社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－HPC－118（15）
2008／12／17

プロセス情報を利用したリアルタイム計算機負荷予測手法

小林道治† 菅谷至寛† 阿曽弘具†

複数の非均質な計算機から構成される分散処理環境において，負荷分散やタスクスケジューリングは効率的なリソース利用のために欠かせない．しかし，分散処理環境では負荷変動が激しいため，アプリケーションの実行時間が一定とならず，最適な負荷分散やタスクスケジューリングを行うことは困難である．各計算機でリアルタイムに計算機負荷予測を行うことが出来れば，より効率的な負荷分散が期待できる．本研究では，プロセス情報を利用した計算機負荷予測手法のリアルタイムシステムへの適用方法を検討し，システムへの実装を行った．その性能を計算機実験により検証した．

Real-Time Long-Term CPU Load Prediction using Process Properties

Michiharu Kobayashi,† Yoshihiro Sugaya† and Hirotomo Aso†

In distributed processing environments(DPE) composed of many heterogeneous computers, load balancing and task shceduling is important for efficient use of resources. But load balancing and task shceduling are difficult in distributed processing enviroments, because load changes frequently by various users. If we can predict CPU load over the long term, load balancing and task scheduling become more efficient. We propose to apply long-term CPU load prediction with references of process properties to real-time system. And the performance is verified by experiments.

1. はじめに

近年，複数のコンピュータやプロセッサを使う並列分散処理環境の利用が拡大している．非均質な計算機が多数集まって構成される分散処理環境で，その性能を効率的に利用するためには，自動的かつ効率的にプロセスを配置する負荷分散システムや，タスクスケジューリングが重要である．しかしながら，複数ユーザが計算機資源を共有する環境では,CPU の負荷変動が大きくプロセスの実行時間が一定でないため，現在の負荷情報のみを頼りに負荷分散やタスクスケジューリングを行うことは困難である．リアルタイムに CPU 負荷予測や CPU 時間予測が可能なシステムが計算機上にあれば，より効果的な負荷分散やタスクスケジューリングを実現させる上で，有効であると考えられる．

関連研究として，R.Wolski らの開発した NWS[1)]，小出らの開発した RIS[2)] がある．NWS では，複数の予測手法のパラメータを動的に変化させながら，適切な予測手法を選択して負荷予測を行う．RIS では CPU 負荷予測手法として，予測時刻以前の負荷変動と類似したパターンを過去の負荷時系列から検索し，今後の負荷変動は過去最も類似したパターンのその後の変動と同様であると仮定して予測を行う．これらの手法における長期的予測の精度は 負荷分散などの指標として使うには十分とは言えない．

本研究ではプロセス情報を用いたリアルタイム計算機負荷予測システムを構築する．本研究では，特定の人が似たようなタスクを何回も投入する環境を想定する．このような環境下では，過去の負荷時系列情報以外に，現在実行中のプロセス情報も有力な情報となる．そこで，予測手法として，プロセス検索法[3)]，実行時間予測法[4)] の二つを基礎としてリアルタイム予測が可能となるように改良を行い，システムの実装を行った．これらの予測手法は他の従来手法と比較して，過去の負荷時系列だけでなく現在実行中のプロセス情報を利用することで，負荷が大きく変動する非定常時での長期負荷予測を高精度かつ高速に行うことができる．この特徴からは，両手法は想定環境下でリアルタイム予測を行うための手法としては有効なものであると言える．

リアルタイム CPU 負荷予測システム（以下本システムと呼ぶ）では，定期的に負荷，およびプロセス情報を収集し，ユーザ側からの要求があった際に収集した情報を使って予測を行う．ユーザ側はシステムが稼動しているマシンに TCP 通信を行うことで，任意に両手法の予測結果を取得することができる．

本稿の構成は以下の通りである．2 章では本研究で扱う負荷予測手法について述べる．3 章でリアルタイ

† 東北大学 大学院 工学研究科 電気・通信工学専攻
Department of Electrical and Communication Engineering, Graduate School of Engineering, Tohoku University

