VBR ストリーム処理のための適応的スケジューリングポリシとその性能評価

滝沢泰久[†] 芝 公仁[††] 大久保英嗣[†††]

動画や音声に代表される連続メディアを扱うストリーム処理タスクをスケジュールする場合，それらの時間制約を満たすために，最悪処理時間に基づいたリアルタイムスケジューラが利用されてきた．しかし，多くの連続メディアデータは，圧縮・伸長処理や差分処理などにより，処理時間は必ずしも一定とはならない．このような連続メディアデータを従来の最悪処理時間に基づいた方式で処理すると，過度な CPU 利用率の予約のために，システム全体の CPU 利用率が大きく低下する．本論文では，以上の問題を解決するために，ストリーム処理タスクの時間制約と処理量の変動に動的に適応するスケジューリングポリシを提案する．提案ポリシは，連続メディア資源モデルである Linear Bounded Arrival Process に変動処理量の変更を加えたモデルを基本とし，そのモデルに Parallel Distributed Processing モデルと熱力学モデルを適用している．本論文では，提案ポリシの詳細と性能評価について述べる．

An Adaptive Scheduling Policy for VBR Stream Processing and Its Performance Evaluation

Yasuhisa Takizawa,[†] Masahito Shiba[††] and Eiji Okubo[†††]

Real-time scheduling policies based on the worst-case execution time have been used for stream processing tasks which manipulate continuous media such as voice, audio, video and animation. However, the processing time for continuous media stream dynamically changes as the result of compressions, extensions and differences between data. Therefore, conventional schemes cause excessive reservations of the processing time since the processing time of each continuous datum is shorter than its worst-case execution time. In this paper, a new scheduling policy which is adaptable for stream processing tasks with timing constraints and processing delay is proposed. The proposed policy is based on the model which modifies Linear Bounded Arrival Process, and applies Parallel Distributed Processing model and thermal dynamics model to this model. In this paper, details of the proposed policy and its performance evaluation are described.

1. はじめに

マイクロプロセッサの急速な進歩により，パーソナルコンピュータ（以降 PC）やワークステーション（以降 WS）上で動画や音声に代表される連続メディアを処理するアプリケーション（以降連続メディアアプリケーション）が数多く出現している．連続メディアアプリケーション[5]は，それらの処理するメディアの特性上，ある時間周期でデータが発生し，次の周期までに処理を完了しなければならない．すなわち，連続メディアデータを処理するタスクは，周期タスクモデルに基づいた時間制約を持つ．PC や WS 上のマルチメディアシステムにおける連続メディア処理は，このような時間制約を持つタスクが複数実行され，かつ直列にデータ通信を行うストリーム処理により行われる場合が多い．このようなストリーム処理タスクをスケジュールする場合，その時間制約を満たすために，最悪実行時間と周期タスクモデルに基づいて CPU 利用率を予約したうえで，リアルタイムスケジューラを用いる方式が有効とされている．

一方，連続メディアデータは，入出力装置から固定周期で生成/消費され，そのデータ量は動画データなどの圧縮/伸張や差分処理[7]に見られるように可変量

† 株式会社 ATR 環境適応通信研究所
ATR Adaptive Communications Research Laboratories
†† 立命館大学大学院理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
††† 立命館大学理工学部情報学科
Department of Computer Science, Facutly of Science and Engineering, Ritsumeikan University

図 1 連続メディアデータのストリーム処理
Fig. 1 Stream processing for continuous media data.

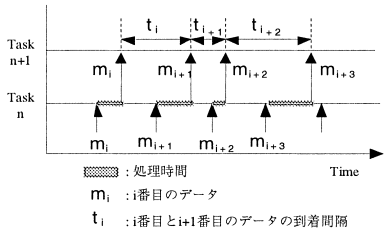


図 2 パイプラインモデルにおける連続メディアデータ到着の非周期性
Fig. 2 Aperiodic arrival interval between continuous media data in pipeline model.

となる．すなわち，連続メディアデータは，Variable Bit Rate（以降 VBR）データであるといえる．このような連続メディアデータのストリーム処理は，順次処理されるパイプラインモデルにより行われる（**図 1**）．このパイプライン上のタスクは VBR データを処理するため，その処理遅延も可変となる．すなわち，VBR データを処理するタスクの処理遅延は可変となり，パイプライン上のタスクにおけるデータの到着は周期性を失う（**図 2**）．このため，周期タスクモデルをそのままストリーム処理に適用すると，タスクにデータ未到着による待ち時間が発生しスケジュール可能性が低下する．これを回避するためには，パイプライン上の隣接するタスク間に十分な初期位相が必要であり，そのため，エンド–エンドの処理遅延が大きくなる．また，タスクの時間制約を満たすために，最悪実行時間に基づき CPU 利用率を予約することは，VBR データ処理環境では過度な予約量となり，システム全体での CPU 利用率が大きく低下する．

本論文では，以上の問題点を解決するために，以下の 4 つを前提として，従来の方式より高いスケジュール可能性を導く適応的スケジューリングポリシを提案する．

- 最悪実行時間より短い時間を CPU 使用時間と想定する．
- 各タスクのメッセージ処理に必要な CPU 使用時間は，ランダムに変動する．
- 各タスクは，連続メディアデータをパイプラインモデルにより処理する．
- 連続メディアデータは，その入出力装置から VBR/固定周期で生成/消費される．
- 複数のストリーム処理が混在する．

提案ポリシでは，ストリーム処理モデルを連続メディア資源モデル Linear Bounded Arrival Process[8]（以降 LBAP）に VBR データ処理のための変更を加えたモデルとし，Parallel Distributed Processing モデル（以降 PDP モデル）[6]と熱力学的モデル[3]を動作メカニズムとしている．これにより，VBR データに動的に適応するタスクスケジューリングが可能となる．

以下，2 章で関連研究とそれらの研究における問題点について述べ，3 章で VBR ストリーム処理モデルを定義する．次に，4 章で，3 章の定義に基づき，従来の方式の問題点を解決するための新たなスケジューリングポリシを提案し，そのアルゴリズムを 5 章で述べる．さらに，6 章で提案ポリシの性能評価結果について述べ，その有効性を議論する．

2. 関 連 研 究

2.1 リアルタイムスケジューラと周期タスクモデル

周期タスク処理に適しているリアルタイムスケジューラには，Earliest Deadline First（以降 EDF）[2]と Rate Monotonic（以降 RM）[2]がある．EDF と RM で用いられる周期タスクモデルは，固定周期と周期内の実行時間から構成され，次のように，タスクの時間属性を定義する．

$$a_n^i = a_n^0 + iT_n$$
$$d_n^i = a_n^0 + (i+1)T_n$$
$$U_n = C_n/T_n$$

ここで，a_n^i は，タスク n における i 周期目のタスク到着時刻，T_n はタスク n におけるタスク起動周期，d_n^i はタスク n における i 周期目のデッドライン時刻，U_n はタスク n における CPU 利用率，C_n はタスク n における実行時間である．

このような周期タスクモデルとタスクの最悪実行時間 C_n^{max} に基づき，CPU 利用率予約を行うシステムには，リアルタイム Mach[4]などがあり，その CPU 利用率 U_n は次のようになる．

$$U_n = C_n^{max}/T_n \quad (1)$$

しかし，VBR ストリーム処理環境において，多くのデータの処理時間は最悪実行時間 C_n^{max} より小さい．すなわち，上式による CPU 利用率予約は，過度な予約量となる．そのため，システム全体としての CPU 利用率が大きく低下する．

2.2 時間制約を満たすための初期位相

周期タスクモデルでは，データ到着が周期的であることを想定している．しかし，パイプラインモデルに

よるVBRストリーム処理においては，前述のように，データ到着は周期性を失う．そのため，周期的なタスク到着とデータの到着が同期せず，タスクにデータ未到着による待ち時間が発生する．この待ち時間は，優先度逆転[1)]に陥る可能性を高め，スケジュール可能性を低下させる要因となる．

