マルチプロセッサ同期プロトコルにおける排他制御のためのロック待ちキューイング優先度の割り当て手法

マ チュウキ[1] 倉地 亮[1] 曾 剛[2] 高田 広章[1]

アブストラクト マルチプロセッサに適用されるパーティション固定優先度スケジューリングには，よく使用されるプロトコルとして，MPCP と FMLP+の 2 つのプロトコルが存在する．前者では，優先度順のキューが適用されるため，高優先度タスクが過度の遅延を受けることはない．しかしながら，低優先度タスクがリソースを待っているとき，高優先度タスクに何回かプリエンプトされてしまう可能性がある．その一方で，後者ではFIFOキューが採用されている．ジョブは，リソースの要求ごとに他タスクのジョブによる遅延が多くとも 1 回あるが，高優先度タスクは必ずしもデッドラインを守ることができるとは限らない．このように，MPCP でも FMLP+でもそれぞれメリットもデメリットもある．本研究では，2 つのプロトコルの比較に基づき，「理論余裕時間」という新しいパラメータを提案する．より具体的には，ロックを取得したジョブは，理論余裕時間で並び，スケジュールされる．本提案手法の有効性と実現可能性を示すため，数値例を挙げる．

Priority Assignment of Wait-Lock Queuing for Mutual Exclusion in Multiprocessor Synchronization Protocol

Zhongqi MA[1] Ryo KURACHI[1] Gang ZENG[2] Hiroaki TAKADA[1]
1 Graduate School of Information Science, Nagoya University
2 Graduate School of Engineering, Nagoya University

Abstract There are two commonly used protocols for partitioned fixed priority scheduling: MPCP and FMLP+. Under the former, the priority queues are used so that higher-priority tasks cannot suffer from excessive delays. However, lower-priority tasks may be preempted by higher ones several times when they wait for a resource. By contrast, the FIFO queues are adopted under the latter. Although a job of task can be directly delay by others at most once per request for a resource, higher-priority tasks may miss their deadlines due to the expense of increased delays. Based on the analysis of two protocols' advantages and disadvantages, a new parameter, named "theoretical slack time", was proposed to order conflicting requests. Lock holders are scheduled in order of increasing theoretical slack time. A numerical example was employed to show the effectiveness and feasibility of our strategy.

1. はじめに

今日までに，リアルタイム・オペレーティングシステム（以下，RTOS）はさまざまな分野に欠かせないものとなっている．リアルタイム組込みシステムは多くのアプリケーションに必要とされる．例えば，工業オートメーションや車載ネットワークやロボティクスや軍事システムだけでなく，特にスマート玩具にも入っている[1, 2].

マルチプロセッサ RTOS において，予測可能なスケジューリングと排他制御は2つの重要な課題である[3]. 前者に対しては，パーティション固定優先度スケジューリングがよく利用されている．各タスクは静的に各プロセッサに割り当てられれば，シングルプロセッサシステムに基づくスケジューリング理論と解析手法が適用できることが，主な利点である．

排他制御に関しては，ミューテックスがよく使用されている．そのうえで，ロッキング・プロトコルの設計時には2つ課題がある．1つ目はブロックされたタスクの順番とクリティカルセクションを実行するタイミングである．前者に対しては，First In First Out（以下，FIFO）キューと

1 名古屋大学大学院情報科学研究科
2 名古屋大学大学院工学研究科

優先度順が主に選択されている．FIFO キューの方が比較的簡単ではあるが，高優先度タスクは遅延が長くなる．優先度順は RTOS にとって適しているが，低優先度タスクはブロッキング時間が増加する．後者に関しては，ジョブが同期プロセッサでクリティカルセクションを実行する分散プロトコルでもあるため，各タスクが割り当てられたプロセッサで実行する共有メモリプロトコルが必要となる．このため，本論文では，同期プロセッサを用いないシステムを仮定する．

Multiprocessor Priority Ceiling Protocol（以下，MPCP）は 1990 年に Rajkumar によって提案され，優先度順に基づく共有メモリプロトコルである．タスクがローカルリソースにアクセスする場合，Priority Ceiling Protocol は適用できる．グローバルクリティカルセクションを実行しているジョブにはグローバルな優先度が与えられる[4].