ム計算機予測手法のための予測手法の適応やデータ構造を提案する．4章では実験とその結果を述べ，最後に5章でまとめと今後の課題を述べる．

2. 予測手法

本論文では負荷の指標として load average を用いる．load average はプロセスの計算時間と強い相関があり，従来の多くの研究で，負荷の指標として用いられている．

2.1 定　　義

負荷をサンプリングした n 番目の実時刻を t_n，時刻 t_n での負荷を V_n とする．サンプリング間隔は d とする．予測開始時刻 (現在時刻) を T，時刻 T の負荷を V_T とし，時刻 $T+d, T+2d, \ldots, T+pd$ の負荷を予測する，時刻 $T+id$ での負荷予測値を P_i とする．p を予測窓幅と呼ぶ．また，各時刻 t_n で実行されているプロセスの集合を，プロセス状態と呼び，R_n と表す．

2.2 プロセス検索法

プロセス検索法では，予測時に負荷時系列 $\{V_n\}$ のみでなく，実行中のプロセス情報を用いる．過去に同一のプロセスが実行されていた場合，負荷変動も類似すると考えられるので，現在実行中のプロセス名と一致するプロセスが実行された時刻 t_n を検索する．一致した時刻 t_n の中で直前の負荷変動が類似するものを抽出し，時刻 t_n 以降の負荷変動 $(V_{n+1}, V_{n+2}, \ldots)$ も類似するものと考え予測値として利用する．

2.2.1 一次階差時系列の生成

負荷時系列 $\{V_n\}$ から，一次階差 $V'_n(=V_n - V_{n-1})$ を作成し，時刻 t_n の一次階差時系列の窓 $W_n = (V'_{n-D+1}, \ldots, V'_n)$ を設定する．その i 番目を $W_n(i) = V'_{n-i+1}$ と記す．D は窓幅を示すパラメータである．予測開始時刻 T における窓は W_T と表せる．

2.2.2 プロセス情報の検索

時刻 T で実行されているプロセス集合と同一のプロセスが実行されていた時刻を検索する．同一プロセスが実行されていた時刻があれば，それらの時刻の集合のみを処理対象とする．完全一致する時刻がなければ，一部のプロセスのみが一致する時刻があるときは，その時刻の集合を，全く同一プロセスがなければ，負荷時系列全体を処理対象とする．

2.2.3 非類似度の計算と仮予測値の統合

処理対象である各時刻 t_n における窓の非類似度 $err(n) = \sum_{i=1}^{D} |W_n(i) - W_T(i)|$ を求める．$err(n)$ の小さい順に K 個の時刻を取り出し，窓 W_n による仮予測結果を $P_k^n = V_T + \sum_{j=1}^{k} V'_{n+j} (k = 1, \ldots, p)$ として定義する．K 個の仮予測結果を平均したもの

$$P_k = \frac{1}{K} \sum_{n \in S} P_k^n \qquad (1)$$

を最終的な予測結果として出力する．S は仮予測結果を定義した時刻の位置の集合である．

表 1　CPU 実行時間予測手法
Table 1　CPU Executing Time predicton method

予測手法	使用するプロセス情報
Normal	ユーザ名，プロセス名
Args	ユーザ名，プロセス名，引数
Mem	ユーザ名，プロセス名，メモリ使用量
Mem+Args	ユーザ名，プロセス名，引数，メモリ使用量

2.3 実行時間予測法

実行時間予測法は，過去の類似プロセスの実行時間から実行中プロセスの CPU 実行時間を予測し，その結果を用いて将来の負荷変動を求める．

類似性を判定するためのプロセス情報は，ユーザー名，プロセス名，経過実実行時間，経過 CPU 時間，使用メモリ量，引数で，サンプリング間隔を d_{time} とする．