これを回避するためには，タスク到着時刻以前にデータが到着する必要がある．すなわち，パイプライン上の各タスクは，隣接する前方タスクが処理完了後に到着する必要があり，そのために，タスク間に十分な初期位相を設定する．初期位相とは，相対タスクの最初の到着時刻と当該タスクの最初の到着時刻の時間差である．周期タスクモデルよるVBRストリーム処理において，式(1)に基づき，CPU利用率を予約している場合，連続メディアデータ出力装置の時間制約を満たすのに必要なタスク初期位相は，次のようになる．

$$a_n^0 \geq a_{n-1}^0 + T_{n-1}$$

したがって，出力装置の時間制約を保証する場合，大きなエンド–エンドの遅延時間が必要となる．

3. VBRストリーム処理モデル

本章では，LBAPについて説明し，LBAPに基づいた従来の処理モデルについて議論する．そのうえで，VBRストリーム処理モデルをLBAPに基づいて定義する．

3.1 LBAP

LBAPでは，処理すべきデータを以下の3つのパラメータにより特徴付けたメッセージのストリームしている．

R^{max}: maximum message rate (messages/second)

W^{max}: maximum workahead (messages)

S^{max}: maximum message size (bytes)

LBAPでは，時間間隔tにおいて到着する最大メッセージ数は$R^{max}t+W^{max}$であると定義している．すなわち，長期的なデータ処理レートは$R^{max}S^{max}$以下であるが，パラメータW^{max}により示されるバースト的なデータ到着により，データレートは短期間$R^{max}S^{max}$を超えることを許容している．このバースト的なデータ到着にともなう未処理メッセージ数を次のように定義する．

$$w_n(m_0) = 0$$
$$w_n(m_i) = \max(0, w_n(m_{i-1}) - (a_n(m_i) - a_n(m_{i-1}))R^{max} + 1)$$

ただし，$w_n(m_i)$はタスクnにおけるi番目のメッセージ到着時の未処理メッセージ数，$a_n(m_i)$はタスクnにおけるi番目のメッセージ到着時刻である．したがって，タスクnにおけるW^{max}，すなわちW_n^{max}は次のように定義できる．

$$W_n^{max} = \max_i(w_n(m_i)) \quad (2)$$

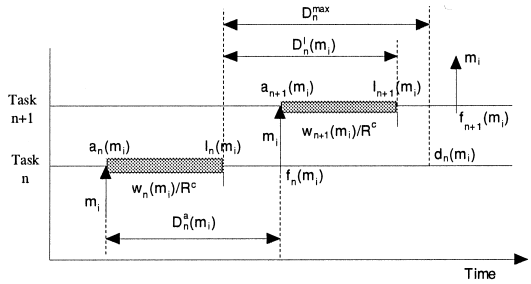


図3 LBAPにおける時間属性
Fig. 3 Timing attributes on LBAP.

到着したメッセージは，未処理メッセージをすべて処理完了した後に，初めて処理対象となる．LBAPでは，メッセージが処理対象となる論理的時刻をタスクnにおけるi番目のメッセージのメッセージ論理到着時刻$l_n(m_i)$として，次のように定義する．

$$l_n(m_i) = a_n(m_i) + w_n(m_i)/R^{max}$$

$$l_n(m_0) = a_n(m_0)$$
$$l_n(m_i) = \max(a_n(m_i), l_n(m_{i-1}) + 1/R^{max})$$

また，LBAPでは，タスクnにおけるi番目のメッセージにおける処理遅延時間$D^a(m_i)$，論理処理遅延時間$D^l(m_i)$およびデッドライン時刻$d_n(m_i)$を，メッセージ到着時刻とメッセージ論理到着時刻を用いて，次のように定義する（**図3**参照）．

$$D_n^a(m_i) = a_{n+1}(m_i) - a_n(m_i) \quad (3)$$
$$D_n^l(m_i) = l_{n+1}(m_i) - l_n(m_i) \quad (4)$$
$$d_n(m_i) = l_n(m_i) + D_n^{max} \quad (5)$$

ただし，D_n^{max}は，前述のパラメータにより資源予約した場合の最大の論理処理遅延時間であり，次のように定義できる．

$$D_n^{max} = \max_i(D_n^l(m_i))$$

LBAPでは，このように特徴付けられたメッセージストリームの処理を，次のように行う．

- タスクのデッドライン時刻に基づき，EDFにより実行順を決定する．
- 前述のように定義されたタスクがパイプラインモデルにより複数接続される．

- タスクは，メッセージの到着時刻に起動される．
- タスクは，メッセージ処理完了時刻に未処理メッセージがある場合，処理完了したメッセージのデッドライン時刻を待たずにただちに起動される．

3.2 LBAP に基づく従来の処理モデル

従来の方式[9]では，ストリームデータを一定のメッセージの到着レートと最悪実行時間により特徴付け，CPU 利用率を予約している．すなわち，各タスクの時間制約を満たすために，すべてのメッセージ処理時間が最悪実行時間である場合でも，十分に処理できる CPU 利用率を予約している．したがって，従来の方式は，最悪実行時間に基づいた Constant Bit Rate（以降 CBR）ストリーム処理として考えられる．このような CBR ストリーム処理を，LBAP のパラメータを用いて表すと，次のようになる．

R^c: constant message rate (messages/second)
W^{max}: maximum workahead (messages)
C^{max}: maximum processing time for a message (seconds/message)

この定義により，LBAP を用いた最悪実行時間に基づく CBR ストリーム処理モデルにおける未処理メッセージ数，およびメッセージ論理到着時刻は，次のようになる．

$$\begin{aligned} &w_n(m_0) = 0 \\ &w_n(m_i) = \max(0, w_n(m_{i-1}) \\ &\qquad -(a_n(m_i) - a_n(m_{i-1}))R^c + 1) \end{aligned} \tag{6}$$

$$l_n(m_i) = a_n(m_i) + w_n(m_i)/R^c \tag{7}$$

$$\begin{aligned} &l_n(m_0) = a_n(m_0) \\ &l_n(m_i) = \max(a_n(m_i), l_n(m_{i-1}) + 1/R^c) \end{aligned} \tag{8}$$

その他の定義は，3.1 節と同様である．上記のように，定義された時間属性において，パラメータ C^{max} と R^c により CPU 利用率を予約している場合，連続メディアデータ出力装置の時間制約を保証するには，周期タスクモデルと同様に，隣接するタスク間に十分な初期位相が必要である．その初期位相は，次のようになる．

$$a_n(m_i) \geq a_0(m_0) + \sum_{j=0}^{n-1}(W_j^{max} + 1)/R^c$$

したがって，出力装置の時間制約を保証する場合，最悪実行時間に基づく CBR ストリーム処理モデルにおいても，大きなエンド–エンドの遅延時間が必要となる．

3.3 VBR データにおけるストリーム処理モデル

前述のように，最悪実行時間に基づく CBR ストリーム処理モデルは，過度な CPU 予約量となり，システム全体での CPU 利用率が低下する．また，出力装置の時間制約を満たすためには，十分な初期位相が必要である．そのため，エンド–エンドの処理遅延時間が大きくなる．

上記の問題を解決するために，メッセージ処理時間を最悪実行時間 C^{max} より短い任意の時間 C^{req} とする．本節では，C^{req} に基づいて，次のパラメータのように，CPU 利用率を予約した場合の処理モデルについて考える．

R^c: constant message rate (messages/second)
b: actual workahead messages (message)
C^{req}: request processing time for a message (seconds/message)

タスク n において，レート R^c で最悪実行時間より短い時間 C^{req} を，CPU 利用率として予約している場合，すべてのメッセージの処理時間が $1/R^c$ 以下であることは保証されない．したがって，式 (6)，(7)，(8) のような予測は困難である．そこで，VBR データ処理モデルにおける時間属性は，変動する処理時間を考慮して定義する．

未処理メッセージ数の定義は，$w_n(m_i)$ に代わり，任意の時刻に実測された未処理メッセージ数を用いる．タスク n において，時刻 t に実測された未処理メッセージ数を $b_n(t)$ と表記し，実効未処理メッセージ数と呼ぶ．

メッセージ論理到着時刻 $l_n(m_i)$ に代わり，メッセージ実効到着時刻 $e_n(m_i)$ を定義する．$l_n(m_i)$ は，メッセージが初めて処理対象と予測される論理時刻である．このことから，メッセージ実効到着時刻 $e_n(m_i)$ は，タスク n において，i 番目のメッセージが，初めて処理対象となった実際の時刻とし，次のように定義する．

$$\begin{aligned} &e_n(m_0) = a_n(m_0) \\ &e_n(m_i) = \max(a_n(m_i), f_n(m_{i-1})) \end{aligned} \tag{9}$$