一方で，FIFO Multiprocessor Locking Protocol（以下，FMLP$^+$）は FIFO キュー付きの共有メモリプロトコルであり，2011 年に Brandenburg によって提案された[5]. FMLP$^+$では，ロックを取得したジョブが，ロック要求時刻で並び，スケジュールされる．つまり，ロックを取得しているジョブは後からロックを要求するジョブによって遅延されることはない．

本論文では，パーティション固定優先度スケジューリングをベースに，新しいパラメータである理論余裕時間を採用した同期プロトコルを提案する．構成を以下に示す．2 章では本論文で扱うシステムのモデル化を行い，3 章では既存のマルチプロセッサ向けリアルタイム同期プロトコルを比較する．4 章では提案する同期プロトコルについて述べ，5 章では数値例を挙げ，6 章ではまとめと今後の予定に関して述べる．

2. システムモデル

2.1. タスクセット

本論文では，m 個のプロセッサ$\{P_1, P_2, \dots P_m,\}$と n 個の周期タスク$\{T_1, T_2, \dots T_n,\}$から構成されるタスクセットを対象としており，タスク T_iのデッドライン時刻 d_iや周期 p_iやクリティカルセクションを含む最悪実行時間 e_i が定義され，d_iと p_i は等しいものとする．T_i の優先度は i とし，値の小さいもの程高優先度である．$P(T_i)$は T_i の割り当てられたプロセッサを示す．

2.2. リソース

タスク間には共有リソースが u_r 個あり，$\{l_1, l_2, \dots l_q, \dots l_{n_r},\}$で表記してある．リソースの要求回数を $N_{i,q}$と定義し，リソース l_qにアクセスする最大クリティカルセクションの長さを $L_{i,q}$と定義する．また，本論文では，リソースのネストは扱わないものとする．

リソースはローカルリソースとグローバルリソースの 2 種類がある．リソースにアクセスするタスクがすべて同じプロセッサに割り当てられたら，そのリソースはローカルリソースと呼ぶ．一方で，異なるプロセッサ上のタスクが使用するならば，グローバルリソースと呼ぶ．なるべく早くグローバルリソースを解放し，リモートプロセッサ（4 章，P_r）にあるタスクを邪魔しないよう，グローバルリソースのロックを取得したジョブは，グローバルな優先度まで優先度が上がり，クリティカルセクションが実行できる．

ただし，ローカルリソースにアクセスする場合は，優先度上げの規則がプロトコルによって異なる．MPCP の場合は，ジョブがローカルリソースをロックした時刻に，優先度は上らない．高優先度タスク（4 章，T_h）もそのローカルリソースを要求したとき，ジョブは優先度が一時的にローカルリソースの優先度上限まで上げられ，高優先度タスクをブロックする．したがって，ローカルリソースをロックしようとする他タスクがスケジュールされることがなくなる[4]. FMLP$^+$の場合は，グローバルリソースと同じように，ローカルリソースをロックしたジョブの優先度がすぐグローバルな優先度まで上げられる[5].

3. プロトコルの比較

3.1. リソース要求間の競合

ローカルプロセッサ（4 章，P_l）で複数のジョブが順番に同じまたは違うリソースを要求する場合や，違うプロセッサで複数のジョブが同じリソースを要求する場合，リソース要求間の競合は起こる．要求の順番はプロトコルによって異なる．

3.1.1. MPCP

各リソースには優先度上限が割り当てられており，それはそのリソースをロックしたタスクの持ちうる最高の優先度である．複数のジョブが違うリソースを要求する場合，要求は優先度上限で並ぶ．一方，同じリソースの場合，元の優先度で並ぶ．[3]に記載された制約条件 15 のとおり，あるタスクはリソースの要求ごとに多くとも 1 個の低優先度タスク（4 章，T_l）に多くとも 1 回直接に遅延される．そういう利点があるにもかかわらず，高優先度タスクに複数回遅延される可能性がある．

図 1 に，MPCP の場合，T_1, T_2, T_3, T_4, T_5 は別々に P_1, P_2, P_3, P_4, P_5 に割り当てられ，l_1 を共有している例を示す．T_5 は時刻 1 に l_1 を要求し，ロックを取得する．時刻 2, 3, 4, 5 において，T_4, T_2, T_1, T_3 が順番に l_1 を要求するが，ロックを取得できないため，待ちキューに

並び，ロックが渡されるのを待つ．続いて，時刻 6 に T_5 がロックを T_1 へ渡し，通常実行し続ける．その後，時刻 7，8，9 において，優先度が高い T_1 と T_2 が互いにロックを渡す．結局，T_3 は(多くとも 1 個の)優先度が低い T_5 に 1 回ブロックされた上，優先度が高い T_1 と T_2 に 2 回邪魔される．

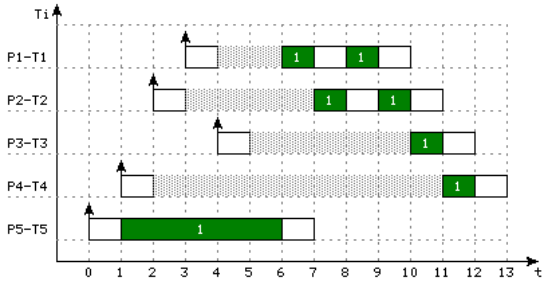

図 1　MPCP でスケジューリングの例(非最悪場合)