2.3.1 CPU 実行時間の仮予測

予測時点から過去一定期間内に実行されたプロセスを検索し，指定したプロセス情報を基に現在実行中のプロセスに類似したプロセスを選択する．ただし，その CPU 実行時間が実行中プロセスの経過 CPU 実行時間以下となっているものについては選択しない．検索時に使用するプロセス情報の指定方法により4手法を考える．各手法で使用するプロセス情報を**表 1** に示す．各手法での仮予測値算出法を以下に示す．

2.3.1.1 Normal, Args での仮予測値算出

Normal ではユーザ名とプロセス名，Args ではさらに引数を考えて，それらが一致した場合に類似プロセスとして選択する．

選択した類似プロセス群を実行開始時刻順に並べ，CPU 時間のクラス間分散が最大となるところで2分割をする．分割された2集合のうち実行開始時刻が新しい側を選択し，その CPU 実行時間平均値を仮予測値とする．

2.3.1.2 Mem, Mem + Args 時の仮予測値算出

Mem ではユーザ名とプロセス名，Mem+Args ではさらに引数が一致するプロセスを類似候補として検出する．この候補プロセスと実行中プロセスとの仮想メモリ使用量を予測点 T から過去5状態に渡って比較し，累積誤差 $CumError$ を計算する．この累積誤差が実行中プロセスの窓内仮想メモリ 10%以内のものを，類似プロセスとして扱う．

$$CumError = \sum_{k=0}^{D-1} |M(T-k) - M'_{close}| \qquad (2)$$

ただし，D は窓の範囲，$M(T-k)$ は実行中プロセスの予測点 T から過去 k 状態目の仮想メモリ使用量，M'_{close} は候補プロセスの対応する状態における仮想メモリ使用量である．この累積誤差が実行中プロセスの窓内実メモリ使用量合計値の 10%以内ならば，類似プロセスとして選択する．

選択した類似プロセス群について累積誤差が小さい

順に並びかえを行なう．CPU 実行時間のクラス間分散が最大となるところで，集合を 2 分割し，累積誤差が小さい側の集合の CPU 実行時間平均値を仮予測値とする．

2.3.2 信頼度による仮予測結果の選択

2.3.1 で求めた各仮予測に対して t 分布を用いて信頼度を算出する．そのうち信頼度が最大の仮予測結果をを最終的な CPU 実行時間予測結果とする．

取得した類似プロセス集合と仮予測値に対して t 分布を用い，各仮予測の信頼度 $C(p)$ を設定する．信頼度は，次式で与えられる．

$$C(p) = 2\int_0^{\frac{P\cdot 0.1}{\sqrt{S/n}}} f(t, n-1)\cdot dt \qquad (3)$$

P は算出した仮予測値，S は取得した類似プロセス群における CPU 実行時間の不偏分散，n は取得した類似プロセス数，$f(t,k)$ は自由度 k における t 分布の確率密度関数である．取得した類似プロセス数が多いほど，またその CPU 実行時間がまとまっているほど信頼度は高くなる．

2.3.3 実実行時間の推定と負荷予測結果

CPU 利用率から各タスクによって生じる推定負荷 $X.eload$ を求める．

$$X.eload = \begin{cases} 1\cdots if\ CPU利用率 \geq M\% \\ 0\cdots otherwise \end{cases} \qquad (4)$$

ここで M はあらかじめ与えるパラメータである．次に，タスクの CPU 実行時間予測値と推定負荷を用いて負荷を予測する．予測した残りの CPU 実行時間 $X.rt$ だけ負荷 $X.eload$ が発生すると仮定し，各時刻におけるタスク X の負荷 $X.load(t)(t = T+1,\ldots,T+p)$ を式 (5) で予測する．

$$X.load(t) = \begin{cases} X.eload & (t \leq T + X.rt) \\ 0 & (t > T + X.rt) \end{cases} \qquad (5)$$

