入出力の仮想化と並列処理によるバッチ処理の高速化機能：PREST

長須賀 弘文[†] 吉澤 康文[†]
新井 利明[†] 今居 和男[††]

オンライントランザクション処理（OLTP）では，取り扱ったデータの集計やデータベースの更新等を行うバッチ処理が周期的に発生する．バッチ処理は，OLTP 終了後や低負荷時に実行されるが，OLTP の高負荷状態が長くなるにつれ，バッチ処理の性能を向上させる必要が生じている．OLTP に付随して実行されるバッチ処理のネックとして，実行中に生成されるデータをジョブ間で受け渡すための入出力があげられる．そこで，こうした入出力処理の高速化を図るバッチ処理高速化方式 PREST を提案する．PREST は，バッチジョブ特有のジョブ間のデータ受渡し用の入出力処理を主記憶上でシミュレートすると共に，データの一部が生成された時点で受渡しジョブを実行可能とすることでバッチ処理時間を短縮する機能である．また，PREST は，従来プログラムは変更せずに適用することが可能であり，さらに適用対象選定のための支援機能を用意した点に特長がある．PREST を適用評価した結果，入出力処理に要する CPU ステップ数を約 1/6 に，また，バッチ処理時間を約 1/2 に短縮した．

High Speed Function of Batch Processing by Virtualizing Input/Output and Parallel Execution: PREST

Hirofumi Nagasuka,[†] Yasufumi Yoshizawa,[†]
Toshiaki Arai[†] and Kazuo Imai[††]

In on-line transaction processing (OLTP), batch jobs are periodically executed to sum up data and update DBs. The batch jobs are executed after on-line transaction processing are completed or while system load for the transactions is low. As the demand for non-stop on-line transaction is increased, batch jobs are required to have higher performance. This paper presents a new high-speed batch processing facility, parallel reference and synchronous transfer facility (PREST). Batch job processings in OLTP are usually realized by passing data through a file from a job to a successive one. To reduce I/O operations for the data passing, PREST changes the I/O operations with main storage access. PREST also allows the batch jobs to be executed in parallel by scheduling a successive job when a job outputs a record of data to the successive one. Since PREST intercepts I/O operations, application programs for the batch jobs can take the benefits of PREST without modification. With PREST, the performance of batch job processing is significantly improved; (1) CPU overhead for the I/O operation is reduced to 1/6. (2) Execution time of a typical batch job is shortened to 1/2.

1. はじめに

オンライントランザクション処理（OLTP*）は，停止することなく運用されることが理想である．バンキングシステムなどにおいても，顧客へのサービス向上の面から，そのニーズが高くなっており，24時間/日，365日/年の連続運用の方向へ技術が進みつつある．ところで，OLTPが一定期間稼働した後には，トランザクションの集計処理や，振込み等によるデータベースの一括更新といったバッチ処理が周期的に発生する．これらの処理は，バッチ処理で取り扱うデータの矛盾を排除するため OLTP 終了後に実行されたり，あるいはトランザクション処理の応答を保証するため OLTP 低負荷時に実行される．ところが，OLTP が高負荷状態で運用される時間帯が長くなり，バッチ処理のための時間帯が次第に制限されるようになりつつある．そのため，バッチ性能を抜本的に改善する機能

† (株)日立製作所システム開発研究所第3部
Systems Development Laboratory, Hitachi, Ltd.
†† (株)日立製作所ソフトウェア開発本部
Software Development Center, Hitachi, Ltd.
* Online Transaction Processing.

が必要となった．

一般に，バッチ処理は複数のジョブで構成し，保守性および拡張性を高めている．おのおののジョブで実行させるプログラムの機能を独立させ，それらを組み合わせることでバッチ処理を構築し，業務が変化した場合でも該当するプログラムのみを修正/追加することで容易に対応がとれる構成となっている．このようなバッチ処理システムでは，先に実行されたジョブの実行結果を使用して，後続するジョブが処理を進める形態が多い．また，実行結果の受渡しには不揮発でジョブ間の共用が容易な外部記憶装置を利用したファイルが用いられている．

以上のようなバッチ処理形態では，データ量が増大するにつれ，ファイルに対する入出力処理時間がネックとなる．すなわち，処理の本体を実行する前に，データの読み込み処理が必ず行われ，結果は再びファイルに書き出されるからである．また，データの受渡しはファイル単位で行われるため，後続ジョブは，先行ジョブの実行が完了するまで実行を開始することができない．そのため，部分的に並列実行が可能な場合にもデータの受渡しのために，ジョブを逐次実行することとなり，処理時間を短縮できない．さらに，データを受け渡すために磁気テープを使用する場合には，掛け替え作業に人手を要している．集計処理ではデータ量の予測が不可能な場合があり，実質的に無限のファイル容量を確保できる磁気テープを使用しなければならないことがある．