ただし，$f_n(m_{i-1})$ は，タスク n における $i-1$ 番目のメッセージの処理完了時刻である．

次に，デッドライン時刻について定義する．パイプラインモデルにおいて，タスク $n+1$ における i 番目のメッセージが処理対象となる時刻は，タスク $n+1$ における $i-1$ 番目のメッセージの処理完了時刻 $f_{n+1}(m_{i-1})$ である．すなわち，タスク n は，i 番目のメッセージの処理を，$f_{n+1}(m_{i-1})$ までに完了す

ればよいと考えられる．したがって，タスク n における i 番目のメッセージのデッドライン時刻は，次のように考える．

$$d_n(m_i) = f_{n+1}(m_{i-1})$$

時刻 $e_n(m_i)$ において，$f_{n+1}(m_{i-1})$ は不明である．しかし，周期 $1/R^c$ 内でメッセージの処理を完了しなければ，後続タスクの周期的時間制約を満たせなくなる可能性が高まる．このことから，メッセージ処理が $1/R^c$ 内で完了すると仮定し，時刻 $e_n(m_i)$ におけるタスク $n+1$ の実効未処理メッセージ数から，$f_{n+1}(m_{i-1})$ を次のように想定する．

$$\bar{f}_{n+1}(m_{i-1}) = e_n(m_i) + b_{n+1}(e_n(m_i))/R^c$$

したがって，デッドライン時刻 $d_n(m_i)$ を，次のように定義する．

$$d_n(m_i) = e_n(m_i) + b_{n+1}(e_n(m_i))/R^c \quad (10)$$

その他の定義は，3.1 節と同様である．

3.4 VBR ストリーム処理モデルにおける特性

本節では，最悪実行時間より短い処理時間に基づき CPU 利用率を予約する場合の VBR ストリーム処理モデルの特性に関して考察する．

[定理 1] VBR ストリーム処理モデルにおいて，最悪実行時間より短い時間を，パラメータ C^{req} と R^c により CPU 利用率として予約する場合，メッセージの処理完了時刻は，以下の条件で，デッドライン時刻を満たす．

$$b_{n+1}(e_n(m_i))/R^c \geq D_n^a(m_i)$$

[証明] メッセージの処理完了時刻が，デッドライン時刻を満たすためには，次のような条件となる．

$$f_n(m_i) \leq d_n(m_i) \quad (11)$$

上式を，式 (9) のメッセージ実効到着時刻の定義に従い，$a_n(m_i)$ が $f_n(m_{i-1})$ より大きい場合と，そうでない場合とに分けて考える．

$a_n(m_i) \geq f_n(m_{i-1})$ の場合，$e_n(m_i)$ は $a_n(m_i)$ となるので，式 (9) により，式 (11) は次のようになる．

$$f_n(m_i) \leq a_n(m_i) + b_{n+1}(e_n(m_i))/R^c$$

また，メッセージ処理完了時刻 $f_n(m_i)$ は，メッセージ到着時刻 $a_n(m_i)$ に処理遅延時間 $D_n^a(m_i)$ を加えた時刻である．したがって，上式は次のようになる．

$$b_{n+1}(e_n(m_i))/R^c \geq D_n^a(m_i) \quad (12)$$

したがって，$a_n(m_i) \geq f_n(m_{i-1})$ の場合，上式を満たすならば，処理モデルはデッドライン時刻を満たす．

次に，$a_n(m_i) < f_n(m_{i-1})$ の場合について考える．この場合，$e_n(m_i)$ は $f_n(m_{i-1})$ となるので，式 (9) により，式 (11) は次のようになる．

$$f_n(m_i) \leq f_n(m_{i-1}) + b_{n+1}(e_n(m_i))/R^c$$

パイプラインモデルにおいて，タスク n のメッセージ処理完了時刻 $f_n(m_i)$ は，タスク $n+1$ のメッセージ到着時刻 $a_{n+1}(m_i)$ と一致する．したがって，上式は，次のようになる．

$$\frac{b_{n+1}(e_n(m_i))}{R^c} \geq a_{n+1}(m_i) - a_{n+1}(m_{i-1}) \quad (13)$$

また，場合分けの条件も同様に，次のようになる．

$$a_n(m_i) < a_{n+1}(m_{i-1}) \quad (14)$$

ここで，式 (13) の右辺の式 $a_{n+1}(m_i) - a_{n+1}(m_{i-1})$ について考える．この式を $D_n^a(m_i)$ を用いて表すと，次のようになる．

$$\begin{aligned} &a_{n+1}(m_i) - a_{n+1}(m_{i-1}) \\ &\quad = (a_{n+1}(m_i) - a_n(m_i)) \\ &\qquad + a_n(m_i) - a_{n+1}(m_{i-1}) \\ &\quad = D_n^a(m_i) + a_n(m_i) - a_{n+1}(m_{i-1}) \end{aligned}$$

上式は，式 (14) により，次のようになる．

$$a_{n+1}(m_i) - a_{n+1}(m_{i-1}) = D_n^a(m_i) - x$$

ただし，x は正の数である．

このことから，式 (13) は次のようになる．

$$\frac{b_{n+1}(e_n(m_i))}{R^c} \geq D_n^a(m_i) - x \quad (15)$$

したがって，$a_n(m_i) < f_n(m_{i-1})$ の場合，上式を満たすならば，処理モデルはデッドライン時刻を満たす．

以上のことから，2 つの場合ともに，デッドライン時刻を満たすには，式 (12)，(15) を同時に満たす必要がある．したがって，次のようになる．

$$\frac{b_{n+1}(e_n(m_i))}{R^c} \geq D_n^a(m_i)$$

□

[定理 2] 定理 1 を満たす VBR ストリーム処理モデルにおいて，以下の条件を満たすならば，メッセージ処理に必要な CPU 利用率は，予約した CPU 利用率を超えない．

$$b_{n+1}(e_n(m_i)) \geq \frac{C_n(m_i)}{C_n^{req}} \quad (16)$$

ただし，$C_n(m_i)$ は，タスク n における i 番目のメッセージ処理に使用する CPU 時間である．

[証明] タスク n において，1 メッセージ処理あたりに予約した CPU 使用時間を C_n^{req} とすると，CPU 予約量は $C_n^{req}R^c$ である．したがって，タスク n における i 番目のメッセージの処理のために使用した CPU 利用率 $U_n(m_i)$ は，次のようになる．

$$U_n(m_i) = \frac{C_n(m_i)}{d_n(m_i) - a_n(m_i)} \quad (17)$$

処理モデルは定理 1 を満たすことから，メッセー

ジ処理完了時刻はデッドライン時刻より小さい．したがって，上記の $[a_n(m_i), d_n(m_i))$ の時間間隔は，次のようになる．

$$\begin{aligned} d_n(m_i) - a_n(m_i) &\geq f_n(m_i) - a_n(m_i) \\ &\geq D_n^a(m_i) \end{aligned}$$

上式を式 (17) に適用すると $U_n(m_i)$ は次のようになる．

$$U_n(m_i) \leq \frac{C_n(m_i)}{D_n^a(m_i)}$$

タスク n において，処理に必要な CPU 利用率 $U_n(m_i)$ が，予約した CPU 利用率 $C_n^{req}R^c$ を超えないためには，次式を満たす必要がある．

$$U_n(m_i) \leq \frac{C_n(m_i)}{D_n^a(m_i)} \leq C_n^{req}R^c$$

したがって，上式は，次のようになる．

$$D_n^a(m_i) \geq \frac{C_n(m_i)}{C_n^{req}R^c}$$

さらに，定理 1 から，上式は次のようになる．

$$\frac{b_{n+1}(e_n(m_i))}{R^c} \geq D_n^a(m_i) \geq \frac{C_n(m_i)}{C_n^{req}R^c}$$