これを全ての計算タスクについて加算して，予測期間中の負荷値 $P(t) = \sum_X^{全てのタスク} X.load(t)$ を求める．

各タスクの実実行時間 ($X.rt$ は 1 つのタスクだけが動いている時の実行時間) は，負荷に応じてほぼ線形に増加するので，次のように負荷予測値を補正する．$P(t) = a$ となる予測区間の長さを I_a とおき，

$$I'_a = \begin{cases} I_a * \frac{a}{CPU数} \cdots if\ a \geq CPU数 \\ I_a \qquad \cdots otherwise \end{cases} \qquad (6)$$

と定める．$P(t) = a$ となる区間長を I'_a に延長する．延長後の負荷予測値 $P'(t)$ を，実行時間予測法の予測結果とする．

3. 負荷予測システムの構築

本システムの構造として，**図 1** のようにクライアント・サーバーモデルを採用した．本システムは負荷予測スレッドと情報収集スレッドから構築される．

負荷予測スレッドでは，クライアント側であるユーザーから TCP 通信でコマンドを受け取り，そのコマンドに応じて予測手法を選択する．情報収集スレッドで収集した情報を使って予測を行い，予測結果をユーザー側へと送信する．

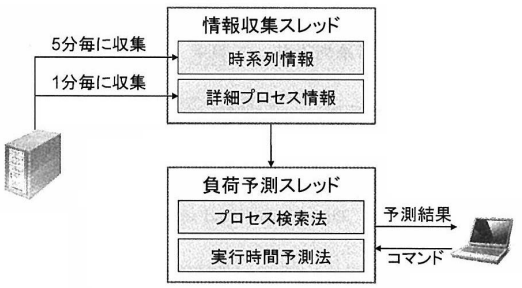

図 1 リアルタイムシステムのモデル
Fig. 1 The model of real-time system

```
struct process_data_t{
  char* Argument;
  long* memory_usage;
  long* cpu_exec_time;
};
```

図 2 詳細プロセス情報
Fig. 2 The detailed process infomation

情報収集スレッドでは次のようにして情報収集を行う．マシンから d 分毎に負荷 V_n とそのときのプロセス状態 R_n を時系列情報として保存する．詳細なプロセス情報 (**図 2**) は，$d/m(= d_{time})$ 分毎に収集を行い，新しく実行されたプロセスは引数，CPU 実行時間，メモリ使用量を現在実行中のプロセスとして保存し，1 つ前以前から実行中のプロセスについては，CPU 実行時間，メモリ使用量を追加更新を行い，終了したプロセスは過去のプロセス情報として保存する．実験では，$d = 5$, $m = 5$ とした．

3.1 プロセス検索法の改良

3.1.1 リアルタイム予測への適応

本システムでは予測手法として，プロセス検索法と実行時間予測法の二つを扱う．リアルタイム負荷予測システムでは，ユーザから予測要求を受けるとすぐに負荷予測を行い，その結果を返すことが求められる．実行時間予測法は，どの時刻でも予測を行うことができる．しかし，プロセス検索法では次に説明する理由から，このままの状態ではリアルタイム予測を行うことができない．そのためプロセス検索法の改良が必要となる．

従来のプロセス検索法では，予測時刻 T が負荷をサンプリングする実時刻に一致するときでなければ窓 W_T を作成できない．T の直前のサンプリング時刻を t_{last} とすると，$T \neq t_{last}$，すなわち $T = t_{last} + a(0 < a < d)$ のときは W_T を作成できない．そこで，予測システム

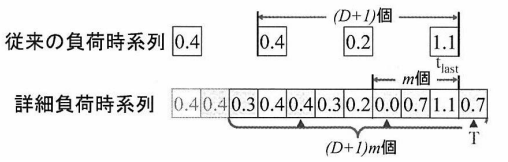

図 3 詳細負荷時系列
Fig. 3 The detailed load time series data

では, 時刻 T がサンプリング時刻でないときは, 窓 W_T の代わりとして, 窓 W' を作成する. 窓 W' の作成方法として, 最新負荷法と平行移動法を提案する.