本論文では，上記のような大量データを処理するバッチ処理を高速化する PREST* 機能を提案する[7),8)]．PREST は，逐次アクセス法を用いたジョブのデータ受渡し用の入出力処理を，主記憶上に設けたバッファを介したデータ転送でシミュレーションする．これにより，ファイルへの入出力操作に要する処理時間を削減する．また，データ受渡し用のデータ転送単位を，ファイルの構成要素であるレコード単位で制御し，先行ジョブの実行結果の一部が生成された時点で後続ジョブを実行可能とすることで，ジョブを並列実行させ，トータルスループットの向上を図る．さらに，これらの機能をプログラムの修正なしに適用可能とする．

入出力処理を主記憶上のデータ転送でシミュレートして高速化する手法は，入出力の仮想化技術として知られている[1),2)]．従来の仮想化技術は，前もって外部記憶装置上のファイルを仮想記憶上に読み込み，入出力をページング動作で行うものである．そのため，データ保全のための書き込み処理，効果的なバッファ管理方式，ファイルアクセスのローカリティ等の問題がある[5)]．したがって，アクセスにローカリティがあり，かつ読み出し頻度が高いファイルを高速化の対象[6)]としていた．

PREST は，アクセスローカリティが低い逐次アクセス法を用いたファイルへのアクセスで，書き込み頻度と読み出し頻度が同等であるファイルに対して入出力仮想化の効果をあげることを目的とする．PREST では，バッチ処理のデータ受渡し処理に着目し，(1)データ受渡し用入出力操作の削除方式，(2)データの一部が利用可能となった時点で後続ジョブの実行を可能とする並列同期方式，および(3)小量のメモリを使用するだけでデータ受渡し処理を実現するバッファ制御方式，によるバッチ処理の高速化方式を提案する．

また，大規模 OLTP システムでは，数千本のジョブを毎日実行させている．その中から PREST を適用できるデータ受渡し処理を人手で抽出するためには，膨大な作業を要する．そこで，オペレーティングシステムが収集するシステム稼働情報をもとに，PREST の適用が可能なデータ受渡し処理を抽出する支援機能（PREST/AID）を提供する．これにより，PREST の適用に要する作業の省力化を図る．

本稿は，PREST の実現方式とその適用効果，ならびに PREST/AID について論ずる．

2. PREST の特徴と効果

PREST では，大量データを取り扱うバッチ処理において，ジョブ間でのデータ受渡し処理に着目し，並行して実行しているジョブ間でのデータ受渡し処理を可能とすることでジョブの並列度向上を図り，データ受渡し用入出力をメモリ上のデータ転送でシミュレーションすることで物理入出力の削減を図る．これにより，バッチ処理時間の短縮を図る．

こうした大量ファイルの受渡し処理は，同一ジョブ内のジョブステップ間でも存在することが想定できる．しかし，多段のジョブステップで構成されるジョブは，上段のジョブステップから一段ずつ実行される．PREST はデータの受渡しをするアプリケーションプログラムを並列実行させることで処理時間の短縮

* Parallel REference and Synchronous Transfer facility.

を図るため，逐次的に実行されるジョブステップ間でのデータ受渡し処理には適用できない．PREST の適用対象は，ジョブ間でのデータ受渡し処理のみである．

以下，本論文で記す PREST の特徴を示す．

2.1 動的なバッファ容量制御

データを出力するジョブ（出力ジョブ）と，そのデータを入力するジョブ（入力ジョブ）の実行状況は異なることが普通であり，出力ジョブのデータ出力頻度と入力ジョブの入力頻度は異なる．そのため，データ出力頻度が高い場合には無限の主記憶バッファが必要となり，逆に，入力頻度が高い場合には入出力要求ごとに出力ジョブと入力ジョブ間の排他/同期処理が必要となり，CPU オーバヘッドが増加する[3),4)]．PREST では，ジョブの実行状況によって使用するメモリ量を動的に変更することにより，計算機資源の有効活用と処理の高速化を可能とする．

