に関して$R_k^{ub} = C_k$から始まる不動点反復法を解くことによって求められる[5].

$$R_k^{ub} \leftarrow C_k + \left\lfloor \frac{1}{M} \sum_{i \neq k} \hat{I}_k^i\left(R_k^{ub}\right) \right\rfloor \quad (14)$$

ただし$\hat{I}_k^i\left(R_k^{ub}\right) \min\left(\omega_i^{ub}\left(R_k^{ub}\right), W_i^{ub}(T_k), R_k^{ub} - C_k + 1\right)$であり，$W_i^{ub}(T_k)$は式(14)の値である．

各パターンにおける反応時間の上限値が求められることで，スケジュール可能性判定が得られる．

定理 2. タスクセット$\tau = \{\tau_1, \tau_2, \ldots, \tau_N\}$は，すべてのタスクが以下の式(15)を満たせば各パターン 1~6 によりスケジュール可能である．ただし，R_k^{ub}は定理 1 で求められる値である．

$$L_k^{lb} = T_k - R_k^{ub} \geq 0 \quad (15)$$

6. 評価

評価はシミュレーションにより行う．システム利用率が 70%から 100%(スケジュール可能性判定のみ 0%から 100%)まで 5%間隔で無作為に生成したタスクセットを 1000 個投入し，各パターンでシミュレーションを行う．プロセッサ数は$M = 4, 8, 16$ の 3 つで実験し，各タスク利用率の範囲は[0.01, 1.0]とする．比較する項目は次の 4 つとする．スケジュール成功率(スケジュールできたタスクセット数/1000)，平均コンテキストスイッチ回数(コンテキストスイッチ回数の合計/1000)，平均スケジューラ起動回数(スケジューラ起動回数の合計/1000)，スケジュール可能性解析での成功率(スケジュール可能であると判定されたタスクセット数/1000).

本論文では，一番差が顕著に表れたプロセッサ数が 16 のときのみグラフに示す．また，すべてのパターンでスケジューラの起動はタスクの起動時と完了時のみであり，回数にほとんど変化が見られなかったため，平均スケジューラ起動回数のグラフについては省略する．

図 11 に各パターンにおけるスケジュール成功率を示す．

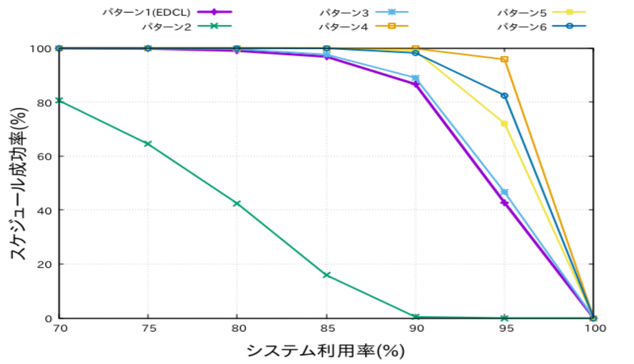

図 11 スケジュール成功率(M=16)

余裕時間$L_i(t_s)$を EDF スケジューリングで実行されるタスクにおける最小残り実行時間$e_{min}(t_s)$と比較するパターン 1~3 よりも自タスク内の残り実行時間$C_i(t_s)$と比較するパターン 4~6 の方がスケジュール成功率は高くなっている．高負荷においては，元の EDCL であるパターン 1 よりもパターン 4 では 2 倍程度スケジュール成功率が向上しており，高いスケジュール成功率をキープしている．また，スケジュール成功率の向上に伴い，デッドラインオーバー量の総量についても 1/2 倍程度に抑えることができた．