3.1.2. FMLP$^+$

プロトコル名のとおり，複数の要求は FIFO キューで並ぶ．[3]に書いた制約条件 12 からも明らかなように，タスクはリソースの要求ごとに他タスクに多くとも 1 回直接に遅延される．しかしながら，要求ごとに複数の低優先度タスクに遅延される可能性があるという欠点がある．

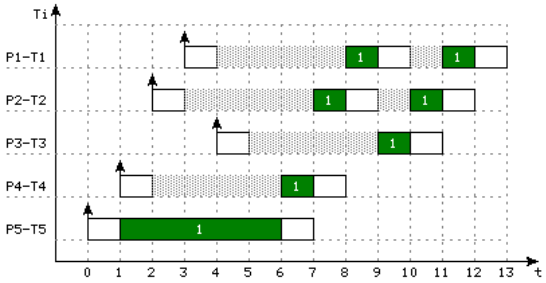

図 2　FMLP$^+$でスケジューリングの例(非最悪場合)

図 2 に，図 1 と同じタスクセットに対し，FMLP$^+$を採用した場合の例を示す．図 2 同様，T_5，T_4，T_2，T_1，T_3 が順番に l_1 を要求する．時刻 6 に T_5 がロックを解放した後，T_4，T_2，T_1，T_3 が l_1 の要求順番でロックを取得する．その結果，T_3 は他タスクに(多くとも)1 回しか遅延されないが，優先度が低い T_4 にも T_5 にも 1 回ずつブロックされる．

3.2. 共通点

表 1 に，各方式において，何個の他タスクに何回遅延されるかをまとめた表を示す．

表 1　プロトコルの相違点と共通点

個数／回数	高優先度タスク	低優先度タスク
MPCP	**複数**／数回	**1 個／1 回**
FMLP$^+$	**複数／1 回**	複数／**1 回**

2 つのプロトコルは共通点が，複数の高優先度タスクによる 1 回の遅延および 1 個の低優先度タスクによる 1 回の遅延である．以下の式で表す:

$$\sum_{h<i} L_{h,q} + \max_{l>i} L_{l,q}$$

前例の T_3 の最悪応答時間を考えると，優先度が高い T_1 と T_2 による 1 回の遅延および優先度が低い T_4 か T_5 による 1 回の遅延は，どのプロトコルでも避けられない．すなわち，

$$L_{1,1} + L_{2,1} + \max(L_{4,1}, L_{5,1}).$$

4. 提案手法

本章では，提案するパラメータについて述べ，パラメータの計算式の導出を可能にする．

4.1. 新しいパラメータ

最悪応答時間の計算については，プロトコルの共通点による遅延時間が避けられない．それによって，新しいパラメータ「避けられない応答時間（Inevitable Response Time）」を提案し，λ_i で表記する．

リソース要求間の競合をよく対応できるように，避けられない応答時間を検討すべきである．既存のパラメータである slack time または laxity$(= d_i - e_i)$ではなく，パラメータをもう 1 つ提案する．「理論余裕時間（Theoretical Slack Time）」と呼び，Δ_i で表される．T_i の理論余裕時間は下記のとおりである:

$$\Delta_i = d_i - \lambda_i$$

提案手法は，複数の要求が理論余裕時間で並ぶことである．理論余裕時間の短いタスクはロック待ちキューイング優先度が高い．

4.2. λ_i 計算式の導出

避けられない応答時間はもちろん最悪実行時間を含み，残りの部分は 4.2.1~4.2.4 四つの要素からなる．

4.2.1. P_r に割り当てられた T_h

P_r に割り当てられた T_h による邪魔は，グローバルクリティカルセクションで起こることが予想される．表 1 のとおり，複数の T_h に 1 回遅延される時間は避けられない．この時間を$\lambda_i^{r,h}$で表記する．

$$\lambda_i^{r,h} = \sum_{q=1}^{n_r} \sum_{P(T_h)\neq P(T_i) \wedge h<i \wedge N_{i,q}>0 \wedge N_{h,q}>0} L_{h,q}$$

4.2.2. P_r に割り当てられた T_l

P_r に割り当てられた低優先度タスクからの遅延は，T_h と同じように，グローバルクリティカルセクションで起こるが，数個ではなく，1 個しか起こらないので，最長クリティカルセクションを考慮すべきである．$\lambda_i^{r,l}$で表す．

$$\lambda_i^{r,l} = \sum_{q=1}^{n_r} \max_{P(T_l)\neq P(T_i)\wedge l>i\wedge N_{i,q}>0\wedge N_{h,q}>0} L_{l,q}$$