2.2 同期処理オーバヘッドの削減

並列実行しているジョブ間でデータを受け渡す場合には，出力ジョブと入力ジョブとの間で，お互いにデータ受渡しのための待ち合わせをする必要がある．このジョブ間での待ち合わせを PREST が制御する．これを同期制御と呼ぶ．

ジョブ間で大量データ量を受け渡す場合に，同期制御を実行する頻度が高くなると，CPU オーバヘッドが処理の遅延を招く．そこで，用意したバッファ容量で同期処理の頻度を最小にするように制御し，ジョブの実行状況により，データ受渡し用の入出力頻度が異なるために，同期処理が多発するときには，バッファを動的に増量することで，同期処理頻度を低減させる．このように，動的なバッファ容量の変更により主記憶の有効活用を図ると共に，同期処理回数を削減することで CPU の有効利用を図る．

2.3 異常終了時のジョブ間の連動

バッチジョブ環境のように，複数のジョブで一連の業務を遂行するような場合には，あるジョブが異常終了した場合には，ある時点（チェックポイント）にまで遡ってジョブ群を再実行させる必要がある．従来，ジョブは逐次実行されていたためジョブの正常終了を確認した後，次のジョブを起動していた．PREST ではジョブを並列実行させるため，あるジョブが異常終了したときには他のジョブはすでに起動された後である．そこで，ジョブが異常終了した場合には，受渡しの相手となっている他方のジョブを異常終了させる機能を提供する．

2.4 汎 用 性

大規模オンラインシステムでは膨大なアプリケーションプログラムが蓄積され，日々実行されるバッチジョブも数千ジョブ以上となる．そのため，これらのジョブを PREST 適用を可能とするよう修正することは事実上不可能である．そこで，プログラムを修正することなく，ジョブの実行環境を規定するジョブ制御文の修正のみで PREST の高性能化機能を適用可能とする．さらに，ジョブ実行中にファイル入出力処理をトラップして，逐次アクセス法を用いたファイルアクセスであることを確認することで PREST の適用性を判断する．

2.5 支 援 機 能

オペレーティングシステムは，複数のアクセス法を用意し，ユーザは個々の処理の目的に応じてアクセス法を使い分け，効果的なファイル入出力を行っている．そのため，PREST を適用するために，数多くのジョブの中から，逐次アクセス法を用いてファイルへアクセスしている処理を抽出しなければならない．また，どのジョブが出力したデータをどのジョブが参照しているのかをも判別しなければならない．数千本のジョブを実行させるような大規模システムでは，各ジョブについて，プログラムが実行時に使用するアクセス法や，ジョブ間のファイル受渡し関係をすべて掌握することは，事実上不可能である．そこで，逐次アクセス法を使用したデータの受渡し処理を自動的に検出する支援機能（PREST/AID）を提供する．これにより，バッチ処理への PREST の適用に要する作業時間を短縮する．

以上の特徴を持つ PREST を，データを受渡しながら処理を進めるバッチ処理に適用すると，以下の効果が期待できる（**図1**参照）．

（1） バッチ処理時間の短縮が図れる．

（2） 磁気ディスクや磁気テープなどへの入出力動作を削減できるため，入出力装置，チャネル等のネックを回避できる．

（3） 入出力動作をメモリ上でのデータ転送で実現するため，高速なデータ受渡しを行うことができる．

（4） 出力ジョブと入力ジョブのデータ入出力頻度の差に応じてバッファ容量を決定するので，小量のメモリでデータ転送を実現できる．

（5） PREST を適用するプログラムの変更を必要と

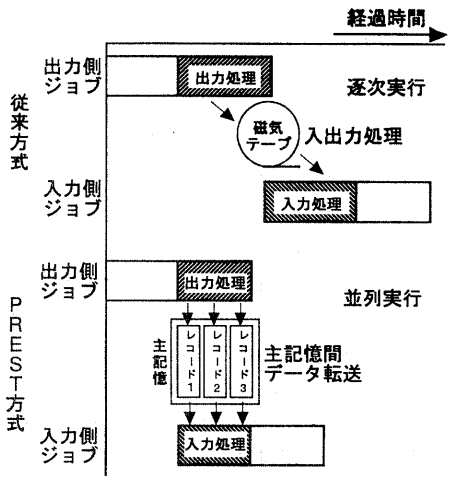


図 1 PREST の概要
Fig. 1 Summary of PREST.

しない.

