非同期的ログバッファ転送機構によるリモートロギングの効率的実行

川島　英之†

† 慶應義塾大学 理工学部 情報工学科
〒 223-8522 神奈川県横浜市港北区日吉 3–14–1
E-mail: †kawasima@ics.keio.ac.jp

あらまし　本研究の課題は，DBMS へ頻繁に挿入トランザクションが到着する環境で下記項目を実現することである．(1) スループットの極大化，(2) ロギング時間の極小化，(3) ブロッキング時間の極小化．これらの課題を解決するために，本論文は非同期的ログバッファ転送機構を提案する．そして提案機構を設計・実装し，実験を通してその性能を評価する．実験結果は，リモートロギングはディスクロギングに比べて (1) は 10 倍程度，(2) は 17 倍程度優れた値を示し，(3) は平均 4.38 マイクロ秒という極小値を示す．

キーワード　センサデータ，先行書き出しロギング，データストリーム，リカバリ，データベースシステム

An Efficient Execution of Remote Logging by Asynchronous Log Buffer Transfer

Hideyuki KAWASHIMA†

† Department of Information and Computer Scicence, Faculty of Science and Technology, Keio University
Hiyoshi 3–14–1, Kohoku-ku, Yokohama, Kanagawa, 223–8522, Japan
E-mail: †kawasima@ics.keio.ac.jp

Abstract　For the frequently arriving data environment, this paper tackles the following three problems. (1) Maximizing throughput. (2) Minimizing logging time. (3) Minimizing blocking time. To solve these problems, this paper proposes asynchronous log buffer transferring technique. Furthermote this paper designs and implements it. To evaluate the technique, this paper conducts experiments. The result of experiments show that remote logging provides better performance than disk logging, 10 times for (1), 17 times for (2). Furthermore, average blocking time is shown as 4.38 micro seconds.

Key words　Sensor Data, Write Ahead Logging, Data Stream, Recovery, Database System

1. はじめに

実世界認識を目的としたセンサネットワーク [9] や無限長のデータストリームを処理する DSMS [4] [2] [1] の研究が盛んに行われている．センサネットワークではアグリゲーション技術に，DSMS ではデータストリームの効率的なフィルタリング技術に主眼が置かれており，いずれの研究でもデータの即時永続化は考えられていない．

このことが，これまでのセンサネットワークやデータストリームの応用が警報器 (Alerter) 程度に制限された理由だと筆者は考えている．もしも頻繁に DBMS に到着するセンサデータやデータストリームを永続化できれば，これまでは主にオフィス/ビジネス用に使われていた DBMS を実時間処理分野 (プラント，自動車，医療) へ適用可能になり，過去履歴を用いて危険状態/要求サービスを予測する，予報器 (Predictor) 開発を支援できるだろう．このようなビジョンを実現するために，本論文では頻繁に DBMS に到着するデータを確実に高速に挿入する技法を提案する．

そもそも，データを消失させないために，DBMS は作成したログを永続記憶装置に記録したのち持続記憶装置の更新を行う [14] [5]．ログを永続記憶装置に記録する処理を本論文では永続化処理と表記する．永続化処理を高速化する技法として，これまでページ差分を複数ファイルに分割処理する差分ロギング方式 [3] と，2 台のリモートマシン上メモリを永続記憶装置とするリモートロギング方式 [6] [12] [13] が考案されてきた．文献 [3] の技法，差分ロギングは主記憶 DBMS に対して，ページ毎にログファイルを作成し，それらのログファイルを分割することで更新処理を高速化する．文献 [6] の技法，隣接 WAL は 2 相コミットを用いて近接 2 ノードのメモリにログを置くリモートロギングによりロギング処理を高速化する．文献 [12] の技法

は “センサデータは多少欠落しても補間可能” というコンセプトの元，インプリサイス計算モデル [8] に基づいてロギングを不確実に実行することでロギングを高速化している．文献 [13] の技法は確実に永続化可能なリモートロギングを実行する．