□

[定理 3] VBR ストリーム処理モデルにおいて，最悪実行時間より短い時間を，パラメータ C^{req} と R^c により，CPU 利用率として予約している場合，タスク n の初期位相が以下を満たすならば，連続メディアデータ出力装置の時間制約が満たされることが保証される．

$$a_n(m_0) \geq a_0(m_0) + \sum_{j=0}^{n-1} D_j^a(m_i) \qquad (18)$$

[証明] 出力装置の時間制約を満たすには，出力装置は $1/R^c$ 時間ごとにメッセージを必要とすることから，次式が成り立つ必要がある．

$$a_{out}(m_i) \leq a_{out}(m_0) + i/R^c$$

ただし，$a_{out}(m_i)$ は出力装置における i 番目のメッセージの到着時刻である．

出力装置をパイプライン上の終端タスク n の次のタスク，すなわちタスク $n+1$ として考えると，上式は，次のようになる．

$$a_{n+1}(m_i) - a_{n+1}(m_0) \leq i/R^c \qquad (19)$$

パイプラインモデルにおいて，隣接するタスクのメッセージ到着時刻 $a_{n+1}(m_i)$ とメッセージ処理完了時刻 $f_n(m_i)$ が一致することから，$a_{n+1}(m_i)$ は，次のようになる．

$$a_{n+1}(m_i) = f_n(m_i) = a_n(m_i) + D_n^a(m_i)$$

この式を，n に関して展開すると，次式が得られる．

$$a_{n+1}(m_i) = a_0(m_i) + \sum_{j=0}^{n} D_j^a(m_i)$$

上式の $a_0(m_i)$ は，タスク 0 におけるメッセージ到着時刻であるが，タスク 0 はパイプラインの先頭のタスクである．したがって，タスク 0 のメッセージは入力装置から得られる．入力装置のメッセージの到着時刻は周期的であることから，$a_0(m_i)$ は次のようになる．

$$a_0(m_i) = a_0(m_0) + i/R^c$$

以上より，$a_{n+1}(m_i)$ は次のようになる．

$$a_{n+1}(m_i) = a_0(m_0) + i/R^c + \sum_{j=0}^{n} D_j^a(m_i)$$

上式から，$a_{n+1}(m_i) - a_{n+1}(m_0)$ を求めると，次のようになる．

$$\begin{aligned} &a_{n+1}(m_i) - a_{n+1}(m_0) \\ &= a_0(m_0) + i/R^c + \sum_{j=0}^{n} D_j^a(m_i) - a_{n+1}(m_0) \end{aligned}$$

上式を，式 (19) に適用すると，次の式が得られる．

$$a_{n+1}(m_0) \geq a_0(m_0) + \sum_{j=0}^{n} D_j^a(m_i)$$

□

4. 提案スケジューリングポリシ

提案スケジューリングポリシは，複数の VBR ストリーム処理において，タスクのスケジュール可能性を高める．以下，本章では提案ポリシについて説明する．

4.1 スケジュール可能性を高める制御

本節では，前章で述べた VBR ストリーム処理モデルを前提として出力装置の時間制約を満たすタスクの制御について考える．

定理 3 により，タスクの初期位相が式 (18) を満たすなら，最悪実行時間より短い時間を CPU 利用率として予約する場合においても，出力装置の時間制約を満たすことができる．しかし，出力装置の時間制約を満たすためには，各タスクに十分な初期位相が必要である．そのため，エンド–エンドの処理遅延時間が大きくなる．ライブによるシステム，たとえば，テレビ会議システムでは，エンド–エンドの処理遅延時間が少ないことが求められる．このことから，十分な初期位相がない場合において，出力装置の時間制約を満たす可能性を高める制御について考える．

定理 3 の式 (18) を見直す．この式の $a_n(m_0)$ に，パ

イプラインモデルにおけるタスク n のメッセージ処理完了時刻 $f_n(m_i)$ とタスク $n+1$ のメッセージ到着時刻 $a_{n+1}(m_i)$ が一致する性質を適用すると，次式が得られる．

$$\begin{aligned}&\textstyle\sum_{j=0}^{n}(D_j^a(m_i)-D_j^a(m_0))\\&\qquad\le (a_0(m_0)-a_0(m_i))+i/R^c\end{aligned}$$

ここで，$(a_0(m_0)-a_0(m_i))$ は，タスク 0 のメッセージ到着時刻の差である．タスク 0 はパイプラインの先頭タスクであるため，メッセージ到着時刻は，入力装置からのメッセージ出力時刻である．したがって，メッセージ到着時刻は周期的であり，$(a_0(m_0)-a_0(m_i))$ は $-i/R^c$ である．このことから，上式は次式となる．

$$\sum_{j=0}^{n}(D_j^a(m_i)-D_j^a(m_0))\le 0 \tag{20}$$

さらに，0 番目のメッセージ到着時刻から i 番目のメッセージ到着時刻までの時間間隔を，メッセージの到着間隔ごとに分割した場合，式 (20) の $D_j^a(m_i)-D_j^a(m_0)$ は，次のように分解できる．

$$\begin{aligned}&D_j^a(m_i)-D_j^a(m_0)\\&\quad=D_j^a(m_i)-D_j^a(m_{i-1})\\&\qquad+D_j^a(m_{i-1})-D_j^a(m_{i-2})\\&\qquad\quad\vdots\\&\qquad+D_j^a(m_2)-D_j^a(m_1)\\&\qquad+D_j^a(m_{1)})-D_j^a(m_0)\end{aligned}$$

したがって，式 (20) は次のようになる．

$$\sum_{j=0}^{n}\sum_{k=0}^{i}(D_j^a(m_{k+1})-D_j^a(m_k))\le 0 \tag{21}$$

式 (21) から分かるように，各タスクにおいて，メッセージ間での処理遅延時間を均等にすると，時間制約が満たされる．しかし，各メッセージの処理遅延時間は，そのメッセージごとに変動するため，単一のタスク内でメッセージ間での処理遅延時間を均等にすることは難しい．そこで，各タスクのメッセージ間での処理遅延時間の差異を，隣接するタスク間で吸収することを考える．まず，式 (21) をタスク j に関して展開すると次のようになる．

$$\begin{aligned}&\textstyle\sum_{j=0}^{n}\sum_{k=0}^{i}\{D_j^a(m_{k+1})-D_j^a(m_k)\}\\&\quad=\textstyle\sum_{k=0}^{i}\{D_0^a(m_{k+1})-D_0^a(m_k)\\&\qquad\qquad+D_1^a(m_{k+1})-D_1^a(m_k)\\&\qquad\qquad+D_2^a(m_{k+1})-D_2^a(m_k)\\&\qquad\qquad\quad\vdots\\&\qquad\qquad+D_{n-1}^a(m_{k+1})-D_{n-1}^a(m_k)\\&\qquad\qquad+D_n^a(m_{k+1})-D_n^a(m_k)\}\\&=\textstyle\sum_{k=0}^{i}\{D_0^a(m_{k+1})-D_1^a(m_k)\\&\qquad\qquad+D_1^a(m_{k+1})-D_2^a(m_k)\\&\qquad\qquad+D_2^a(m_{k+1})-D_3^a(m_k)\\&\qquad\qquad\quad\vdots\\&\qquad\qquad+D_{n-1}^a(m_{k+1})-D_n^a(m_k)\\&\qquad\qquad+D_n^a(m_{k+1})\\&\qquad\qquad-D_0^a(m_k)\}\end{aligned}\tag{22}$$

ここで，タスク n における $k+1$ 番目のメッセージ処理遅延時間 $D_n^a(m_{k+1})$ は，タスク $n+1$ における k 番目のメッセージ処理遅延時間 $D_{n+1}^a(m_k)$ と比較されるべきであるが，タスク $n+1$ は存在しない．しかし，タスク $n+1$ は出力装置として考えられるので，$D_n^a(m_{k+1})$ の比較対象は出力装置における k 番目のメッセージ処理遅延時間 $D_{out}^a(m_k)$ とする．同様に，タスク 0 における k 番目のメッセージ処理遅延時間 $D_0^a(m_k)$ は，入力装置における $k+1$ 番目のメッセージ処理遅延時間 $D_{in}^a(m_{(k+1)}$ と比較する．このことにより，式 (22) は次のようになる．

$$\begin{aligned}&\textstyle\sum_{k=0}^{i}\{\sum_{j=0}^{n-1}(D_j^a(m_{k+1})-D_{j+1}^a(m_k))\\&\qquad+(D_{in}^a(m_{k+1})-D_0^a(m_k))\\&\qquad+(D_n^a(m_{k+1})-D_{out}^a(m_k))\}\le 0\end{aligned}$$

