システムソフトウェアと オペレーティング・システム 74−27
マルチメディア通信と分散処理 81−27
（1997. 2. 28）

周期調整によるリアルタイムスレッドの動的スケジューリング

小杉 尚子 盛合 敏
nao@isl.ntt.co.jp *moriai@isl.ntt.co.jp*
NTT 情報通信研究所

本報告では、リアルタイムスレッドを周期調整して動的にスケジュールする方法について述べる。そして Real-Time Mach 3.0 上に CPU 資源管理用のサーバ「MART サーバ」を実装し評価を行ない、本手法の有効性を示す。プロトタイプシステムでは、MART サーバに CPU 資源の管理を要請したクライアントの CPU 資源を MART サーバが一括管理し、新しいスレッドの生成や起動周期の変化に応じて各スレッドに CPU 資源を再配分する。クライアントはサーバの指示に従って内部のスレッドの起動周期を変化させるのでシステム全体は常にスケジュール可能な状態に保たれる。

Dynamic Scheduling for Real-Time Threads by Period Adjustment

Naoko Kosugi Satoshi Moriai
NTT Information and Communication Systems Labs.

This paper describes a dynamic scheduling method for real-time threads by adjusting their periods. MART server, a server for CPU resource management, is implemented on Real-Time Mach 3.0 and evaluated to verify its effectiveness. In this prototype system, the server manages CPU resource of clients those request management of their CPU resource. It re-allocates CPU resource among threads when a new thread is created or a thread's period is modified. The system always keeps the schedulability of all threads because clients adjust threads' periods following the MART server's order.

1 はじめに

動画や音声など時間制約を持つデータを使用した実時間アプリケーションを処理する場合、タイムシェアリングのように公平性を重視した既存のスケジューリングアルゴリズムでは、各データの時間的な制約を満足させることができない。そのため Rate Monotonic [7] や Earliest Deadline First [4] といった、処理の時間制約を保証することを目的としたリアルタイムスケジューリングアルゴリズムが多数提案されてきた [9]。しかしこれらのリアルタイムスケジューリングアルゴリズムは柔軟性に乏しく、マルチメディアシステムのようなユーザからのインタラクションなどによってシステムの状態が変化するリアルタイムシステムには不向きである。そこで本報告では、システムの状態変化にも柔軟に対応できるリアルタイムシステムを構築するためのスケジューリングアルゴリズムを提案し実装・評価する。

本研究ではスケジューリング機構として CPU 資源管理サーバを実装する。サーバの基本アルゴリズムとして MART[1] アルゴリズム [1] を使用する。MART アルゴリズムは Rate Monotonic アルゴリズムを動的なリアルタイムシステム用に拡張したものである。

本報告は 6 節からなる。まず 2節で MART アルゴリズムについて、次に 3節で実装した CPU 資源管理システムについて説明する。その後 4節では 2 つの評価結果を報告し、5節で関連研究について述べる。最後に 6節で結論と今後の課題を述べる。

2 MART アルゴリズム

MART アルゴリズムは、スレッドの数やスレッドの時間的属性 (実行時間や周期など) を変化させることができるリアルタイムシステムを対象にしたスケジューリングアルゴリズムである。そして、システムが過負荷になりそうな場合は、複数の処理の品質・内容を調整することで常に全スレッドをスケジュール可能な状態に維持できる。本報告では周期の調整に限定してアルゴリズムを説明する。