3. PREST の実現方法

3.1 解析モデルによる制御方式の検討

PREST では，出力ジョブが書き出したデータを主記憶バッファに格納しておき，入力ジョブからの読み込み要求が発生した時点で，そのデータを受け渡すことにより入出力処理をシミュレートしている．このとき，出力ジョブからの書き出し要求が入力ジョブの読み込み要求に比べて大きい場合には，主記憶バッファは無限に大きくなる可能性がある．逆に，入力ジョブの読み込み要求頻度が高い場合には，主記憶バッファ容量は1レコード分用意すればよいが，入出力要求ごとに同期制御が必要となり，CPU オーバヘッドが増加する．すなわち，入力頻度と出力頻度の差が大きい場合でも入力ジョブと出力ジョブを待たせることなく実行させるためには，ある程度，主記憶バッファの容量を大きくする必要があり，同期オーバヘッドを削減するためには複数のレコードに対して同期を取る必要がある．そこで，これらを以下のようにモデル化し，出力頻度と入力頻度の関係から，バッファ容量と同期制御を実行するタイミングを決定する．

（1） バッファ領域は先頭領域から順次使用し，入力ジョブが読み込んだ領域は再利用する．

（2） バッファが空のときに入力ジョブがデータを読み込もうとした場合，入力ジョブは出力ジョブがmレコード分のデータをバッファに書き込むまで待つ．

（3） バッファがフルのときに出力ジョブが次のデータを書き込もうとした場合には，出力ジョブは入力ジョブがmレコードのデータを読み込むまで待つ．

ここで，

λ：書き込み頻度（回/sec）

μ：読み込み頻度（回/sec）

n：バッファ領域のレコード数（レコード）

として解析を行う．

【解析1】 書き込み頻度が読み込み頻度よりも高い場合（$\lambda>\mu$）

バッファがフルなため出力ジョブが待ち状態になり，入力ジョブが m レコードのデータを読み込んだ後，出力ジョブの待ち状態を解除したとする．待ち状態を解除した後，再び出力ジョブが待ち状態になるまでの時間 t は，式(1)で求められる．

$$\lambda t=\mu t+m$$
$$\therefore\ t=m/(\lambda-\mu) \qquad (1)$$

この時間 t が長ければ，同期制御の実行頻度が低くなる．バッファのレコード数は n なので，t の最大値は，$m=n$ のときである．

これは，一度待ち状態になった出力ジョブは，入力ジョブがバッファ内のデータをすべて読み込むまで待ち状態でいることが，同期制御の実行頻度が最小になることを意味する．

さらに，時間 t の間に α 回以上のデータ書き込みをすることができれば，同期制御の実行頻度が，α回のデータ書き込みに対して1回未満とすることができる．この目標値を満足するための最小のバッファ容量（レコード数）n は，式(2)で求められる．

$$\lambda t=\lambda n/(\lambda-\mu)>\alpha$$
$$\therefore\ n=\{\alpha(\lambda-\mu)/\lambda\}+1 \qquad (2)$$

【解析2】 読み込み頻度が書き込み頻度よりも高い場合（$\lambda<\mu$）

入力ジョブが待ち状態になり，出力ジョブが m レコードのデータを書き込んだ後に，入力ジョブの待ち状態を解除したとする．待ち状態を解除した後，再び入力ジョブが待ち状態になるまでの時間 t は，式(3)で求めることができる．

$$\mu t=\lambda t+m$$
$$\therefore\ t=m/(\mu-\lambda) \qquad (3)$$

この時間 t が長いほど，同期制御の実行頻度が低くなる．バッファのレコード数は n なので，t の最大値は $m=n$ のときである．

これは，一度待ち状態になった入力ジョブは，出力ジョブがバッファ内のすべてのレコードにデータを書

き込むまで待ち状態になっていることが，同期制御の実行頻度を最小にできることを意味する．

さらに，この時間 t に入力ジョブが，α 回以上のデータを読み込むことができれば，同期制御の実行頻度を α 回のデータ読み込みに対して1回未満にできる．この目標値を満たす最小のバッファ容量（レコード数）n は，式(4)で求められる．

$$\mu t = \mu n/(\mu-\lambda) > \alpha$$
$$\therefore\ n = \{\alpha(\mu-\lambda)/\mu\} + 1 \qquad (4)$$

式(2)と式(4)から，書き込み頻度と読み込み頻度に応じて，式(5)に示すバッファ領域を確保すればよい．

$$n = \frac{\alpha|\lambda-\mu|}{\max(\lambda, \mu)} + 1 \qquad (5)$$

ここで，α を一定値とすると，書き込み要求頻度と，読み込み要求頻度を観測しておけば，α 回のデータ受渡し処理に対して，同期制御の実行頻度を1回未満にできる最小のバッファ容量を求めることができる．