これらの技法は，挿入処理が頻繁に生じるセンサデータを処理するには問題がある．文献 [3] の技法は主記憶に収まるサイズのデータにしか適用できないために，無限に到着するデータを蓄えることはできない．文献 [6] の技法では，メモリに蓄えたログが溢れた際の処理手法が不明確である．文献 [12] の技法は，性能向上を実現するためにロギングをあえて不確実に実行する．そのため同技法はロギングを失敗する可能性がある．さらに，そもそもこの技法は固定長データのみを扱うシステム向けに開発されており，可変長タプルの作成が必要な DBMS には適用できない．文献 [13] の技法はログバッファが一杯になった時にログレコードをスレッドから集めて時系列順に整列するアルゴリズムを採用している．従って同処理時には数十秒　数分程度のブロッキングが発生し，それが DBMS の挿入処理スループットを激減させる．

本論文では，これまでの研究では実現されなかった，頻繁に DBMS に到着するデータを確実に挿入可能なリモートロギング技法を提案する．提案技法の核は，**2 レベル非同期的ログ転送**による，リモートログバッファ溢れ対処の高速化にある．

本論文の構成は次の通りである．2 節では関連研究を述べる．3 節では本研究で扱う課題を定式化すると共に，提案を述べるために必要となる DBMS の基本機構を述べる．4 節では課題解決案をアプローチ面とアルゴリズム面から述べる．5 節では提案機構を実験的に評価する．6 節では議論を行い，最後に 7 節では結論を述べる．

2. 関 連 研 究

2.1 並列ディスクを用いる高速永続化技法

P*TIME [3] [7] では WAL を高速化するために Differential Logging という手法を用いている [7]．差分ロギングの特徴は 2 つある．第 1 特徴は差分ログ方式を用いることである．これによりログの量を物理ロギングを用いる場合の半分程度まで減らせる．なぜなら更新操作のログとして，ARIES 方式 [10] の物理ロギングは before イメージと after イメージの両方を必要とするが，差分ロギングはそれらのビットレベルの XOR 差分しか必要としないからである．第 2 特徴は WAL ファイルを複数個用いることである．これにより WAL 実行時の負荷を分散し，WAL の実行を高速化する．通常は WAL ファイルの分割 (ストライピング) はチェックポインティング処理の性能劣化を招くが，差分ロギングはページ単位の差分ログを利用することで性能向上を実現している．

P*TIME がセンサデータ処理に不向きな点は，全てのデータがメモリに常駐することが求められる点である．更新が頻発するだけのデータベースならばこの方式は適用可能だが，センサデータは更新ではなく挿入が主たる操作であるために，時間と共にデータベースのデータ量は増加する．それゆえ差分ロギングは挿入型データに適用できない．

2.2 リモートメモリを永続記憶装置とする高速永続化技法

2.2.1 ClustRa 方式

リモートメモリを永続記憶装置とする高速永続化技法として隣接 WAL がある [6]．ClustRa が用いる隣接 WAL とは，ローカルディスクの代わりに 2 台の隣接ノードのメモリに 2 相コミットを用いてログを書く方式である．ローカルディスクを使わない理由は，ログをディスクに書くよりも隣接ノードのメモリにログを書く方が高速であるために WAL の高速化が見込まれるからである．ClustRa の永続化技法である隣接 WAL はディスクを用いる WAL に比べて高性能を実現できると考えられる．

しかし，ClustRa は主記憶データベースシステムとされていることから，全てのデータが主記憶に収まることが仮定されていると考えられる．さらに，時間と共に増え続けるデータを管理するには，リモートメモリ溢れの対処が必要だが，その技法は ClustRa では考えられていない．従って本研究の目的に ClustRa は不適である．

2.2.2 不確実リモートロギング

文献 [12] では，性能向上を図るためにリモートロギングを不確実に実行する技法が提案されている．これを本論文では不確実リモートロギングと表記する．この技法はリモートロギングを高性能化するが，各ロギングは失敗する可能性があるため，データが失われる可能性がある．一方，本研究の目的は確実なロギングであるため，不確実リモートロギングは本研究の目的に不適である．