したがって，隣接するタスク間のメッセージ処理遅延時間を均等にすることにより，出力装置の時間制約を満たすことが可能となる．したがって，次のような制御を行う．

- パイプライン上のタスクの実行機会が同じになるように，各タスクの優先度を連動させる．
- メッセージ処理遅延時間は実効未処理メッセージ数 $b_n(t)$ に依存することから，タスク n の i 番目のメッセージの実効到着時刻における $b_n(e_n(m_i))$ と後続タスク $n+1$ の同じ時刻における $b_{n+1}(e_n(m_i))$ を比較する．前者の値が大きい場合は，タスク n に大きな遅延が発生することが予想されるのでタスク n の優先度を高め，処理を早める．また，後者の値が大きい場合は，タスク n の処理遅延は小さいことが予想されるので，優先度を低め，処理を遅らせる．
- メッセージ処理完了後に，実測された処理遅延時間が定理 1 を満たさないならば，処理遅延時間を小さくするため，優先度を高める．
- メッセージ処理完了後に，実測された処理遅延時間が定理 2 を満たさないならば，処理遅延時間を

大きくするため，優先度を低める．

以上により，予約 CPU 利用率に基づき，連続メディア出力装置の時間制約を満たし，かつ，エンド–エンドの処理遅延時間を小さくすることが可能となる．

4.2 適応デッドライン時刻

ストリーム処理するタスクは，航空システムや原子力システムに見られる処理完了時刻に厳密なハードリアルタイムタスクと異なり，その処理完了時刻には許容される遅延幅があるソフトリアルタイムタスクとして考えられる．したがって，デッドライン時刻に許容遅延幅 h_n を持たせる．これにより，タスクの優先度として用いるデッドライン時刻は，式 (10) で求められる時刻に，許容遅延幅内 $[-h_n, h_n]$ で，スケジュール可能性を高める制御による優先度の修正を行った時刻とする．この時刻を適応デッドライン時刻 $d_n^{adap}(m_i)$ と呼ぶ．

4.3 複数のストリーム処理が混在する環境と多重制約

4.1 節では，スケジュール可能性を高める制御について述べた．しかし，この制御は単一のストリーム処理における制御である．マルチメディア環境においては，多地点のテレビ会議システムに見られるように，複数のストリーム処理が混在する場合が多い．すなわち，マルチメディア環境におけるストリーム処理は，パインラインモデルにより接続されたタスク群が複数混在するタスクセットとして考えられる．このようなタスクセットにおいて前節における制御を行うスケジュール問題は，複数のパイプラインのタスク群において前節における制御，すなわち制約を同時に満たすような状態を探し出す多重制約問題である．

提案ポリシは，この多重制約問題を処理するために，認知科学における Parallel Distributed Processing モデル[6]（以降 PDP モデル）を応用する．PDP モデルでは，多くの単純な情報処理ユニットが相互に結合し，それぞれが他のユニットから信号を受け取る．その入力信号が正の値である場合は，ユニットの状態値を高める．負の値である場合は，その状態値を低める．さらに，その状態値に応じた信号を他のユニットに送る．この相互作用をユニットごとに非同期に繰り返すことにより情報処理を行う．各ユニットの相互結合により構成されるネットワークは，多重制約の構造を表している．このネットワークに，何らかの外部条件を与え，各ユニットで非同期に相互作用の繰り返し処理を行う．すると，各ユニットはある状態に落ち着く．その状態がある条件での制約を最大に充足した状態を表す．

以上のことから，複数のパイプラインのタスク群における制約を充足する状態を算出するために，タスク間の依存関係を PDP モデルの相互結合ネットワーク（以降，ネットワーク）に次のように対応付ける．

- 各ユニットの状態値（0 から 1 までの連続値）は，各タスクの重要度とする．タスクの重要度は，最高値（1）を適応デッドライン時刻 $d_n^{adap}(m_i)$ の最短時刻 $d_n(m_i) - h_n$ に，最低値（0）を適応デッドライン時刻の最長時刻 $d_n(m_i) + h_n$ に対応付ける．これにより，重要度から適応デッドライン時刻を算出する．
- ユニット間の結合は，通信するタスク間の場合正の重みを，通信しないタスク間の場合負の重みを持たせる．正の重みを持つタスク間の結合では，ユニットの状態値に比例した正の入力が他のタスクに作用し，相互に近い状態値になる．一方，負の重みを持つタスク間の結合では，ユニットの状態値に比例した負の入力が他のタスクに作用し，互いに遠い状態値になる．これにより，同一パイプライン上のタスク群の重要度が連動し，タスクの実行機会が同じになる．
- 各ユニットに与えられる外部条件は，変動するタスク実行状況に対応してネットワークの動作を修正する力とする．すなわち，外部条件は，各タスクの変動する処理遅延時間に応じて，4.1 節で述べた制御を，次のようにネットワークに作用させる．
  - タスク n の i 番目のメッセージの実効到着時刻における $b_n(e_n(m_i))$ と後続タスク $n+1$ の同じ時刻における $b_{n+1}(e_n(m_i))$ を比較し，前者の値が大きい場合は，正の入力とする．また，後者の値が大きい場合は，負の入力とする．
  - 実測された処理遅延時間が定理 1 を満たさないならば，正の入力とする．
  - 実測された処理遅延時間が定理 2 を満たさないならば，負の入力とする．

以上により，複数のパイプラインのタスク群における制約をネットワークを動作させて処理する．しかし，PDP モデルは，ネットワークを動作させると，その状態は初期値に依存してある制約充足度の高い状態に至り，動作は静止する．一方，ストリーム処理環境において，各タスクの処理遅延時間は，処理するメッセージにより，つねに変化する．ネットワークは，この変化に応じて，1 つの状態に静止することなく，いくつかの制約充足度の高い状態間を移動する必要がある．このため，PDP モデルに熱力学的モデル[3]を加える．熱力学的モデルは，温度による物質の熱揺動（高温で

分子間の結合が弱まり不安定，低温で分子間の結合が強まり安定する性質）を PDP モデルの処理において擬似する．すなわち，高い充足状態から離れた場合は温度を高くし，PDP モデルの処理動作を大きく振動させ新しい状態を見つける可能性を高める．一方，高い充足状態の近傍にある場合は温度を低くし，PDP モデルの処理動作を現在の状態の近傍で小さく振動させ，その状態を維持する．この温度による熱揺動制御により，PDP モデルを各タスクの変動する処理遅延時間に連続的に動作するようにし，定理 1 と 2 を満たし，かつ，タスク間の処理遅延時間を均一にする適応デッドライン時刻を探し続けることを可能とする．

5. 提案ポリシのアルゴリズム

5.1 タスク重要度の更新則

タスク重要度は，PDP モデルに従って，他のタスクからの入力および外部条件としての入力により決める．しかし，提案ポリシでは，その決め方が熱力学的モデルの熱揺動により制御され，確率的に決まる．この変更を PDP モデルの状態更新則に加え，タスクの重要度更新則とする．次式にそれを示す．以降，ユニットをタスクに，状態値を重要度に対応付けて述べる．

$$\alpha_n(t+1) = \alpha_n(t) + \begin{cases} (1-\alpha_n(t))net_n(t) & prob(net_n(t)) \geq R \\ \alpha_n(t)net_n(t) & \text{else} \end{cases}$$

$\alpha_n(t)$ はタスク n の時間 t におけるタスクの重要度（$0 \leq \alpha_n(t) \leq 1$），$net_n(t)$ はタスク n の時間 t における総入力，$prob(x)$ は熱力学モデルの熱揺動による振動を擬似するときに用いる確率関数，R は [0,1] での一様乱数である．

上式の $(1-\alpha_n(t))net_n(t)$ は，タスクの重要度を高める方向へ更新する．一方，$\alpha_n(t)net_n(t)$ は重要度を低める方向へ更新する．どちらの式を適用するかは，$prob(net_n(t)) \geq R$ によって確率的に決める．この確率は，タスクの重要度が 1 に向かう確率を意味する．したがって，タスクの重要度更新則は，熱力学モデルの熱揺動による振動を擬似する確率関数に依存して，重要度を高める方向へまたは低める方向へ更新する．