最新負荷法では, サンプリングした最新の負荷 D 個に時刻 T の負荷 V_T を加えて, この $D+1$ 個の負荷の一次階差をとることで窓を作成する.

平行移動法では, あらかじめ, サンプリング間隔 d を m 等分した時刻でも負荷を計測しておく (m は適切に設定する). その負荷は現在時刻 T の過去 $(D+1)d$ の時間分 $(D+1)m$ 個だけ保存する. これを詳細負荷時系列と呼ぶ. 予測開始時刻 T に対して時刻 $T-dD, T-d(D-1), \ldots, T-d, T$ のそれぞれの時刻に最も近い時刻の負荷を詳細負荷時系列から選び, その負荷時系列から一次階差をとって窓 W' を作成する. **図 3** は, $D=2$, $m=3$ の時の負荷時系列と, 詳細負荷時系列を表している.

この両手法の特徴は次のようになる. 最新負荷法では, 単純な操作でリアルタイム予測を行うことができるが, 負荷 V_T と V_{last} のサンプリング間隔が d と等しくないという欠点がある. 一方, 平行移動法ではサンプリング間隔は一定であるが, 最新負荷法よりも $(D+1)m$ 個だけ, 負荷値を保存する領域が必要となる.

3.1.2 プロセス検索法の窓幅動的決定法

プロセス検索法では, 予め窓幅 W を決める必要がある. これはシステムを起動するマシンのスペック, 投入されるタスクに対して固有の値をとるため, マシン毎に予備実験などによって決める必要が出てくる. このため, 数百台といった規模で稼動するクラスタに導入する場合には非常に大きな手間となる. さらにプロセス状態や負荷変動によっては, 決定した窓幅が常に最適であるとは限らない. そこで本論文では, 予測時にプロセス状態に合わせて決められた最大窓幅 D_{max} 以下で予測を行う窓幅の動的決定法を, プロセス検索法に導入した.

窓幅の動的決定法の流れは次のとおりである. まず, 現在時刻 T からさかのぼっていき, 現在時刻のプロセス状態 R_n がどれだけ続いているのかを求める. 現在時刻のプロセス状態の開始時刻が t_k である時, 予測に使う窓幅 D は時刻 t_k から時刻 T までに存在する負荷の個数, すなわち $last-(k-1)$ とする. **図 4** では, 現在時刻 T より三つ前から同じプロセス状態であるので, このときの窓幅は $D=4$ となる.

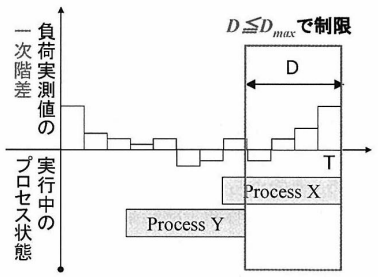

図 4 プロセス検索法の窓幅動的決定法
Fig. 4 The automatic determination of window size for Process search method

ただし $D > D_{max}$ のときは, 最大窓幅 D_{max} を超えることになるので, $D = D_{max}$ とする. 最大窓幅の制限により, 予測のための計算時間の増加を防ぐことができる.

また, 現在時刻のプロセス状態 $R_n = \phi$ であるときは, 実行中のプロセスがない状態ということになり, このときは短期間の負荷を参照するのみで十分と考えられる. そこで現在時刻のプロセス状態 $R_n = \phi$ のときは, $D=3$ とする.

3.2 データ構造

本システムで使うデータ構造として, 二つのデータ構造を用いる. 本システムでは時系列情報, 詳細プロセス情報の 2 つのデータ構造を持つ. リアルタイムシステムとして常駐するので, 使用メモリの節約や予測の高速化を行うために, 負荷時系列に対しての圧縮や, 予測に必要な情報を高速に取り出せるようにする必要がある.