2.2.3 応答時間最小化リモートロギング

文献 [13] はセンサ指向 DBMS である KRAFT における確実なリモートロギングを示している．これはセンサデータストリームごとにコネクションを担当するスレッドをリモートログサーバで立ち上げ，スレッドごとにログレコードを保持する方式である．この方式は排他制御が不要であるためにロギング処理時間を極限まで短縮するため，応答時間を最小化するリモートロギング技法だと考えられる．しかし，リモートログサーバにおいてログレコードに使えるメモリ領域がなくなった際に，長時間 (数分単位) のブロッキングが発生する．なぜなら，その際にログレコードを各スレッドから集めて時系列に整列してシリアライズする必要があるからである．そして，そのブロッキングによりスループットが激減する．従って，応答時間最小化リモートロギングは本研究の目的に不適である．

2.3 追記型システム

データ修正をすべて記録するシステムとして，様々な追記型システムが考案されてきた．追記型システムではあらゆる変更が記録されるため，あるデータの昔の版を取り出すことができる．

2.3.1 追記型ファイルシステム

文献 [14] [15] では追記型ファイルシステムが実装を含めて綴られている．

文献 [14] はウェラブルコンピューティングにおけるデータ記録デバイスの開発を指向している．そして文献 [14] では，一部のデータの欠落は許容可能し，それらをメモリ上のログバッ

ファにまとめ，後で一括してディスク書き込みを実行することで，性能を向上している．同技法はデータを失う可能性がある一方で，本論文の目的はデータを確実に高速に永続化することである．それゆえ同研究は本論文の目的にそぐわない．

文献 [15] は Linux 向けのログ構造化ファイルシステムである NILFS を詳細に述べている．NILFS はデータ修正時に高々1 つのロックしか用いないために，ファイルシステム全体をロックする．それゆえ多くのセンサデータストリームを並行的に扱うと性能が大きく劣化する．一方，本論文の目的はデータを確実に高速に永続化することである．それゆえ同研究の技法は本論文の目的にそぐわない．

2.3.2 追記型データベースシステム

文献 [11] では追記型データベースシステム Tapestry が提案されているが，Tapestry は既存の RDBMS を用いて実装したことを特徴としている．既存の RDBMS のデータ挿入処理速度は，センサデータを扱うには問題がある．一方，本論文の目的はデータを確実に高速に永続化することである．それゆえ同システムの設計は本論文の目的にそぐわない．

2.4 関連研究のまとめ

本研究の目的は，頻繁に DBMS に到着するデータを確実に高速に挿入する技法を考案することにある．これを実現するためには，大量データ管理・挿入速度・永続化確実性・可変長タプル管理，の 4 つが満足される必要がある．このうち，「挿入速度」を除いた 3 項目は従来の DBMS で満足されている．

関連研究のこれら 4 項目に対する満足度を表 1 に示す．表 1 から，技法 [13] は挿入速度を改善すれば目的を達成できることがわかる．挿入速度が劣る原因は，リモートメモリが使い尽くされる際に生じるブロッキングの時間が長いことにある．そこで技法 [13] の問題である長時間ブロッキングを解決する技法を次節で提案する．

表 1　関連研究比較

従来研究	大量データ管理	挿入速度	永続化確実性	可変長タプル管理
P*TIME	×	○	○	○
ClustRa	×	○	○	○
技法 [12]	○	○	×	×
追記型システム	○	△	○	○
技法 [13]	○	△	○	○
目標	○	○	○	○

3. 準　　備

本節では課題を定式化すると共に，提案機構を述べる準備を行う．

3.1 課　　題

本研究の目的は，頻繁に DBMS に到着するデータを確実に高速に挿入する技法を考案することにある．目的の達成度を評価するために，スループット (Throughput, TP) とロギング時間 (Logging Time, LT) を測定する．TP は大きいほど望ましく，LT は小さいほど望ましい．

$$TP = \frac{DBMS\text{が処理したトランザクション数}}{\text{全トランザクションの処理に要した時間}} \quad (1)$$

$$LT = 1\text{ロギング処理に要した時間} \quad (2)$$

また，目的であるデータ挿入処理の高速化のためには，2.4 節で述べたように，リモートメモリが使い尽くされることでブロックされる時間 (Block Time, BT) の短縮化が必須となる．

$$BT = \text{ブロックされた時間} \quad (3)$$

以上をまとめ，本研究の課題を次のように定式化する．

$$\text{課題 1：}TP\text{ 極大化} \quad (4)$$

$$\text{課題 2：}LT\text{ 極小化} \quad (5)$$

$$\text{課題 3：}BT\text{ 極小化} \quad (6)$$

3.2 基 本 機 構

本論文で述べる提案機構は，我々が開発してきた DBMS である KRAFT [13] を元にして設計・実装を行った．提案機構を述べる前にその土台部分の機構を説明する．これは本論文の提案とは直接関係ないが，次節に述べる提案の理解に必要である．