5.2 タスクへの総入力とタスク間の結合重み

タスクへの総入力は，他のタスクからの入力（他のタスクの出力）の総和と外部条件としての入力により構成される（**図 4** 参照）．タスクへの総入力を以下に示す．

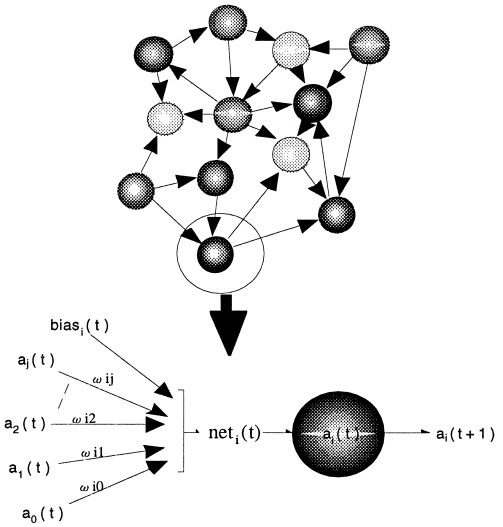


図 4 相互結合ネットワーク
Fig. 4 Inter-connection netowrk.

$$net_n(t) = \sum_{j=0, j\neq n}^{N} \omega_{nj}\alpha_j(t) + \sum_{z=0}^{Z} bias_n^z(t)$$

N はタスク数，Z は外部入力数，ω_{nj} はタスク n とタスク j との結合重み，$bias_n^z(t)$ はタスク n の時間 t における外部入力である（Z は，複数の外部からの入力があることを示す）．

上式から分かるように，他のタスクからの入力は他のタスクの重要度とそのタスク間の結合重みとの積となっている．このため，正の結合重みを持つ，すなわち通信するタスクからはそのタスクの重要度の高さに比例した正の入力が，当該タスクに作用する．一方，負の結合重みを持つ，すなわち通信しないタスクからはそのタスクの重要度の高さに比例した負の入力が当該タスクに作用する．タスクへの総入力が大きな正の値となると，重要度更新則は高い確率でタスクの重要度を高める．その反対の状態では，タスクへの総入力は大きな負の値となり，更新則は高い確率で重要度を低める．すなわち，通信するタスクの重要度は連動して更新される可能性が高い．

5.3 外部条件としての入力

4.1 節で述べたように，変動する処理遅延時間に応じてネットワークの動作を修正するため，外部条件の入力（以降，外部入力）を次のようにする．

$$bias_n^0(t) = \begin{cases} 1-a_n(t) & b_n(e_n(m_i)) \geq b_{n+1}(e_n(m_i)) \\ 0-a_n(t) & b_n(e_n(m_i)) < b_{n+1}(e_n(m_i)) \end{cases}$$

$bias_n^1(t)$

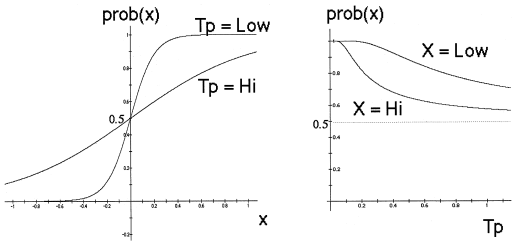


図 5 確率関数と温度
Fig. 5 Probability function and temperature.

$$= \begin{cases} 1 - a_n(t) & b_{n+1}(e_n(m_i))/R^c < D_n^a(m_i) \\ 0 & \text{otherwise} \end{cases}$$

$$bias_n^2(t) = \begin{cases} 0 - a_i(t) & D_n^a(m_i) < \frac{C_n(m_i)}{C_n^{req} R^c} \\ 0 & \text{otherwise} \end{cases}$$

上記から分かるように，$bias_n^0(t)$ は隣接するタスクの処理遅延時間を均一にする入力であり，$bias_n^1(t)$ は定理 1 を満たすための入力である．また，$bias_n^2(t)$ は，定理 2 を満たすための入力である．

5.4 熱揺動の制御

提案ポリシでは，タスクの重要度を高めるかまた低めるかは熱揺動の振動により決まる．このときに使用する確率関数を以下に示す．

$$prob(x) = \frac{1}{1 + \exp(\frac{-x}{Tp})}$$

Tp は温度である．

上の式と**図 5** から分かるように，確率関数は x（タスクへの総入力）が大きいほど 1 になる確率が高くなる．しかし，温度 Tp を高めるとグラフはなだらかになり，入力が正の大きな値であっても確率が 1 より小さな値となる．また，入力が負の大きな値であっても確率が 0 より大きな値となる．したがって，温度 Tp を高くすると更新則が充足度を高める方向だけでなく低める方向にも作用する可能性が高くなる．これにより，様々な状態を動き回り不安定な状態となる．一方，温度 Tp を下げ 0 に近づいた場合，確率関数はほぼステップ関数と同等なり，更新則が充足度を高める方向に作用し，収束へ向かう．

提案ポリシでは，熱揺動の温度 Tp により PDP モデルの動作メカニズムを制御できることに着目する．

(1) 制約充足度の高い状態

PDP モデルにおける制約充足度を次式に示す．

$$G(t) = \sum_{n=0}^{N} \sum_{j=0}^{N} \omega_{nj} \alpha_n(t) \alpha_j(t) + \sum_{n=0}^{N} \sum_{z=0}^{Z} bias_n^z(t) \alpha_n(t)$$

制約充足度 $G(t)$ は，次のようにすると最大となる．すなわち，外部入力 0 の場合，相互に通信するタスク集合のすべてのタスク重要度を 1 に，通信しないタスク集合のすべてのタスク重要度を 0 にする．これは，相互に通信するタスクは重要度を同時に高くして実行を早め，その他のタスクは実行を遅らせることを意味する．このような状態が制約充足度 $G(t)$ を高くする．また，制約充足度の高い状態は，相互に通信するタスク集合ごとに存在する．したがって，パイプラインモデルにより通信するタスクおいては，そのパイプラインごとに制約充足度の高い状態がある．

以上のことから，外部入力が 0 において制約充足度が最大となる状態は，パイプライン上のすべてのタスクの重要度が 1 に，その他のすべてのタスクの重要度が 0 となる状態である．一方，実環境では，処理遅延時間の変動に応じた外部入力がある．そのため，実環境の制約充足度の高い状態は，上記の制約充足度の高い状態と一致しない可能性が高い．しかし，パイプライン上のタスクは同一の処理レートで動作するため，同じ実行機会を与えると，未処理メッセージ数は一定となる．したがって，外部入力が 0 における高充足度状態の近傍に制約充足度の高い状態が存在していると考える．

(2) 温度 Tp の制御

温度は，制約充足度の高い状態からの距離によって制御する．以下にそれを示す．

$$Tp(t+1) = \begin{cases} Tp(t) + r2(T^C - Tp(t)) & D \le D^{Near} \\ Tp(t) + r1(T^C - Tp(t)) & D^{Near} < D < D^{Far} \\ Tp(t) + r3(T^H - Tp(t)) & \text{otherwise} \end{cases}$$

D は，充足度の高い状態と現在のネットワークの状態とのユークリッド距離（以降，距離）において最小となる値とし，D^{Near} および D^{Far} はネットワーク状態間でとりうる最大距離に対する比率から決定する．ネットワークの状態間の距離は，ある状態におけるタスクの重要度と他の状態におけるタスクの重要度の差が，すべてのタスクにおいて 1 である場合に最大なる．これをネットワーク状態間の最大距離とし，その値はタスク数の平方根値となる．また，$Tp(t)$ は時間 t の温度，T^C は下限温度，T^H は上限温度，$r1$，$r2$ および $r3$ は制約充足度の高い状態からの距離範囲に応じた温度の変化量を決定するゲイン係数である．

上の式から分かるように，タスク間の相互結合ネットワークの状態が制約充足度の高い状態に近い場合は温度を低くする．その結果，確率関数は 1 に近い値を出力し，タスクの重要度は制約充足度の高い状態へ近づく確率が高くなる．しかし，制約充足度の高い状態へ近づきすぎると，温度が小さな値となり，確率関数はほぼ 1 を出力し，タスクの重要度は制約充足度の高い状態に固定化する．想定環境は，動的な環境であるので，制約充足度の高い状態は時間とともに変わる．そこで下限温度により，温度を下げ止まりさせ，その近傍で小さく振動させる．これにより，適度な制約充足と外部変化に敏感に反応する動作を実現する．一方，遠い場合は温度を高くし，大きく振動させ充足度の高い状態を発見する可能性を高める．これらの制御により，タスクの重要度の初期値に依存することなく，外部入力の変化に応じて複数の制約充足度の高い状態の間を移動する．