2.1 Rate Monotonic アルゴリズム

Rate Monotonic アルゴリズムは周期スレッドを対象とした静的なアルゴリズムで、システム内の全スレッドが式 (1) を満たす時そのスケジュール可能

[1] Modification and Adjustment of Real-time Tasks

性を保証することが知られている。(n: スレッドの総数, C_j: スレッド j の実行時間, T_j: スレッド j の周期)

$$\forall i\ 1 \le i \le n \qquad \min_{(k,l)\in R_i} \sum_{j=1}^{i} \frac{C_j}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil \le 1.0 \qquad (1)$$

$$R_i = \{(k,l) \mid 1 \le k \le i,\ l = 1, \cdots, \lfloor T_i/T_k \rfloor\}$$

あるスレッドの実行時間をその周期で割ったものは「CPU 資源使用率」と定義され、本報告の「CPU 資源の割当量」に相当する。

2.2 アルゴリズムの概要

MART アルゴリズムは以下の 4 種の要求

- 新しいスレッドの生成
- スレッドの消滅
- スレッドの実行時間の変更
- スレッドの周期の変更

に対応することができるが、その結果として過負荷になると判断した場合の動作は要求が「周期変更」であったかそれ以外であったかによって異なる。これは MART アルゴリズムがいくつかのスレッドの周期を調整することでスケジュール可能性を維持していることに起因する。

周期変更が要求されなかった場合

式 (1) において、$i = I$ の時にスケジュール不可能と判断されたとする。このとき、以下の式 (2) が成り立っている [5] 。

$$L'_I > 1.0 \qquad (2)$$

ただし、

$L'_I = \min_{t\in R_I} L_I(t),\ L_I(t) = \sum_{j=1}^{I} C_j \lceil t/T_j \rceil / t$
$R_i = \bigcup_{j=1}^{i} Q_j^i\ (1 \le j \le i), \quad Q_i^i = \{T_i\}$
$Q_j^i = \{\lfloor t/T_j \rfloor T_j \mid t \in Q_k^i (j+1 \le k \le i)\}$

である。従って少なくとも 1 つの $L_I(t)$ が 1.0 以下になるようにいくつかのスレッドの周期を調整すれば、再びスケジュール可能な状態に戻すことができる。MART アルゴリズムでは計算量が不確定になってしまうのを避けるために、式 (1) において最小値を持つ k, l、すなわち L'_I を生成するスレッド k の周期 T_k を基準に各スレッドの周期を調整する。

具体的には $L_I(m_l^p T_p) = L'_I$ の場合、

$$L'_I = \sum_{j=1}^{I} C_j \lceil m_l^p T_p / T_j \rceil / m_l^p T_p \qquad (3)$$

である。スレッド p の新しい周期 T'_p は、

$$T'_p = \sum_{j=1}^{I} C_j \lceil m_l^p T_p / T_j \rceil / m_l^p \qquad (4)$$

残りの $(I-1)$ 個のスレッドの周期 T_j は式 (5)

$$\frac{m_l^p T'_p}{\lceil m_l^p T_p / T_j \rceil} \le T_j \qquad (1 \le j \le I\ ,\ j \ne p) \qquad (5)$$

の範囲内にあれば調整の必要はないが、そうでなければ式 (6) を用いて調整する。

$$T'_j = \left\lceil \frac{m_l^p T'_p}{\lceil m_l^p T_p / T_j \rceil} \right\rceil \qquad (1 \le j \le I\ ,\ j \ne p) \qquad (6)$$

式 (4)、式 (6) における周期調整で T'_I が T_j を越えてしまった場合、スレッド間の優先度が逆転してしまうので T_j も式 (7)

$$T'_j = T'_I \qquad (I < j \le n) \qquad (7)$$

を用いて調整する。

周期変更が要求された場合

スレッド k の周期を T_k から T'_k に変更する要求が受理されスケジュール不可能になると判断されたとする。スレッド k は新しい周期に基づいてソートされ、すでにスレッド集合内での順番が変化している。すなわちスレッド k の周期が変更される前は

$$1, \ldots, k'-1, k', k'+1, \ldots, k-1, k, k+1, \ldots, n$$

という順番だったスレッドの集合が、スレッド k の周期が変更された後ソートされて

$$1, \ldots, k'-1, k, k', k'+1, \ldots, k-1, k+1, \ldots, n$$

という順番になっているとする。

基本的には T'_k の整数倍の時点がスケジュール可能な時点になるように調整を行なう。まずスレッド k 以外のスレッドの実行を、少なくとも 1 回は保証するためにはスレッド k が実行されない時間間隔 $(T'_k - C_k)$ がどれだけ必要かを計算する。

$$m = \left\lceil \frac{\sum_{j=1, j\ne k}^{n} C_j}{T'_k - C_k} \right\rceil \qquad (8)$$