3.2.1 データモデル

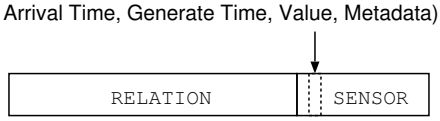

図 1　KRAFT のデータモデル

本研究ではデータモデルとして時系列関係型データモデルを用いる．図 1 にタプルの構成例を示す．

タプルは RELATION と SENSOR から構成される．タプルは可変長サイズであり，タプルに許される属性には，INT 型 (固定長 4 バイト)，TEXT 型 (可変長，バッファプールのページサイズ以下), SENSOR 型 (説明は次パラグラフ) の 3 種類がある．本論文執筆時において，タプルサイズの上限はバッファプールのページサイズ以下に制限されている．

SENSOR にはセンサオブジェクトの集合から構成される．各センサオブジェクトの構成は，センサデータ格納部 (8 バイト)，到着時刻格納部 (16 バイト)，発生時刻格納部 (16 バイト)，アノテーション用メタデータ格納部 (64 バイト) から成る．

3.2.2 ソフトウェア構成

提案機構の土台である DBMS である KRAFT のソフトウェア構成を図 2 に示す．KRAFT はロギングをリモートメモリへ実行するため，**DBMS Server** と **Log Server** の 2 つのサーバにより構成される．KRAFT は多くのモジュールから構成されているが，本論文の提案に関わるのは太線で囲まれている **Recovery Manager**，**Log Receiver**，**CheckPointer**，そして **Buffer Manager** である．

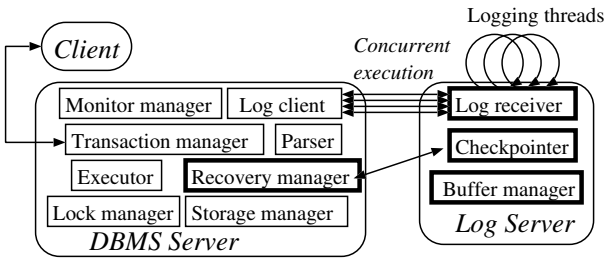

図 2　KRAFT アーキテクチャ

表 2　本論文で扱うオペレーション

コマンド名	内容
INSERT	タプル追加
DELETE	タプル削除
APPEND	センサオブジェクト追加

表 3　挿入ログレコードの形式

属性	内容
テーブル名	挿入先テーブルの名前
挿入オブジェクト	挿入するタプルのデータ
ページ情報	ファイル内オフセット情報．詳細は表 6.

表 4　削除ログレコードの形式

属性	内容
テーブル名	削除先テーブルの名前
タプル ID	削除するタプルの ID

表 5　追加ログレコードの形式

属性	内容
テーブル名	追加先テーブルの名前
属性名	追加先センサ属性の名前
センサオブジェクト	追加するセンサオブジェクト
ページ情報	ファイル内オフセット情報．詳細は表 6.

3.2.3　トランザクションモデルとオペレーション

本研究でサポートするオペレーションとしては，タプル挿入処理，タプル削除処理，センサオブジェクト追加処理を実現した．これを表 2 に示す．

また，本研究では，**1 オペレーション = 1 トランザクション**となるよう設計した．従って，クライアントにより検索されたデータは必ず永続化されていることが保証されると共に，故障後のリカバリにおいてもその存在が保証される．なぜならばセンサデータはロギングが終了してからバッファプールに置かれるために，クライアントが検索時にバッファプールのデータを読む際には，そのデータはすでに永続化されているからである．

3.2.4　ログレコード

タプル挿入，タプル削除，センサオブジェクト挿入の際に生成されるログレコードの設計を表 3，表 4，表 5 に示す．また，タプル挿入及びセンサオブジェクト挿入の際に用いられるページ情報を表 6 に示す．