図 12 に各パターンでの平均コンテキスト回数を示す．

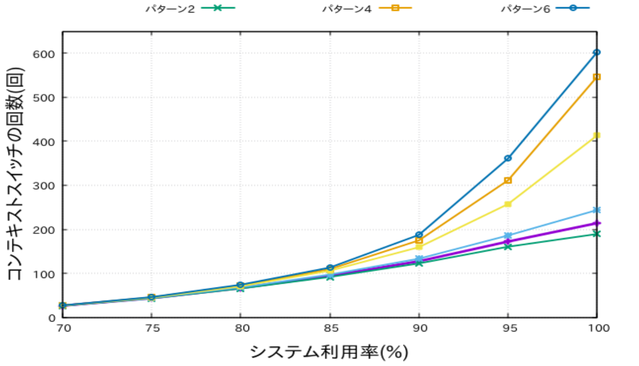

図 12 平均コンテキストスイッチ回数(M=16)

スケジュール成功率の場合とは反対に，パターン 1~3 の方がパターン 4~6 よりもコンテキストスイッチの回数は抑えられている．特に高負荷では，パターン 1~3 はコンテキストスイッチの回数を 1/2 倍程度に抑えられている．

図 13 に各パターンでのスケジュール可能性判定を示す．

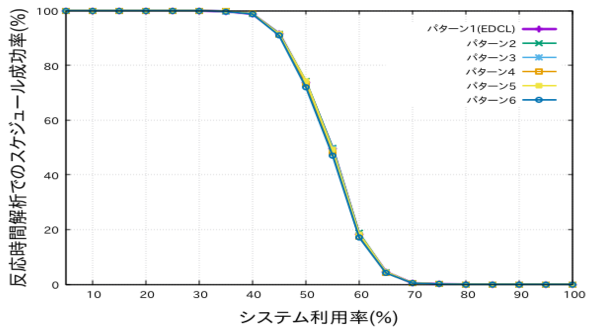

図 13 スケジュール可能性解析での成功率(M=16)

今回スケジュール可能性解析の手法として用いた RTA では，どのシステム利用率を見てもパターンによって結果に大きな変化は見られなかった．

以上より，パターン 4 がスケジュール成功率の面では最もクリティカルの基準に適している．しかし，パターン 4 はコンテキストスイッチ回数の面で問題がある．

7. 結論

本研究では，EDCL におけるクリティカルの基準を 6 つのパターンに変更し，それぞれの性能を比較・分析した．シミュレーションの結果，元のパターン 1 よりもパターン 4 はスケジューラの起動回数を抑えつつも大幅にスケジュール成功率を向上させ，かつデッドラインオーバー量を減少させていることを示した．しかし，パターン 4 のコンテキストスイッチの回数は大幅に増加している．また，シミュレーションによるスケジュール成功率の結果と可能性解析によるスケジュール成功率の結果に大きな差が生じているため，これらを今後の課題とする．

謝辞

本研究は JSPS 科研費 20K11755 の助成を受けたものです．

参考文献

[1] C. L. Liu, J. W. Layland, “Scheduling algorithms for multiprogram-ing in a Hard-Real-Time Environment“, Journal of the ACM, Vol. 20, No. 1, pp. 46-67, (1973).
[2] 武田 瑛, 船岡 健司, 加藤 真平, 山崎信行, “マルチプロセッサにおけるグローバル RM に基づくリアルタイムスケジューリングアルゴリズム”, 電子情報通信学会技術 研究報告, CPSY, 107 巻, 558 号, pp. 191-196, (2008).
[3] S. Kato, N. Yamazaki, “Global EDF-based scheduling with laxity-driven priority promotion”, Journal of systems Architecture, Vol. 57, No. 5, pp. 498-517, (2011).
[4] S. Cho, S. Lee, A. Han, K. Lin, “Efficient real-time scheduling algorithms for multiprocessor systems”, IEICE Transactions on Communications , Vol. E85-B, No. 12, pp. 2859-2867, (2002).
[5] M. Bertogna, M. Cirinei, “Response-Time Analysis for globally scheduled Symmetric Multiprocessor Platforms”, 28th IEEE International Real-Time Systems Symposium, pp. 149-160, (2007).
[6] M. Bertogna, M. Cirinei, G. Lipari, ”Improved schedulability analysis of EDF on multiprocessor platforms”, Proceedings of the 17th EuroMicro Conference on Real-Time Systems, pp. 209-218 (2005).