式 (8) はスレッド k を m 回と他のスレッドを 1 回実行するには、mT'_k 時間必要だということを意味している。この mT'_k を元に他のスレッドの周期を

$$L_n(mT'_k) \le 1.0$$

を満たすように調整する。

まず T'_k より周期が長いスレッドの周期 T_j を調整する。ここでの調整は非常に悲観的で、スレッド j が mT'_k 時間内に 1 回だけ実行されるように調整する。従って T_j が式 (9)

$$T_j > mT'_k \qquad (k' \le j \le n\ ,\ j \ne k) \qquad (9)$$

を満たしている場合は調整する必要はないが、そうでなければ以下のように調整する。

$$T'_j = mT'_k \quad (k' \leq j \leq n\,,\, j \neq k) \tag{10}$$

次に T'_k が最小周期ではない場合は T'_k より周期が短いスレッドの周期 $T_{j'}\,(1 \leq j' \leq k'-1)$ を調整する。ここではこれらのタスク間の大小関係をできるだけ維持できるように調整を試みる。

$$l = \left\lfloor \frac{m(T'_k - C_k) - \sum_{j=k',j\neq k}^{n} C_j}{\sum_{j=1}^{k'-1} \lfloor T_{k'-1}/T_j \rfloor C_j} \right\rfloor \tag{11}$$

$l = 0$ の場合は各スレッドの周期の大小関係を維持するように調整することはできないので、$T_{j'}$ を以下のように調整する。

$$T'_{j'} = mT'_k \quad (1 \leq j' \leq k'-1) \tag{12}$$

$l \neq 0$ の場合はスレッドの周期間のおおよその大小関係を l 個維持できるということを意味する。$T_{j'}$ が式 (13) を満足するならば調整の必要はない。

$$\left\lceil \frac{mT'_k}{l\lfloor T_{k'-1}/T_{j'} \rfloor} \right\rceil \leq T_{j'} \quad (1 \leq j' \leq k'-1) \tag{13}$$

$T_{j'}$ が式 (13) を満足しないとき、以下のように調整する。

$$T'_{j'} = \left\lceil \frac{mT'_k}{l\lfloor T_{k'-1}/T_{j'} \rfloor} \right\rceil \quad (1 \leq j' \leq k'-1) \tag{14}$$

T'_k が最小周期の場合は l は算出できないので、スレッド k 以外の全スレッドの周期 T_j は $T_j < mT'_k$ ならば、

$$T'_j = mT'_k \quad (1 \leq j \leq n\,,\, j \neq k) \tag{15}$$

に調整する。

さらに $m > l$ の場合は スレッド k 以外のスレッドの周期調整によってスレッドの順序が変化する可能性がある。すなわち周期調整前は

$$1, \ldots, k''-1, k'', k''+1, \ldots, k'-1, k, k', \ldots, n$$

という順番だったスレッドの集合が、スレッド k' 以外のスレッドの周期を調整した結果

$$1, \ldots, k''-1, k, k'', k''+1, \ldots, k'-1, k', \ldots, n$$

という順番になる可能性がある。よって全スレッドの周期を調整し終った後、それらをその周期にしたがってソートする必要がある。

2.3 スレッドの周期調整の順序制御

2.2節で述べたアルゴリズムでは、実際に各スレッドを新しい周期に切替えるタイミングについては考慮していない。例えばスレッドの周期を短くする要求が受理されてシステム全体の負荷が増加する場合は、他のスレッドの周期を先に新しい値に切替える必要がある。よってCPU資源の使用率を増加させる以下の3種の変更要求