4.2.3. P_lに割り当てられた T_l

2 章で述べたとおり，MPCP の場合，あるタスクが必ずしも低優先度タスクのローカルクリティカルセクションに邪魔されるとは限らない．あらゆる低優先度タスクの最長グローバルクリティカルセクションで$\lambda_i^{l,l}$を導出する．

$$\lambda_i^{l,l} = \sum_{P(T_l)=P(T_i)\wedge l>i} \sum_{q_g=1}^{n_r} L_{l,q_g}$$

ここで，q_gはグローバルリソースを表す．

4.2.4. P_lに割り当てられた T_h

P_l に割り当てられた T_h である以上，クリティカルセクションだけでなく，通常実行も影響を何回か与える．したがって，繰り返して計算せざるを得ない．$\lambda_i^{l,h}$は単独では導出できない．

4.2.5. 要求回数

タスクはリソースを数回要求すれば，他タスクに数回遅延される可能性がある．これは，$\lambda_i^{r,h}$，$\lambda_i^{r,l}$，$\lambda_i^{l,l}$に $N_{i,q}$ をかけ導出できる．λ_i は下式で表すことができる．

$$\begin{cases} \lambda_i^{(0)} = e_i + \left(\lambda_i^{r,h} + \lambda_i^{r,l} + \lambda_i^{l,l}\right) \cdot N_{i,q} \\ \lambda_i^{(s)} = \lambda_i^{(0)} + \sum_{P(T_h)=P(T_i)\wedge h<i} \left\lceil \frac{\lambda_i^{(s-1)}}{d_h} \right\rceil \cdot e_h \end{cases}$$

5. 数値例

既存プロトコルとの比較のため，数値計算を行う．

表 2　4 つのタスクからなるタスクセット

T_i	$P(T_i)$	e_i	d_i	$L_{i,q}$	$N_{i,q}$
T_1	P_1	3	7	1	1
T_2	P_2	5	8	1	1
T_3	P_3	3	10	1	1
T_4	P_4	5	11	3	1

Brandenburg が提案した線形計画法[3]を用いて，表 2 にリストした各タスクの最大ブロッキング時間と最悪応答時間を計算し，表 3 で示す．

表 3　MPCP と FMLP$^+$から得られる最悪応答時間

T_i	b_i^M	r_i^M	b_i^F	r_i^F	λ_i	Δ_i	b_i	r_i
T_1	3	6	5	8	6	1	4	7
T_2	4	9	5	10	9	-1	3	8
T_3	5	8	5	8	8	2	5	8
T_4	3	8	3	8	8	2	3	8

ここで，b_i^M，b_i^F，b_i は別々に MPCP，FMLP$^+$，提案手法で得られた最大ブロッキング時間であり，r_i^M，r_i^F，r_i は対応する最悪応答時間である．それらの結果から見ると，FMLP$^+$の場合は，T_1 と T_2 がデッドラインを守らず，MPCP の場合も，T_2 がデッドラインをミスする．結果，全タスクセットもスケジュールできない．

本手法では，Δ_2 が最も小さいため，T_2 のキューイング優先度が一番高くなるように改善する．リソース要求間の競合発生時，T_2 は T_1 をプリエンプトし，先に l_1 をロックする．T_2 は応答性がよくなり，デッドラインを守り，さらに全タスクセットもスケジュールできると言える．

6. おわりに

本論文では，マルチプロセッサ RTOS におけるパーティション固定優先度スケジューリングを対象とし，タスクセットのスケジューラビリティを改良する同期プロトコルを提案した．既存のプロトコルを比較することで，リソース要求間の競合発生時キューイング優先度の割り当て方の改善を目的とし，新しいパラメータの計算式を導出した．そして最後に，数値計算を行い，本手法のメリットと可用性を示した．

今後の予定として，さらに複雑な例を挙げ，それらの例に対応するスケジューリング図を作るのである．理論的に解析するについては，避けられない応答時間の計算式を再検討し，提案手法に関する制約条件をまとめ，優れた線形計画法で最大ブロッキング時間を求める．一方で，ランダムにタスクセットを生成し，シミュレーションで評価する予定である．

参考文献

[1] Davis, R. I., Burns, A., A survey of hard real-time scheduling for multiprocessor systems, ACM computing surveys, 43(4), 35, 2011

[2] Buttazzo, G., Hard real-time computing systems: predictable scheduling algorithms and applications, Springer Science & Business Media.

[3] Brandenburg, B. B., Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling, IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), 141-152, 2013

[4] Rajkumar, R., Sha, L., Lehoczky, J. P., Real-time synchronization protocols for multiprocessors, Real-Time Systems Symposium Proceedings, 259-269, 1988

[5] Brandenburg, B. B., Scheduling and locking in multiprocessor real-time operating systems, 2011