5.5 適応デッドライン時刻の算出

タスクの重要度更新則は，タスクにおけるメッセージの実効到着時刻 $e_n(m_i)$ に各タスクごとに単独に適用する．これにより求められたタスクの重要度 $\alpha_n(t)$ から，次式のようにデッドライン時刻を修正する時間を算出する．

$$v_n(m_i) = h_n - 2h_n\alpha_n(e_n(m_i))$$

ただし，h_n はタスク n における処理遅延許容幅，$v_n(m_i)$ はタスク n における i 番目のメッセージ処理におけるデッドライン時刻を修正する時間（$-h_n \leq v_n(m_i) \leq h_n$）である．

この $v_n(m_i)$ から，タスク n における i 番目のメッセージ処理の適応デッドライン時刻 $d_n^{adap}(m_i)$ を次のように求める．

$$d_n^{adap}(m_i) = d_n(m_i) + v_n(m_i)$$

上記により求められた適応デッドライン時刻に基づいて，タスクの実行順を EDF により決める．

6. 性能評価

提案ポリシを，我々が分散 OS の構成法の研究のため開発している分散 OS Solelc [10] 上に実装し，性能評価を行った．本章では，その結果について述べる．

6.1 性能評価環境

ハードウェアの環境としては，次のものを使用した．

- 使用機種　エプソン社製　Endeavor ATX-7000
- プロセッサ　Pentium-S 200 MHz
- キャッシュ　512 KB
- 主記憶　128 MB

表 1 評価タスクセットの時間属性
Table 1 Timing attributes for example taskset.

	$1/R^c$	h	C^{max}	C^{min}	C^{ave}
task1	100 ms	10 ms	12 ms	2 ms	7 ms
task2	100 ms	10 ms	28 ms	2 ms	15 ms
task3	100 ms	10 ms	13 ms	3 ms	8 ms
task4	70 ms	10 ms	8 ms	2 ms	5 ms
task5	70 ms	10 ms	20 ms	2 ms	11 ms
task6	70 ms	10 ms	8 ms	2 ms	5 ms
task7	40 ms	10 ms	5 ms	1 ms	3 ms
task8	40 ms	10 ms	10 ms	2 ms	6 ms
task9	40 ms	10 ms	5 ms	1 ms	3 ms
task10	30 ms	–	–	–	–

h：遅延許容幅　C^{max}：最大実行時間
C^{min}: 最小実行時間　C^{ave}：平均実行時間

今回の評価では，Solelc のディスパッチ機能とメッセージ通信機能のみを使用し，スケジューリング分解能は 1 msec とした．また，コンパイラは gcc（version egcs-2.91.60）を使用した．

6.2 評価方法と評価タスクセット

図 1 に基づき，パイプラインは次のように構成する．

- 周期的にメッセージを生成する入力装置
- 周期的にメッセージを消費する出力装置
- メッセージごとに処理時間が変動する入力サーバタスク，出力サーバタスク，アプリケーションタスク

提案ポリシは複数のストリーム処理が混在することを想定していることから，評価タスクセットを，上記のパイプライン処理を行うタスクの組を 3 組（**表 1** の task1〜9）と，ランダムな負荷を生成する周期タスク（表 1 の task10）より構成する．

評価方法は，周期タスクモデルによる方式，LBAP による方式および提案ポリシの 3 つの方式を，上記タスクセットの負荷タスクとパイプライン上のタスクのスケジューリング成功率で比較評価する．また，各タスクの時間属性を表 1 に示す．ただし，タスクの実行時間は，メッセージの処理ごとに，最大実行時間と最小実行時間の間をランダムに変動する時間とし，スケジューラには未知とする．

タスク重要度更新則のパラメータに関しては，ネットワークの結合重みは通信関係がある場合は 5.0，通信関係がない場合は −5.0 とした．温度制御パラメータにおいて，D^{Near} および D^{Far} はそれぞれネットワーク状態間の最大距離 × 0.2，ネットワーク状態間の最大距離 × 0.5 とし，$r1$，$r2$ および $r3$ はそれぞれ 0.5，0.9，0.9 とした．

6.3 スケジューリング成功率

評価タスクセットに関して，負荷状況，初期位相状

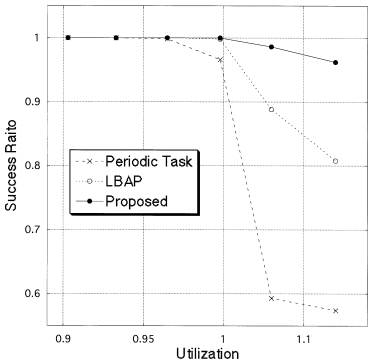


図 6 負荷状況に応じたスケジューリング成功率
Fig. 6 Scheduling success raito by utilization.

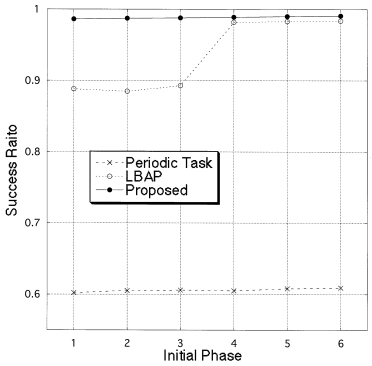


図 7 初期位相状況に応じたスケジューリング成功率
Fig. 7 Scheduling success raito by initial phase.

況を変動させて，スケジューリング成功率を前述の 3 つのスケジューリング方式について比較評価した．スケジューリング成功率を，次のように定義する．また，評価期間は 180,000 msec とした．

$$SuccessRaito = \frac{Success(Load)+Success(Msg)}{Finish(Load)+Finish(Msg)}$$

$Success(Load)$ はデッドライン時刻までに処理を完了した負荷周期タスク数，$Success(Msg)$ は出力装置の処理開始時刻までに出力装置に到着したメッセージ数，$Finish(Load)$ は処理完了した負荷周期タスク数，$Finish(Msg)$ は入力装置から生成され出力装置に到着したメッセージ数である．

負荷状況に応じたスケジューリング成功率を**図 6** に示す．負荷は，すべての実行タスクにおける周期（$1/R^c$）に占める平均実行時間の割合の総和とした．また，評価時の負荷量は，負荷周期タスクの最大実行時間と最小実行時間により調整した．各タスク間の初期位相は，各タスクの 1 周期分の時間とした．さらに，LBAP による方式の最大処理遅延時間は初期位相と同じ時間とし，その最大処理遅延時間によりデッドライン時刻を設定した．

図 6 から分かるように，提案ポリシは他の方式よりスケジューリング成功率がつねに高い．また，過負荷の状況においても，高いスケジューリング成功率となる．これは，タスク間の初期位相により発生した未処理メッセージ数をタスク間で均一にするメカニズムが，各タスクの処理遅延時間を均一とすることで，タスクの負荷を分散させていると考えられる．一方，他の方式は，負荷が高まると急激にスケジューリング成功率が低下する．このことからも，未処理メッセージ数をタスク間で均一にするメカニズムが高負荷時にスケジュール可能性を高めるのに有効であることが分かる．

表 2 ストリーム処理と周期タスクの成功率
Table 2 Success raito for stream processing and periodic task.