3.2 同期制御方式とバッファ容量管理方式

出力ジョブと入力ジョブが共に待ち状態となっていないときに，データ受渡し用の入出力回数を観測する．これを入出力頻度として式(5)へ代入することで，用意すべきバッファ容量を動的に決定する．

さらに，式(1)と式(3)より，同期制御と同期制御との時間間隔 t が最大となるのは $m=n$ のときであり，このとき，同期制御を実行する頻度が最小となる．すなわち，データ受渡しのために待ち状態の解除は，もう一方のジョブが待ち状態になる直前に行うことで，同期制御を実行する頻度を最小化できる．

3.3 異常処理方式

PREST を適用しない従来のケースでは，出力ジョブと入力ジョブを逐次実行させている．したがって，どちらかのジョブが，何らかの理由で異常終了した場合，その原因をつきとめ，異常終了したジョブを再実行させていた．

PREST を適用するケースでは，出力ジョブと入力ジョブとを並列実行させる．もし，どちらかのジョブが何らかの理由で異常終了した場合には，出力ジョブと入力ジョブの両方を再実行させなければならない．そこで，PREST では，1レコードのデータ受渡しをするたびに，関連したジョブが異常終了していないかをチェックしながら処理を進める．異常終了していることがわかった場合には，自ジョブも異常終了する．そして，出力ジョブと入力ジョブをともに再実行させる．これは PREST の一つの欠点である．

3.4 ユーザインタフェース

既存のアプリケーションプログラムを全く改造することなく，PREST を適用できることが望ましい．そこで，PREST の利用はジョブ制御文の装置指定のパラメタで指定する．また，受け渡すデータの出力先を定義したファイル名を出力ジョブのジョブ制御文で指定し，同一のファイル名を入力ジョブのジョブ制御文で指定することで，ジョブ間でのデータ受渡し関係を決定する．さらに，アプリケーションプログラムが発行した入出力要求(スーパバイザコール)をオペレーティングシステムの中で制御を奪い，データ転送に置き換える方式で実現する．

これにより，アプリケーションプログラムを改造せずに，PREST を利用することができる．

4. PREST 支援機能 (PREST/AID) と適用効果

4.1 PREST 支援機能 (PREST/AID)

大規模システムでは，毎日数千本のバッチジョブを実行させている．これらのジョブの中から，PREST を適用できるジョブ列を人手で抽出する作業工数は大きく，PREST の実用性の障害となる．

一般に，大規模な計算機システムでは，管理者がシステムの動作を正しく把握し，分析を可能とするシステム稼働情報を収集している．収集されるシステム稼動情報には，ジョブステップ単位の課金情報や計算機資源の利用状況等がある．

PREST/AID は，このシステム稼働情報を用いて，PREST を適用できるデータ受渡し処理を自動的に抽出する機能である．

PREST/AID が参照するシステム稼動情報は，ジョブの実行履歴情報と各ジョブが実行中にアクセスしたファイルアクセス履歴情報である．これらの情報から，ジョブ間でのデータ受渡しの関係と各データ受渡し処理で使用するアクセス法を求め，PREST を適用できるデータ受渡し処理を抽出する．

ユーザは，PREST/AID によって抽出されたジョブ列のジョブ制御文を PREST 適用可能なジョブ制御文に変換することで，バッチジョブに PREST を適用できる．

PREST/AID が PREST の適用が可能なデータ受渡し処理を抽出するアルゴリズムは，以下のとおりで

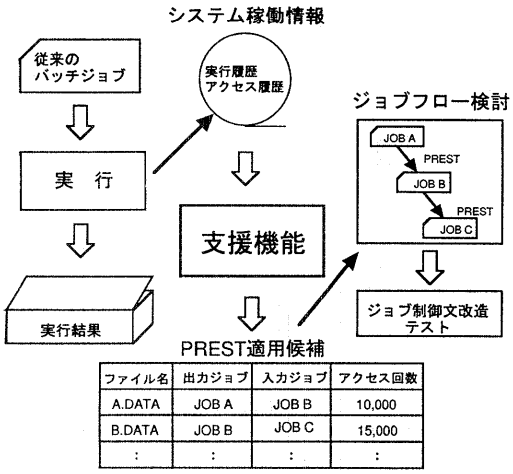


図 2 支援機能を用いた PREST 適用手順
Fig. 2 A procedure to apply PREST to conventional batch jobs by using PREST/AID.