プロセス検索法では負荷時系列 $\{V_n\}$ とプロセス状態時系列 $\{R_n\}$ を使用して予測を行う. 負荷時系列 $\{V_n\}$ とプロセス状態時系列 $\{R_n\}$ はサンプリング時刻 t_{n+1} で, 時系列の最後に, 計算機から計測した負荷 V_{n+1}, プロセス状態 R_{n+1} を追加する. 削除を行う場合は逆に, 時系列の先頭のデータを削除する. これらの点から, 時系列情報は先頭データの削除, 最後尾へのデータの追加を容易に行えるデータ構造がよい.

実行時間予測法では, 現在実行中のプロセスの CPU 実行時間を求めるために, 過去に実行されたプロセスでユーザ名, プロセス名の一致するプロセスを検索する. 詳細なプロセス情報を保存するデータ構造が, ユーザ名, プロセス名の一致するプロセスを簡単に取り出せる構造であれば, 実行時間予測法をより高速に行うことが出来る.

3.2.1 時系列情報

時系列情報では, 各時刻 t_n における負荷とそのプロセス状態を保存する. 負荷時系列に対して Packbits を基にした圧縮を行い, 各プロセス状態の終了時刻に対応する負荷がどの位置あるのかを保存する.

負荷時系列は, 連続した同一データを, データ一つ

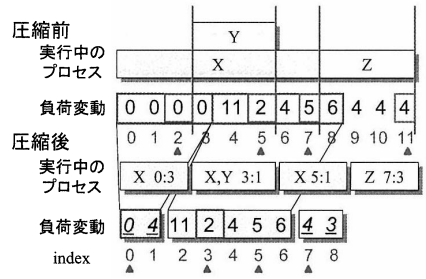

図 5 時系列情報のデータ構造
Fig. 5 The structure of time series data

分と連続した長さで表現し, 非連続データはそのままの形とするものである. 圧縮した連続部分にはフラグビットを与えておく. 例えば [4,4,4,4,4,2,3,1,1,1] という時系列データは, [4,5,2,3,1,3] として圧縮される. 下線がフラグで, 最初の 2 つは負荷 4 が 5 回続いたことを意味する.

プロセス情報は次のようにして保存する. 時刻 t_n で実行中であったプロセス集合を, 時刻 t_n のプロセス状態とし, 過去のプロセス情報をプロセス状態が変化する位置を基準にして分割を行う. この分割したプロセス状態と, プロセス状態が変化する直前の時刻 t_n に対応する負荷の位置をペアにして, 時系列順に保存する.

負荷の位置は圧縮後の負荷時系列に index を与えておき, 格納されている負荷値の index:i とその負荷値が始まってから何番目かを表す j を用いて, $i:j$ と表す. **図 5** を使って具体的に説明する. 一番左側の実行中プロセス X が変化する直前の負荷の位置は, index:0 に格納されている負荷値 0 の 3 番目にあたるので, 「$X0:3$」として保存する. 左から二番目の実行中プロセス X,Y が次に変化する直前の負荷の位置は, 圧縮後の負荷時系列の index:3 で, そこに保存されている値は 1 つしかないので, 「$X,Y3:1$」として保存する. この変換を, すべての実行中プロセスが変化する負荷について行うと, 図 5 に示すものの圧縮後のようになる.

本システムでプロセス検索法による予測を行うときに, 時系列情報が参照される過去の負荷連続部分での非類似度計算は, すべで同一の結果となるため, 非類似度計算の省略を行うことが可能である. そのためにもこのデータ構造は適している

3.2.2 詳細プロセス情報

詳細プロセス情報は, "ユーザ名:プロセス名"をキーとしたハッシュテーブルであり, 値として過去に実行されたプロセスの情報 (引数, メモリ使用量の変動, cpu 実行時間) のリストを, プロセスの終了時刻順に保存している. 本システムでは, STLport[5] にある hash_map クラスを使うことで実装を行っている.