初期位相	Periodic		LBAP		Proposed	
	load	str	load	str	load	str
$1/R^c$	0.97	0.02	1.0	0.67	0.98	1.0
$2/R^c$	0.97	0.02	1.0	0.66	0.98	1.0
$3/R^c$	0.97	0.03	0.98	0.73	0.98	1.0
$4/R^c$	0.97	0.02	0.97	1.0	0.98	1.0
$5/R^c$	0.97	0.03	0.97	1.0	0.98	1.0

load：負荷周期タスクのスケジューリング成功率
str：ストリーム処理のスケジューリング成功率

次に，初期位相に応じたスケジューリング成功率を**図 7** に示す．負荷は 1.03 とし，初期位相は各タスクの周期の倍数で設定した．すなわち，各タスクを，パイプライン上おいて隣接する前方タスクの最初の到着時刻から周期の倍数分の時間経過後に，起動した．

図 7 から分かるように，提案ポリシと LBAP は初期位相を大きくすると，スケジューリング成功率が改善されるが，周期タスクモデルによる方式は，スケジューリング成功率が改善されない．これは，周期タスクモデルによる方式は，大きな初期位相により発生した多くの未処理メッセージ数を考慮していないため，隣接するタスクがメッセージを必要とする時刻までに余裕があるにもかかわらず，固定のレートで処理を行うことによる．すなわち，余裕があるにもかかわらず，固定のレートで処理を行うため，負荷周期タスクの CPU 利用率が増えない．したがって，負荷周期タスクのスケジューリング成功率が改善されず（**表 2** 参照），そのため，全体としてのスケジューリング成功率も改善されない．一方，LBAP による方式と提案ポリシは，スケジューリング成功率が改善されるが，両者を比較すると，LBAP による方式はスケジューリン

グ成功率を改善するためには大きな初期位相が必要である．これは，LBAP による方式は，固定の処理遅延時間に基づいてデッドライン時刻（優先度）を決めるため，固定の処理遅延時間を超える未処理メッセージ数が発生する場合，周期タスクモデルと同様の現象に陥ることによる（表 2 参照）．提案ポリシは，変動する処理遅延時間に適時対応してデッドライン時刻（優先度）を決めるため，上記の現象の発生が抑えられると考える．

以上のことから，提案ポリシは，従来の方式と比較して，次のような特性を持つ．

- スケジューリング可能性が高く，特に高負荷時にその差が著しい．このことから，CPU 利用率を高めることが可能である．
- 少ない初期位相でスケジューリング成功率が改善される．このことから，少ないエンド–エンド処理遅延時間でスケジューリングすることが可能である．

6.4 スケジューリングオーバヘッド

提案ポリシの計算量は，PDP モデル更新則の計算量に依存する．PDP モデルの更新則は 5.1 節の式，5.2 節の式および図 4 から分かるように，他のタスクの重要度と結合重みの積の総和とバイアスの総和によって求められる．したがって，更新則の計算量はタスク数とバイアス数に比例して増加する．一方，提案ポリシではバイアス数は一定（3 つ）であり，タスク数に応じて結合重みと他のタスク数の積が求められることから，ほぼタスク数に比例して計算量が増加することが予測できる．スケジューリングオーバヘッドの実測評価結果を**図 8** に示す．この図から分かるように，タスク総数に応じて，直線的にオーバヘッドは高くなる．以上のことから，スケジューリングオーバヘッドはタスク数に比例すると考えられる．プロセッサが Pentium-S 200 MHz を用いた場合，タスク総数 10 で平均約 72 μsec である．現在の一般的なプロセッサのクロックは評価環境の数倍である．このことから，一般的な PC は，評価環境の 2 倍のクロックを持つと仮定すると，10 個のタスクのスケジューリングで約 36 μsec の計算時間が必要となる．ただし，この負荷は，適応デッドデッドラインを計算するタスクの到着時のみに必要であり，それ以外のコンテキストスイッチ時には不要である．一方，マルチメディア環境で扱う音声，アニメーションおよびビデオは，文献 11) によると最短周期が 75 Hz，すなわち約 13 msec となっている．この最短周期のタスクを 10 個スケジュールする場合，一般的な PC 環境において 1 周期の適応

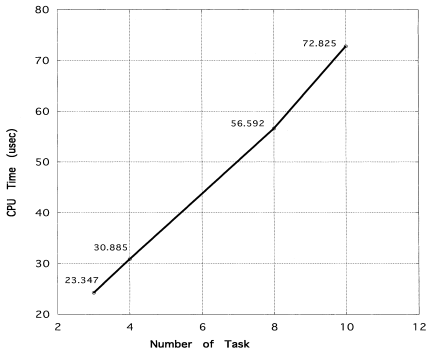


図 8 スケジューリングオーバーヘッド
Fig. 8 Scheduling overhead.

デッドライン計算に必要となる時間は，上記で示した約 36 μsec である．この時間のタスクの周期に占める割合は，たかだか 0.28%にすぎない．また，タスクを 100 個スケジュールする場合でも，2.8%ほどである．

以上のことから，提案ポリシはタスク総数に比例してオーバヘッドは高くなるが，タスクのスケジューリング周期に占める割合は極微量であるといえる．したがって，提案ポリシは，連続メディアをストリーム処理するタスクのスケジューリングに十分に適用できると考えられる．

7. おわりに

本論文では，VBR ストリーム処理タスクのスケジューリング方式において，高い CPU 利用率環境でも，連続メディア出力装置の時間制約を満たし，かつ，エンド–エンドの処理遅延時間を小さくするポリシを提案した．さらに，提案ポリシの性能評価を行い，その有効性を示した．また，スケジューリングオーバヘッドが微量であることも明らかにした．

参考文献

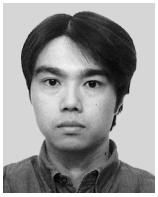
1) Sha, L., Rajikumar, R. and Lehoczky, J.P.: Priority Inheritance Protocols: An Approach to Real-Time Synchronization, *IEEE Trans. Comput.*, Vol.39, No.9, pp.1175–1185 (1990).
2) Liu, C.L. and Layland, J.W.: Scheduling algorithms for multiprograming in a hard real time environment, *J. ACM*, Vol.20, No.1, pp.46–61 (1973).
3) Yavatkar, R. and Lakshman, K.: Optimization by Simulated Anneling, *Science*, Vol.220, pp.671–680 (1983).
4) Tokuda, H., Nakajima, T. and Rao, P.: Real-time mach: Towards a predictable real-time system, *Proc. USENIX Mach Workshop*, pp.1–

10 (1990).
5) Steinmetz, R. and Nahrstedt, K.: *Multimedia: Multimedia Applications*, pp.709–765, Prentice Hall (1995).
6) Rumelhart, D.E., McCelland, J.L. and the PDP Research Group: *PARALLEL DISTRIBUTED PROCESSING*, The MIT Press (1986).
7) LeGall, D.: A video compression standard for multimedia applications., *Comm. ACM*, No.34, pp.47–58 (1991).
8) Anderson, D.P.: Metascheduling for continous media, *Trans. Comput. Syst. ACM*, No.11, pp.226–252 (1993).
9) Govindan, R. and Anderson, D.P.: Scheduling and IPC mechanisms for continuous media, *Proc. 13th ACM on Operating Systems Principles*, pp.68–80 (1991).
10) 芝 公仁，大久保英嗣：分散オペレーティングシステム Solelc の構成，情報処理学会研究会報告，2000-OS-84, pp.237–244 (2000).
11) Furth, B.: Multimedia Systems: An Overview, *IEEE Multimedia*, Vol.1, No.1, pp.47–59 (1994).

(平成 13 年 2 月 15 日受付)
(平成 13 年 4 月 4 日再受付)
(平成 13 年 5 月 9 日採録)

滝沢 泰久（正会員）

昭和 34 年生．昭和 58 年京都工芸繊維大学工芸学部機械学科卒業．同年日本ユニシス（株）入社．平成 2 年住友金属工業（株）入社．平成 10 年より（株）ATR 環境適応通信研究所に出向．現在，実時間オペレーティングシステム等の研究に従事．



芝 公仁（学生会員）

昭和 49 年生．平成 11 年立命館大学大学院理工学研究科博士課程前期課程修了．現在，同大学院理工学研究科博士課程後期課程に在学中．オペレーティングシステム，分散システム，実時間システム等に興味を持つ．

大久保英嗣（正会員）

昭和 26 年生．昭和 52 年北海道大学大学院工学研究科情報工学専攻修士課程修了．同年（株）日立製作所ソフトウエア工場入所．主として FORTRAN コンパイラの開発に従事．昭和 54 年京都大学工学部情報工学科助手．昭和 60 年同講師．昭和 62 年同助教授．平成 3 年立命館大学理工学部情報学科教授となり，現在に至る．工学博士．オペレーティングシステム，データベースシステム，分散システム，実時間システム等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，システム制御情報学会，ACM，IEEE-CS 各会員．