ある.

(1) あるジョブステップが実行結果として出力したデータを他のジョブステップが参照するデータ受渡し処理であること.

(2) 逐次アクセス法を用いたデータ受渡し処理であること.

(3) 出力された中間データを参照するのは，ある一つのジョブのみであること．すなわち，出力ジョブと入力ジョブが1対1のペアとなっていること.

図2に，ユーザが PREST/AID を用いて上記条件を満たすデータ受渡し処理を抽出し，バッチジョブに PREST を適用して実行させるまでの過程を示す.

(a) PREST を適用する計算機システムで，従来のバッチジョブを実行させて，システム稼働情報を収集する.

(b) 収集したシステム稼働情報を入力情報として，PREST/AID を実行させ，PREST の適用候補処理を抽出する.

(c) 抽出した PREST の適用候補処理から，保存が不要なデータを受け渡す処理を抽出し，PREST を適用する処理を決定する.

(d) PREST の適用が決定した処理を，ジョブ間でデータ受渡し処理で実行させる.

4.2 PREST/AID を用いた適用処理の抽出

第5章で述べる適用評価の例では，PREST/AID を使わずに PREST を適用した．そこでの適用基準は，上記の(1)～(3)に加え，データ受渡しのデータ量が大きい処理とした.

このとき，PREST 適用の準備に要した時間の割合は，以下のとおりである.

(1) PREST 適用箇所の抽出に 20% 費やす
プログラムの仕様書やソースプログラム等から PREST を適用できる処理を抽出する作業時間

(2) ジョブフロー検討に 50% を費やす
PREST を適用したときのジョブ実行順序の正当性検証の作業時間

(3) JCL 修正ならびにテストに 30% を費やす
バッチジョブ起動用の JCL を PREST 用に修正し，動作の確認に要した時間

PREST/AID を用いれば，(1)の作業を削除することができるので，PREST 適用のための作業を 20% 短縮できる．ただし，この抽出作業は，PREST を適用するジョブをあらかじめ 17 本に定めて適用箇所を抽出したものである.

PREST/AID では，適用候補となった処理のデータ受渡し用の入出力回数もレポートする．そのため，適用効果が大きいデータ受渡し処理を的確に抽出することができる．したがって，不特定多数のジョブ（例えば数千本のジョブ）から PREST 適用候補処理を抽出するときに PREST/AID を用いると，より効率的な作業を期待できる.

5. PREST の適用と評価

5.1 同期制御方式とバッファ容量決定の評価

HITAC M-682 を用い，マルチプロセッサ環境下とユニプロセッサ環境下において，ジョブ実行中に PREST が設けるバッファ容量の変化と，同期制御の実行頻度を実測した．この実測では，他の入出力を介在させず，データの受渡し処理のみをする処理に PREST を適用した.

表1に，バッファの初期割当て量を 10 レコードとし，同期制御の実行頻度を 50 回のデータ転送に1回未満となることを目標にした場合の結果を示す.

表 1 バッファ容量の変化と同期処理頻度
Table 1 Buffer capacity and frequency of synchronous control.

		ユニプロセッサ	マルチプロセッサ
転送レコード数		100,000	100,000
同期処理回数		2,043	572
同期処理頻度		1/48.9	1/178.8
バッファ容量（レコード数）	初期割当て量	10	10
	実行終了後の容量	50	10

同期制御の実行頻度は，目標値に近い値もしくは目標値を満たす値となっている．バッファ容量の変化では，ユニプロセッサの場合，データ受渡し中にバッファ容量を10レコードから50レコードへと増量している．それに対し，マルチプロセッサでは，初期割り当てである10レコードのままでデータ受渡し処理を完了した．この違いの要因は以下のとおりである．

（1） ユニプロセッサの場合

CPUが一つであるため，ある瞬間では実行可能なジョブは，出力ジョブと入力ジョブのいずれか一方である．したがって，タイムスライス内に出力ジョブがすべてのバッファにデータを書き込んで待ち状態になる．次に，入力ジョブがバッファのすべてのデータを読み込んで，出力ジョブの待ち状態を解除するといった処理を繰り返している．PRESTは，出力ジョブと入力ジョブの入出力要求の割合いが大きく異なると判断し，途中でバッファ領域を拡張して，同期処理のオーバヘッドを削減している．