3.2.5　センサデータ挿入処理

センサ受信要求が来ると **DBMS Server** では新たなスレッドが生成され，そのスレッドは **Log Server** へのコネクションを張る．同スレッドはセンサからデータ挿入を受信すると，ログレコードを生成して **Log Receiver** へ転送し，ACK を受信してからバッファプールへのデータ書き込みを行う．

表 6　ページ情報の形式

属性	内容
タプル ID	修正先テーブルの名前
属性 ID	修正先センサ属性の名前
ファイル内オフセット	ファイル内で何ページ目か
オブジェクトサイズ	挿入/追加するオブジェクトのサイズ
ページ内オブジェクト ID	ページで幾つ目のオブジェクトか

3.2.6　DBMS バッファプール

有限の大きさであるバッファプールはストレージマネージャにより管理される．もしも全てのページが使われていれば，いずれかのページを選択する．そしてそのページをディスクに書き戻し，初期化し，クライアントに提供する．

バッファプールは N 枚のバッファページから構成される．バッファページのサイズは M × PAGESIZE バイトとされる．ここで M は自然数であり，PAGESIZE は OS が提供する変数である(注1)．バッファプールのサイズを PAGESIZE 変数と合わせることでディスク I/O 速度は最適化される．

各バッファページ末尾には各ページに関するページ情報 (表 6) が記録されている．

3.2.7　リモートロギングプロトコル

本論文で対象とする KRAFT のリモートロギングのプロトコルは，2 台のログサーバへログレコードを送信して各ログサーバから ACK を受信する方式である．なお，データ永続化強度に関するリモートロギングとディスクロギングの比較は 6.1 節にて行う．

さて，DBMS サーバのログ送信プロトコルを図 3 に示す．図 3 にはリカバリ処理が記述されているが，この詳細は 4.3 節で述べる．

```
1:  if (ログを Log Server 1 へ送信 == 時間切) {
2:      Log Server 2 を用いてリカバリ処理;
3:  }
4:  if (ログを Log Server 2 へ送信 == 時間切) {
5:      Log Server 1 を用いてリカバリ処理;
6:  }
7:  if (Log Server 1 から ACK を受信 == 時間切) {
8:      Log Server 2 を用いてリカバリ処理;
9:  }
10: if (Log Server 2 から ACK を受信 == 時間切) {
11:     Log Server 1 を用いてリカバリ処理;
12: }
```

図 3　DBMS サーバのログ転送プロトコル

(注1)：我々の環境では PAGESIZE は 4096 であった．

3.3 実　　装

実装するにあたり，プログラミング言語には C 言語を用い，OS には FreeBSD 5.3Release を用いた．FreeBSD の 5 系を選択した理由は，KRAFT が KSE を用いてスレッド制御を行うからである．

4. 課題へのアプローチ

2.4 節の終わりで，KRAFT [13] は挿入速度を改善すれば，本研究の目的を達成できることを述べた．そこで本研究では KRAFT の問題である長大なブロッキング時間の極小化を実現し，本研究の目的を達成するアプローチを採る．本節ではアプローチの直観的説明を 4.1 節で与え，具体的説明を 4.2 節で与える．

4.1 提案：2 レベル非同期的ログ転送

3.1 節で述べた課題を解決するにあたりボトルネックとなるのは，リモートログサーバにおいて，リモートメモリが一杯になった際のロギング処理停止 (BLOCK 状態 (図 4)) である．リモートメモリが一杯になった時には，メモリの中身をどこかに移動するまでは BLOCK 状態からロギング実行状態 (RUN) 状態へ復帰できない．

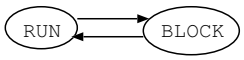

図 4　ログレシーバの取り得る状態

この BLOCK 状態を限りなく短くするために，本論文は **2 レベル非同期的ログバッファ転送機構**を提案する．提案機構は図 5 に示すように，Recoverer，Logger，BufferWriter，そして BufferTransfer の 4 種類のスレッドから構成される．Logger スレッド以外の 3 スレッドはシステム内で 1 つのみ起動され，Logger スレッドは DBMS にアクセスしているクライアントと同数だけ起動される．