- 新しいスレッドの追加
- スレッドの実行時間の増加
- スレッドの周期の縮小

が受理された場合は、他のスレッドのCPU資源の割当を先に減少させてからターゲットスレッドのCPU資源の割当を増加させるようにスレッドの周期調整の順序を決め、作業は同期的に行なうように訂正する。

3 CPU資源管理システム

3.1 プロトタイプシステムの構成

図1はマイクロカーネルと各タスクの関係を示している。点線内がCPU資源管理システムで、

- サーバ側 (CPU資源管理)
  サーバタスク (MART server)
- クライアント側 (アプリケーション)
  クライアントタスク (client) とユーザインタフェースタスク (user interface)

という構成になっている。図中の矢印はタスク間の通信関係を表している。ユーザインタフェースタスクはクライアントタスクにメッセージを送信するが受信はしない。クライアントタスクとサーバタスクは相互にメッセージ通信している。ターゲットマシンとしては以下の仕様を持つ IBM PC 互換機を使用している。

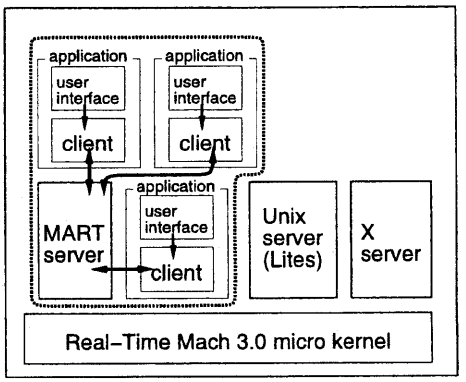


図 1: 資源管理のプロトタイプシステムの構成

- CPU ··· Pentium/133MHz
- メモリ ··· 16MB
- キャッシュ ··· 256KB SRAM (2次キャッシュ)、16KB (CPU 内蔵 1次キャッシュ)

3.2 サーバとクライアントの動作

図2は各タスクの関係とそれぞれの動作の概略を示している。タスク間にまたがる矢印はそのタスク間の通信の形式と内容を表している。例えば一番上のUターンしている矢印は同期通信で、クライアントは登録要求のメッセージをサーバに送信しサーバからの登録完了のメッセージの受信を待つ。図2には記していないが、各タスク中にはメッセージ受信専用のスレッドがありメッセージはいつでも受信できる。MART サーバは、複数のアプリケーションのCPU 資源を管理でき、1つのホスト上にただ1つ存在する。アプリケーションはクライアントタスクとユーザインタフェースタスクのペアで構成する。

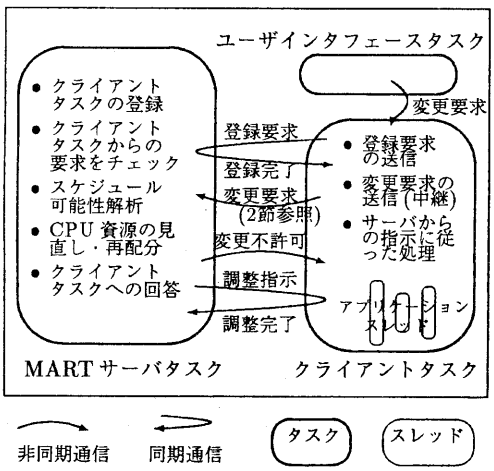


図 2: サーバ／クライアントの関係とそれぞれの動作

3.2.1 クライアント (アプリケーション) 側の動作

ユーザインタフェースタスク

ユーザの要求表現をサポートするタスクである。ユーザは「新しいスレッドの起動」などの要求を出すだけでその要求のその後については関与しないので、クライアントタスクとは非同期通信を行なう。