このデータ構造における計算量は,

- プロセス情報を新たに格納 : $O(1)$,
- 指定のユーザ名:プロセス名の情報を, 新しい順に m 個参照 : $O(m)$
- 現在時刻よりプロセスの終了時刻がある時間以上経っている物を消去 : $O(k)$,

となる. k は, キーとなっている"ユーザ名:プロセス名"の個数である.

また, 実行時間予測法の計算量は, 実行中のプロセスの数を n とすると, 実行中のプロセスと同一のユーザ名:プロセス名を m 個取り出す必要があるので, $O(mn)$ となり, プロセス数が多い状態でも比較的高速に動くと考えられる.

4. 実　　験

3 で述べた, プロセス検索法の改良による負荷予測精度, 予測時間への影響を確認するために実験を行った. 実験に使用した負荷時系列およびプロセス情報は, 主に科学技術計算に用いられている Xeon2.4GHz プロセッサを 2 台搭載した Linux マシンから収集した.

予測時刻は, 負荷変動が非定常時 (過去 60 分間の負荷値の分散が 0.2 以上) となる時刻を対象とした.

実験を実施した計算機環境は以下の通りである.

- CPU : Intel Xeon 2.4GHz;
- メモリ : 1024MB
- OS : Debian GNU/Linux 3.1 Sarge
- 使用言語 : C++ STLport-4.6[5]

4.1 現在時刻窓作成法の影響

現在時刻窓作成法と, 予測開始時刻 T がサンプリング時刻に一致する負荷時系列上でのプロセス検索法との予測誤差の比較実験を行った. 非類似度計算幅 $D = 3, 6, 9$, 仮予測数 $K = 5$, 予測窓幅 $p = 12$, サンプリング間隔 $d = 5min$, 隔分割数 $m = 5$, プロセスの検索期間は予測開始時刻から3ヵ月前までとした. 予測誤差は

$$PredictionError = \sum_{i=1}^{p'} \frac{|V_{n+i} - P_i|}{p'} \qquad (7)$$

と定義する. $p'(\leq p)$ は時刻 T で実行されているプロセスとは別のプロセスが投入される時刻を t' としたときに, $T + p'd \leq t'$ を満たす最大の整数である.

2000 回の実験を行い, その平均を**表 2** に示す. 表 2 の $t_{last} = T$ は, 予測時刻 T が t_{last} と一致する時の結果である. リアルタイムに対応させたことによる誤差増加はあるが, 0.05 以下で許容できる範囲である. 改良手法同士では, 誤差の分散が平行移動法のほうが大きく, 平行移動法のほうが失敗する場合が多かった.

4.2 窓幅自動決定の効果

窓自動決定の効果を示すために, 通常のプロセス検索法 ($D = 1, \ldots, 12$) と窓幅自動決定法を導入したプロセス検索法 ($D_{max} = 12$) の比較実験を行った. 仮