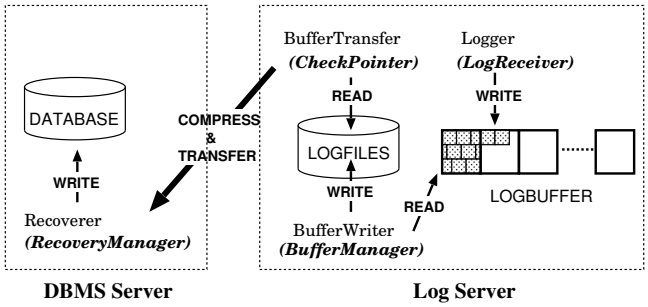

図 5　2 レベル非同期的ログバッファ転送機構

Logger はログバッファページ k を使い終わるとすぐさま $k+1$ ページを使い始める．最終ページを使い切った後は，また最初のページに戻る．BufferWriter は Logger が使い切ったページをログファイルとしてディスクに転送し，同ページを再利用可能にする．BufferTransfer はディスクに置かれたログファイルを圧縮して **DBMS Server** に転送する．**DBMS Server** の Recoverer はそれを受信，展開，REDO 処理をした後，ACK を返す．BufferTransfer は ACK 受信後，同ログファイルを消去する．

すなわち，図 5 では**非同期的ログバッファ転送**と**非同期的ログファイル転送**の 2 つの非同期ログ処理が示されている．**非同期的ログバッファ転送**は効率的に実行されなければ BLOCK 状態に陥るため BufferWriter の挙動は急峻であるべきだと考えられる．一方，**非同期的ログファイル転送**は巨大な空間であるディスクを使い果たさないほど緩慢でなければ問題は起きないため，BufferTransfer は BufferWriter よりも長い周期で動作する．

4.2 アルゴリズム

4.1 節では直観的に述べた，2 レベル非同期的ログバッファ転送機構の諸アルゴリズムを本節で示す．なお，ログページを論理的には無限長の連続ページとして扱うために，ログページはリングバッファとして設計した．

4.2.1 Logger アルゴリズム

Logger のアルゴリズムを図 6 に示す．ログレコードがログバッファページに書き込まれるときには管理用メタデータが用意され，そのメタデータに続くメモリ番地にログが書き込まれる．

```
1: カレントログページ C を 0 番目に設定 (初期化);
2: while (TRUE) {
3:     ログレコード L を受信;
4:     ログバッファ全体を施錠;
5:     C を施錠;
6:     if (C が一杯) {
7:         /* ログページ切り替え */
8:         カレントログページ C を解錠;
9:         C をインクリメント;
10:        C を施錠;
11:    }
12:    ログバッファ全体を解錠;
13:    /* L を C に書き込む */
14:    C 内のログレコード管理構造 M に L のサイズを記録;
15:    L を (M のアドレス + sizeof(M)) 番地へ書き込む;
16:    C を解錠;
17: }
```

図 6　Logger アルゴリズム

4.2.2 BufferWriter アルゴリズム

BufferWriter のアルゴリズムを図 7 に示す．BufferWriter はカレントログページ C を追いかけながらログファイルを作成し C を追い付こうとするが，決して追い越さない．BufferWriter の動作周期が十分短くないと，Logger が使えるログページが無くなってしまい，BLOCK 状態が引き起こされ，性能劣化現象が発現する．なお，本論文執筆時には同現象の検出による動作周期の動的適正化は行っていない．

4.2.3 BufferTransfer アルゴリズム

BufferTransfer はログファイル T が存在する限り，T を圧縮して Recoverer に転送する．圧縮する目的は，転送サイズを小

```
1: ダーティログページ D を 0 番目に設定 (初期化);
2: while (TRUE) {
3:     if (カレントログページ C > D) {
4:         時刻を取得して T とする;
5:         名前を T としてログファイルを作成して D を転送;
6:         D をインクリメント，ただし D がリングバッファ終端なら D を 0 に設定;
7:     }
8:     sleep(TIME_LW);
9: }
```

図 7　BufferWriter アルゴリズム

さくすることにより，ロギングに与える悪影響を小さくすることにある．我々の実験環境においては圧縮によりロギングの性能向上が観察された．なお本論文では圧縮処理を gzip ライブラリにより実現した．

```
1: while (TRUE) {
2:     while (ログファイル T が存在) {
3:         T を圧縮して DBMS Server へ転送;
4:         DBMS Server から ACK を受信;
5:         T を消去;
6:     }
7:     sleep(TIME_LT);
8: }
```