クライアントタスク

クライアントタスクはタスク内のスレッドを管理する。最初に MART サーバに対してクライアント自身を CPU 資源管理の対象にしてもらうための登録要求を送信する。登録完了の返事を受信した後はユーザインタフェースタスクから送信されてくる変更要求をサーバが解釈できる要求とスレッドの属性値に変換してサーバに送信する。本プロトタイプシステムでは1つのスレッドを「識別子・実行時間・周期」の3つの属性値で表現するので、変更要求の種類に応じて属性値のいくつかを指定する必要がある。例えばサーバには「新しいスレッドの生成」を要求することができるが、その場合には新しいスレッドの実行時間と周期をスレッドの属性値としてサーバに申請しなければならない。要求できる変更の種類は2節で述べた。またサーバは複数のクライアントを管理できるので、クライアントは自分のタスク識別子 (task_id) もサーバに送信しなければならない。
task_id は MART サーバからの登録完了のメッセージで与えられる整数である。サーバからは表1のような応答を受信し、クライアントはその応答に従って指示された処理を行なう。例えば新しいスレッドの起動を要求しサーバから「要求をそのまま受け入れる」という応答を受信したら、クライアントは申請した通りの実行時間と周期を持つ新しいスレッドを起動することができる。

表 1: サーバからの応答とその応答に対してクライアントが行なう処理

サーバの回答	クライアントが行なう処理
要求不許可	――――――――
要求をそのまま受け入れる	要求したスレッドを申請通りにセットアップする
CPU 資源の調整が必要	指定されたスレッドを指定されたパラメータ値に調整する[2]

3.2.2 サーバ (CPU 資源管理) 側の動作

MART サーバタスク

登録要求を送信してきたクライアントを登録し今後の CPU 資源管理の対象とする。その後はクライアントからの要求を受信する。まずその要求が受け入れ可能かどうか判断する。例えば実行時間より短い周期への変更が要求された場合などには「要求不許可」のメッセージをクライアントに送信する。一方受け入れた要求に対しては、要求通りの CPU 資源をクライアントに配分できるかどうかを式 (1) を使って判断する。CPU 資源を要求通りに配分 (確保) でき、しかも全スレッドをスケジュール可能な状態に保ち続けられるならばクライアントに「要求をそのまま受け入れる」と通知する。しかし CPU 資源を要求通りには配分できないと判断した場合は MART アルゴリズムを用いて CPU 資源の配分を調整する。CPU 資源の再配分は複数のスレッドの周期を調整す

[2]複数のスレッドを再起動する際に、その順番に注意する必要があることは 2.3節で述べた。

ることで実現する。CPU 資源管理の対象としている全タスクの全スレッドの CPU 資源の配分調整計算が終了したら、周期を調整する必要のあるスレッドを持つ全てのクライアントに周期調整が必要なスレッドの識別子と新しい周期の値を送信し、その完了を待って、要求を出しているクライアントに指示を送信する。

4 評価

4.1 MART サーバによる周期調整

表2のシナリオに基づいて3本のスレッドを生成し、それぞれの変更要求によって、各スレッドの周期が調整されている様子および時間制約を満足できなかった回数を図3に示す。表2は、例えば「スレッド3は時刻20秒に生成されて、その20秒後に周期を30msec に設定する要求を出している。」ということを表している。本実験で MART サーバは CPU 資源の 80% を管理している。図3は横軸が時間 (秒)、左縦軸が調整された周期 (ミリ秒)、右縦軸がデッドラインミスの回数 (回) を表している。

表 2: 各スレッドの振舞いのシナリオ

thread No.	実行期間 sec	実行時間 msec	設定周期 msec
1	0 → 70	20	50
2	10 → 30	30	70
	30 → 50	40	—
3	20 → 40	10	30
	40 → 60	—	30

図3より、スレッドは3本生成されているがスレッド2以外は時間制約を常に満足していたということが分かる。スレッド2も最初に起動された時の10秒間にデッドラインをミスしているだけで、その後は安定した動作をしている。また各スレッドは新しいスレッドが生成されたり他のスレッドがより多くの実行時間を要求したりするなどして CPU 使用率を増加させようとする度に周期が調整されており、新しく設定された周期で時間制約を満足しながら動作している。 逆にスレッドが終了するなどして CPU 使用率が下がる場合は特に MART サーバが関与していない。MART サーバはスレッド数やスレッドの時間属性の変更に対応することができ、しかも過負荷な状態になりそうだと判断すると、各スレッドの周期を調整して全スレッドをスケジュール可能な状態に保つことができる。