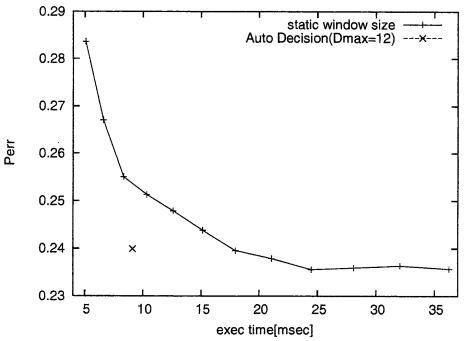

図 6 窓幅自動決定の影響

Fig. 6 Influence of window size automatic decision

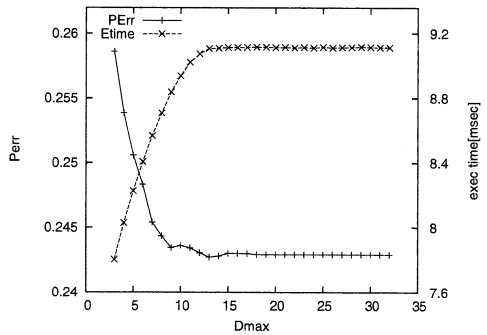

図 7 各最大窓幅における予測誤差と予測時間

Fig. 7 Prediction Error and Prediciton time of each maximam window size

予測数 $K = 5$, 予測窓幅 $p = 12$, サンプリング間隔 $d = 5min$, 隔分割数 $m = 5$, プロセスの検索期間は予測開始時刻から 3ヵ月前までとした. 各窓幅で予測を 2000 回行い, 誤差の平均を求めた. **図 6** に示す. 窓幅自動決定を導入したほうが従来法よりも, 同じ予測誤差では高速に, 同じ予測時間では低い予測誤差となった.

窓幅自動決定法における最大窓幅 $D_{max} = 3, 4, \ldots, 32$ の予測誤差, 予測時間を, 予測実験によって求めた. 予測実験は各 2000 回行い, その平均を**図 7** に示す. 図 7 から, 最大窓幅 D_{max} が増加するにしたがって, 予測時間は増加し, 予測誤差は減少する. $D_{max} = 9$ で変化は緩やかとなり, $D_{max} = 12$ でほとんど増減しなくなる. D_{max} は 12 以上が最適な値となる (12 より大きな値でも, 予測時間, 予測誤差の増加は発生しないので問題はない.).

5. まとめと今後の課題

本論文では, リアルタイムシステムのためのプロセス検索法の改良法と, 負荷時系列および詳細なプロセス情報を保存するデータ構造を提案した. 実験の結果, リアルタイムシステムに対応することで予測誤差の増加が見られたが, 許容範囲内であった. また, プロセス検索法に窓幅の自動決定を導入することで, 窓幅を予め決めた場合よりも高速かつ高精度に予測を行うことが出来た.

今後はリアルタイムシステム全体の予測実験を行い, 予測誤差や常駐負荷について調べる. また, リアルタイム負荷予測システムを実際にタスクスケジューリング等への利用方法を検討する. 例えば, Liu らの手法[6] を n-step 予測を利用するように改良するといった方法が考えられる.

表 2 リアルタイム対応による予測誤差への影響

Table 2 Impact of Real-Time system

非類似度計算幅	予測手法	$Perr$:平均	$Perr$:分散
$D = 3$	平行移動法	0.368	0.226
	最新負荷法	0.361	0.184
	$t_{last} = T$	0.329	0.182
$D = 6$	平行移動法	0.378	0.234
	最新負荷法	0.368	0.188
	$t_{last} = T$	0.339	0.188
$D = 9$	平行移動法	0.387	0.241
	最新負荷法	0.375	0.197
	$t_{last} = T$	0.345	0.196

参 考 文 献

1) Rich, W.: Dynamically forecasting network performance using the network wea her service, *J. Cluster Computing*, Vol. 1, pp. 119–132 (1998).
2) 小出洋, 山岸信寛, 武宮博, 笠原博徳: 資源情報サーバにおける資源情報予測の評価, 情報処理学会論文誌. プログラミング, Vol. 42, No. 3, pp. 65–73 (2001).
3) 立見博史, 菅谷至寛, 阿曽弘具: プロセス情報と実行時間予測を利用した統合的計算機負荷長期予測手法, 電子情報通信学会論文誌 D, Vol. J89-D, No. 11, pp. 2512–2516 (2006).
4) 丹野祐樹, 菅谷至寛, 阿曽弘具: プロセス情報を利用した実行時間予測と信頼度による予測選択手法, 情報科学技術レターズ, Vol. 6, pp. 21–23 (2007).
5) STLport: http://www.stlport.org.
6) Liu, C., Yang, L., Foster, I. and Angulo, D.: Design and Evaluation of a Resource Selection Framework for Grid Applications, *HPDC '02*, IEEE Computer Society, pp. 63–72 (2002).