図 8　BufferTransfer アルゴリズム

4.2.4　Recoverer アルゴリズム

Recoverer は圧縮ログファイルを受け取ると，それを展開して，相対メモリ番地を利用してログを時系列順に走査しながら REDO 処理を行う．この REDO 処理に際してはデータ本体のみならずページ情報の更新も行うのだが，もしも DBMS ストレージマネージャにより書かれたページ情報が REDO 時により書き込まれる情報よりも新しければ，REDO 処理はページ情報を古くしてしまい，データベースを破壊する．そのため，その際には REDO 処理を行わない．これにより本アルゴリズムは冪等性を保証する．

4.3　リカバリプロトコル

これまではログ転送処理の高速化を述べてきた．2 レベル非同期的ログ転送を用いた際のリカバリプロトコルを図 10 に示す．ただし **Log Server N** を用いるとする．

5. 評　　価

提案手法を評価するために，TP，LT，BT を実験的に評価する．

5.1　実験環境

5.1.1　ハードウェア

DBMS サーバおよびクライアントマシンの仕様は，Pentium 4 (3GHz) CPU，4GB RAM，そして FreeBSD 5.3-Release である．ログサーバマシンの仕様は，Pentium4 (2.4GHz) CPU，1GB RAM，そして FreeBSD 5.3-Release である．

ネットワークには 100Mbps Ethernet を用い，スイッチには Gigabit Switching Hub FXG-08TE を使用した．

5.1.2　システムパラメータ

TIME_LW(図 7) は 1 マイクロ秒，TIME_LT(図 8) は 1 ミリ秒にそれぞれ設定した．

DBMS のバッファプールには 4KB ページを 32 枚用意し，ページ入れ換えアルゴリズムは FIFO を用いた．リモートログサーバのログバッファには，16KB ページを 128 枚用意した．

5.2　実験内容

5.2.1　TP(スループット)

TP を評価するために，クライアントの並行性を変えた場合の TP を，ディスクロギングとリモートロギングについて測定する．各クライアントには 100 のセンサデータ挿入トランザクションを発行させる．並行性は 100 から 600 まで 100 刻みで実験を行う．測定回数は各並行度で 5 とし，その平均値をグラフ表示する．

5.2.2　LT(ロギング時間)

LT を評価するために，クライアント数を 1 にして 100 回のセンサデータ挿入トランザクションを実行した際の，ロギングに要する時間 LT を測定する．

5.2.3　BT(ブロック時間)

BT を評価するために，ログページが一杯であった時にブロックされた時間を測定する．ただしクライアント数を 1 としてセ

```
1: while (TRUE) {
2:     圧縮ログサイズを受信;
3:     圧縮ログを展開する空間を確保;
4:     圧縮ログを受信;
5:     圧縮ログをメモリに展開;
6:     ptr = 展開アドレス先頭のログレコード管理構造;
7:     while (ptr != NULL)
8:         ptr が指すログレコードから REDO 実行 (但しページ情報については上記制約を守る)
9:         ptr = ptr->next;
10:    }
11:    確保空間を解放;
12: }
```

図 9　Recoverer アルゴリズム

```
1: DBMS Server は Log Server N にリカバリを要求;
2: Log Server N は BufferWriter および BufferTransfer を停止;
3: Log Server N はメモリ上データをログファイル化;
4: Log Server N は時系列順にログファイルを転送;
5: Log Server N は BufferWriter および BufferTransfer を起動;
```

図 10　リカバリプロトコル

ンサデータ挿入トランザクションを実行し，289 回のブロックについて測定を行う．

5.3 実験結果

5.3.1 *TP*(スループット)

TP の測定結果を図 11 に示す．同図より，リモートロギングはディスクロギングの 7.95　10.6 倍程度高いスループットを実現したことがわかる．ディスクロギングのスループットはいずれの並行度でもさほど変わらず，100 並行度ですでに飽和点に達したことが窺える．リモートロギングのスループットは 200 並行度で飽和点に達し，それ以上は過負荷状態に陥っていることが窺える．