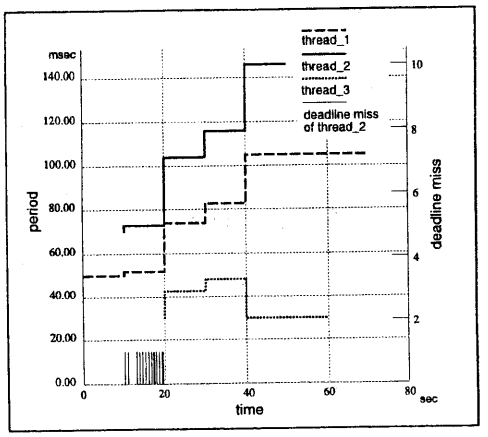


図 3: MART サーバによって調整されたスレッドの周期とデッドラインミスの回数

4.2 MART サーバの実行時間とメッセージの往復時間

MART サーバの処理時間とメッセージの通信時間を測定し結果を図4に示す。実験では MART サーバは CPU 資源の 80% を管理し、クライアントは「実行時間 200 msec, 周期 400 msec」という属性値を持つスレッドを1個ずつ、合計20個生成する要求をサーバに送信する。 スレッドの生成は過負荷な状況を起こす可能性があると判断される要求なので、最初の1個めのスレッドを起動する時以外はサーバは毎回スケジュール可能性を検証し CPU 資源の再割当を行なう。ここでは、

T_t: クライアントがサーバに要求を送信してからサーバからの応答を受信するまでの時間

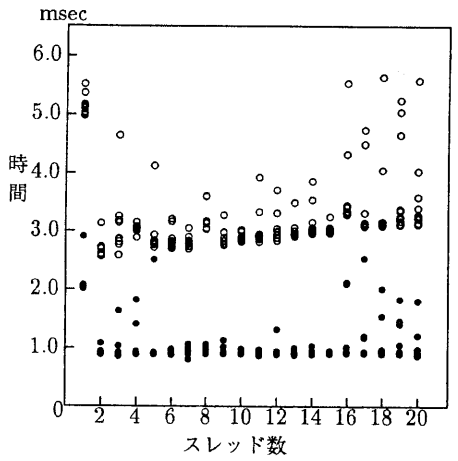
T_s: サーバがクライアントからの要求を受信してからクライアントへの応答を送信するまでの時間

を計測した。図4では白丸 (○) は T_t を、黒丸 (●) は T_s を表す。$(T_t - T_s)$ はメッセージの往復時間に相当し、T_s はサーバの処理時間に相当する。

図4より、T_s はスレッド数が20個以下の場合はスレッド数に関わらず 約 0.9 msec であり、T_t はスレッド数が増えると若干増加するが全体として 約 3 msec であったので、サーバ自体の処理時間は 約 0.9 msec, メッセージの往復時間は 約 2 msec である。結果としてスレッドの実行時間が数十ミリ秒以上の場合は MART サーバの実行が大きな負担にはならないといえる。

5 関連研究

リアルタイムスケジューリングアルゴリズムは多数提案されているが、MART アルゴリズムのように



- ○ クライアントがメッセージを送信してからサーバからのメッセージを受信するまでの時間
- ● サーバがクライアントからのメッセージを受信してから、クライアントにメッセージを送信するまでの時間