（2） マルチプロセッサの場合

出力ジョブと入力ジョブは，おのおの別個のプロセッサで，同時に実行することが可能である．このため，本実測では，178.8回のデータ受渡し処理に対して1回の割合いで同期制御を実行していた．このため，初期割当てである10レコードのまま，データ受渡し処理を完了した．

5.2 入出力にかかるCPUオーバヘッド

1レコードのデータの入出力を仮想化するために要するCPUオーバヘッド（I/O処理ステップ数）と，入出力装置へのI/O処理ステップ数を実測した．その結果，I/O処理ステップ数を約1/6にした（**図3**参照）．

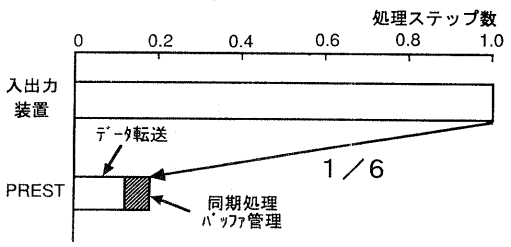


図3 I/O処理ステップの比較
Fig. 3 Comparison of CPU steps for I/O.

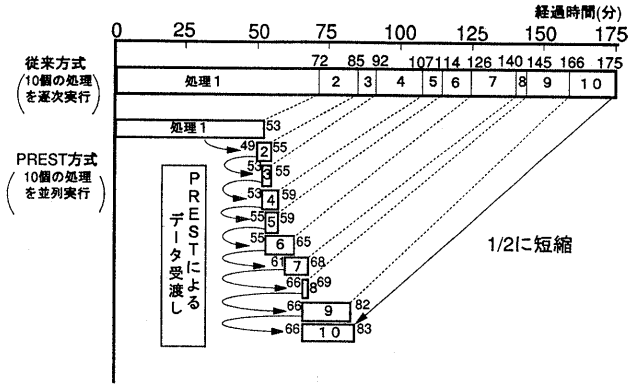


図4 1ジョブの処理時間短縮の内訳
Fig. 4 Comparison batch execution time between conventional method and PREST method.

入出力装置に対するI/O処理では，入出力動作が完了するまでデータが主記憶上に存在することを保証するための処理や，入出力処理が正常に終了したことを確認する処理が必要となるが，PRESTではこうした処理が削除される．しかし，PRESTでは，先に述べた同期処理や，データ受渡し処理の実行中に，バッファ領域を拡張するか否かを判断するときに出力ジョブと入力ジョブの実行をシリアライズさせるために必要となるバッファ管理処理が生ずる．そこで，図3では，同期制御用のCPUステップとバッファ管理用のCPUステップを，1回の入出力操作あたりに換算し，PRESTのI/O処理ステップに加算した．

5.3 実環境における評価

開発したPRESTを流通系のバッファ処理業務に適用し，効果を評価した．適用したバッチ業務は，毎日，OLTP終了後，その売り上げ情報や在庫情報を商品ごとに集計処理を行う．そのため，その日の売り上げ情報等の大量データを一括処理する必要があるため，データの入出力動作がネックとなっていた．

評価のために適用した現行バッチ業務の規模は，以下のとおりである．

（1） ジョブ数：17

（2） ジョブステップ数：487

（3） PRESTの適用数：79

ここで，同一ジョブ内のジョブステップ間において，PRESTの適用が可能なデータ受渡し処理は，そのジョブを複数ジョブに分割してPRESTを適用し，実行させた．また，使用したマシンはHITAC M-660で，主記憶容量は64メガバイトである．

実測の結果，バッチ処理の総処理時間を380分から254分の約2/3に短縮した．そのバッチジョブの中で適用効果の大きいジョブの処理時間の短縮の様子を**図4**に示す．このジョブでは，9カ所のデータ受渡し処理にPRESTを適用した．そのため，従来は1本のジョブで実行されていた処理を，10本のジョブに分割して実行した．PRESTを適用した結果，処理時間を，173分から83分へと1/2以下に短縮した．

6. おわりに

バッチ処理時間の短縮を目的とするPRESTを提案し，開発・評価した．PRESTの制御方式には，(1)ジョブの入出力要求の割合いに応じたバッファ容量決定と，(2)データ受渡しのための同期をとるタイミングを少なくする待ち合わせ制御方式に特徴がある．さらに，PREST適用にかかる工数の削減を目的とするPREST/AIDを提案し，開発をした．