図 4: MARTサーバの実行時間とメッセージの往復時間

システム内のスレッドの時間的属性値を変更するものとしては、負荷調整法 [2]、Mode Change 法 [8]、Self Stabilize 法 [10]、Imprecise Computation [3] などがある。負荷調整法は、外的な要因によるシステムの変化に対応するためのアルゴリズムでMARTアルゴリズムと似ているが、処理数の変化には対応しない。Mode Change 法は Rate Monotonic アルゴリズムを拡張したもので、処理を分割することで周期を調整し優先度を制御するためのアルゴリズムで本質的には静的なアルゴリズムである。Self Stabilize 法はシステムの負荷に応じて自分自身の周期を制御する自律的な周期調整機能で、周期の制御方法が不明確で動作の検証が困難だが今後はMARTサーバと動作比較などを行なうべき有効な方法である。Imprecise Computation は、生成される結果の質とその結果を出すために必要な処理時間とのトレードオフを考えるという方法で、処理の実行時間を調整する方法である。この方法は生成される結果が処理された時間と共に単調増加する場合に有効である。これもMARTサーバとの併用動作を検討すべき有効な方法である。

6 おわりに

本報告では資源管理の基本アルゴリズムとして採用した「MARTアルゴリズム」を検討し、スレッドにCPU資源を再配分する際に重要な再配分の順序制御部分を定義した。次にその改良したMARTアルゴリズムを基にプロトタイプシステムとして Real-Time Mach 3.0 上に資源管理用のサーバ「MARTサーバ」を実装し、MARTサーバが調整したスレッドの周期とその周期で動作しているスレッドのデッドラインミスの回数を計測した。またMARTサーバの実行時間とサーバとクライアント間のメッセージ通信時間を測定した。

結果としてMARTサーバはスレッド数やスレッドの時間属性の変更に対応することができ、しかも過負荷な状態になりそうだと判断するとスレッドの周期を調整して全スレッドのスケジュール可能性を維持できることを実証した。またサーバ自体は 約 0.9 msec で、通信を含めて約 3msec で動作するので数十ミリ秒程度の実行時間を持つスレッドのCPU資源を管理する場合、十分軽く動作することを確認した。

MART サーバはスケジュール不可能になりそうな状況にしか対処していない。今後はCPU使用率が下がりCPUのアイドルが多くなる状況にも対処できるようにCPUの有効利用について検討する。

参考文献

[1] Naoko Kosugi, Kazunori Takashio, and Mario Tokoro. Modification and Adjustment of Real-Time Tasks with Rate Monotonic Scheduling Algorithm. In *The second workshop on Parallel and Distributed Real-Time Systems*, pp. 98–103, 1994.

[2] Tei-Wei Kuo and Aloysius K. Mok. Load Adjustment in Adaptive Real-Time Systems. In *IEEE proceedings Real-Time System Symposium*, pp. 160–170, 1991.

[3] Kwei-Jay Lin, Swaminathan Natarajan, and Jane W.-S. Liu. Imprecise Results : Utilizing Partial Computations in Real-Time Systems. In *IEEE proceedings Real-Time System Symposium*, pp. 210–217, 1987.

[4] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the Association for Computing Machinery*, Vol. 20, No. 1, pp. 46–61, January 1973.

[5] Yoshifumi Manabe and Shigemi Aoyagi. A Feasibility Decision Algorithm for Rate Monotonic Scheduling of Periodic Real-Time Tasks. In *Technical Report of IEICE. COMP94-31*, pp. 63–72. The Institute of Electronics, Information and Communication Engineers, July 1994.

[6] Real-Time Mach Project, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213. *Real-Time Mach 3.0 User Reference Manual*, June 1995.

[7] Lui Sha and John B. Goodenough. Real-Time Scheduling Theory and Ada. *IEEE Computer*, pp. 53–62, April 1990.

[8] Lui Sha, Ragunathan Rajkumar, John Lehoczky, and Krithi Ramamritham. Mode Change Protocols for Priority-Driven Preemptive Scheduling. *The Journal of Real-Time Systems*, Vol. 1, No. 3, pp. 243–264, December 1989.

[9] John A. Stankovic. Scheduling Algorithms for Hard Real-Time Systems–A Brief Survey. In John A. Stankovic and Krithi Ramamritham, editors, *Hard Real-Time Systems*, pp. 150–173. IEEE Computer Society, 1991.

[10] 松渡, 徳田. Real-Time Mach 3.0 における連続メディアサーバの実験. 情処研報 93-OS-60, pp. 75–82, 1993.