性能評価の結果，データ受渡しにかかるCPUオーバヘッドを，外部記憶装置への入出力操作をした場合に比べ，約1/6に削減できた．

ケーススタディとして取り上げた，OLTP夜間バッチ処理にPRESTを適用した結果，総処理時間を従来の約2/3に短縮できた．その中で，適用効果が顕著なジョブでは，処理時間を1/2以下にした．性能評価に用いたバッチ処理にPRESTを適用するときに，PREST/AIDを用いた場合，適用に要する作業時間を約20%短縮できる見込みを得た．

謝辞 本研究の推進にあたり，(株)日立製作所システム開発研究所堂免信義元所長ならびに久保隆重元副所長のご指導に感謝する．また，ソフトウェア開発本部ならびに日立ソフトウェアエンジニアリングの方々からは，実用化における多くのご意見とご協力を頂いた．特に，ソフトウェア開発本部小平光彦元部長には，本研究の成果をVOS3/ASに適用する機会を頂き，多大のご支援を頂いた．ここに感謝の意を表す．

参考文献

1) Stonebraker, M.: Virtual Memory Transaction Management, *ACM Operating Systems Review*, Vol. 18, No. 2, pp. 26-48 (1982).
2) Robinson, J. T. and Devarakonda, M. V.: Data Casche Management Using Frequency-Based Replacement, *ACM Proceedings of SIGMETRICS*, pp. 134-142 (1990).
3) Courtoris, P. J.: Concurrent Control with "Readers" and "Writers", *Comm. ACM*, Vol. 14, No. 10, pp. 667-668 (1971).
4) Johnson, T.: Approximate Analysis of Reader and Writer Access to a Shared Resource, *SIGMETRICS ACM*, Vol. 18, No. 1, pp. 106-113 (1990).
5) Baccelli, F. and Coffman, E. G., Jr.: A Data Base Replication Analysis Using an M/M/m Queue with Service Interruptions, *SIGMETRICS Performance Evaluation Review*, Vol. 11, No. 4, pp. 102-107 (1982).
6) 櫻庭ほか：入出力の仮想化と計算機高速化機能の開発とその分析・評価，情報処理学会論文誌，Vol. 33, No. 11, pp. 1341-1350 (1990).
7) 長須賀ほか：ジョブ間並列同期転送機能の開発と評価—VOS 3/AS：PREST—，第41回情報処理学会全国大会論文集 (4), pp. 65-66 (1990).
8) 長須賀ほか：ジョブ間並列同期転送機能，(PREST) の開発と評価，第54回情報処理学会オペレーティングシステム研究会，pp. 17-24 (1992).

(平成5年1月27日受付)
(平成6年1月13日採録)

長須賀弘文（正会員）

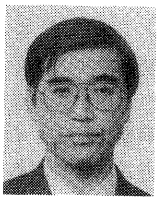
昭和36年生．昭和59年上智大学理工学部機械工学科卒業．昭和61年同大学院理工学研究科修了．同年，(株)日立製作所入社．システム開発研究所にて，メモリ利用技術，性能評価技術等，大型計算機システムのオペレーティングシステムに関する研究・開発に従事．日本OR学会会員．

吉澤　康文（正会員）

1944 年生．1967 年東京工業大学理工学部応用物理学科卒業．同年(株)日立製作所入社．中央研究所に勤務し，HITAC 5020/TSS の研究開発に従事．1973 年発足したシステム開発研究所に勤務．以来，仮想記憶，大規模 TSS，オンラインシステム，など大型計算機のオペレーティングシステムの性能評価ならびに記憶管理方式の研究開発に従事．これらの研究により，1981 年 3 月東京工業大学より工学博士を授与．また，オペレーティングシステムのテスト・デバッギングシステムの開発に従事．現在はハイエンドサーバならびに超並列計算機の研究開発を推進している．情報処理学会論文賞（昭和 47 年度）受賞．IEEE Computer Society 会員．同研究所主管研究員．著書「オペレーティングシステムの実際」（昭晃堂）．東京農工大学，電気通信大学，東京工業大学，非常勤講師．

新井　利明（正会員）

昭和 29 年生．昭和 51 年早稲田大学理工学部電気工学科卒業．昭和 53 年同大学院理工学研究科修了．同年(株)日立製作所入社．システム開発研究所にて，大型計算機のオペレーティングシステムの性能評価および性能向上方式の研究・開発に従事．IEEE Computer Society，ACM 各会員．



今居　和男（正会員）

1977 年東京大学大学院修士課程修了．同年，(株)日立製作所に入社．大型計算機システムの基本ソフトウェアの開発に従事．