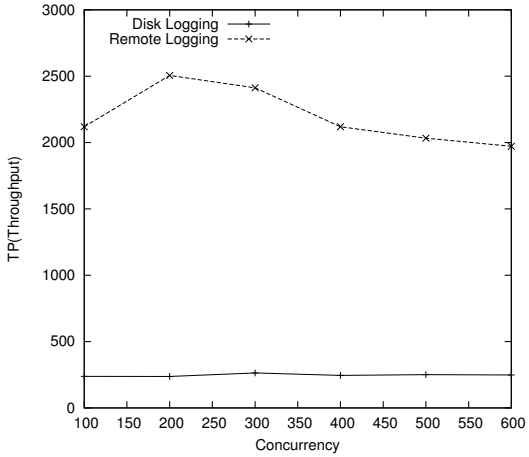

図 11　*TP*(スループット)

5.3.2 *LT*(ロギング時間)

LT の測定結果を表 7 に示す．表 7 より，リモートロギングはディスクロギングに比べて 17.2 倍ほど平均値が優れることがわかる．また，リモートロギングはディスクロギングに比べて標準偏差値が小さく，安定した性能を発揮することが窺える．一方，ディスクロギングは大きな標準偏差値をもつ．

表 7　*LT*(ロギング時間)

測定値	ディスクロギング	リモートロギング
LT 平均値 (μs)	3557	206
LT 標準偏差値 (μs)	5049	26
LT 最小値 (μs)	1284	183
LT 最大値 (μs)	35814	308

5.3.3 *BT*(ブロック時間)

BT の測定結果を表 8 に示す．同表より，*BT* は平均 4.38 マイクロ秒という極めて小さい値に収まっていることがわかる．

表 8　*BT*(ブロック時間)

BT 平均値 (μs)	4.38
BT 標準偏差値 (μs)	0.51
BT 最小値 (μs)	4
BT 最大値 (μs)	5

5.4 考　　察

5.4.1 *TP*(スループット)

並行性の増大につれてリモートロギングのスループットが劣化する理由には TCP_NODELAY オプションの設定が関係した可能性がある．この設定により TCP プロトコルスタックはバッファリングを行わずに，send システムコール経由で得たデータを即座にネットワークへ出力する．それゆえ応答時間は改良されてもスループットが劣化した可能性がある．

この現象を解決するにはグループ送信が有効だろう．なぜなら，グループ送信は TCP プロトコルスタックからのパケット出力回数を減らすのみならず，負荷の大きい処理であるシステムコールの呼出し回数を減らすからである．

5.4.2 *LT*(ロギング時間)

本実験では，リモートロギングの *LT* の 90%以上はネットワークに占められた．それゆえこれ以上の大幅な性能向上はアルゴリズム的には困難であり，高速なネットワークデバイスの利用が必要だと考える．

5.4.3 *BT*(ブロック時間)

TP と *LT* が優れた結果を示した理由は *BT* が極めて小さかったことによる．非同期的処理を 2 段階にしたことが高性能の理由であろう．

6. 議　　論

6.1 リモートメモリ上へログが書き込まれた時点でロギングが終了と考えられる理由

データを消失させないため，DBMS は作成したログを永続記憶装置に記録したのち持続記憶装置の更新を行う [14]．永続化処理の高速化のためにはディスクに代わる永続記憶装置の利用が必要になる．

そこで我々は 2 台のリモートマシン上のメモリを永続メモリとする方式を提案してきた [6] [12] [13]．リモートメモリに置いてあるログレコードはマシン故障時に消失する．しかし我々の提案方式はリモートホストを 2 台用いるため，片方のマシンが故障しても，もう一台はマシン上にログレコードは存在する．

そしてロギング実行時にマシンから ACK が返らなければ故障が検出される．その時にはリカバリ処理が実行されてから，利用可能なリモートマシンが新規に 2 台検出され，永続化処理が再開される．そして 2 台のマシンが同時に故障する確率は，客観的に低いと考えられる．また，文献 [6] では同様にログサーバを 2 台用いていることから，2 台のリモートログサーバとローカルディスクの耐故障性と同じと見做しても良いと筆者は考える．

従って，2 台のリモートメモリ上へログを書き込むことができれば，データベースは一貫性を失わずに済むと筆者は考える．以上がリモートメモリ上へログが書き込まれた時点でロギングが終了と考えられる理由である．筆者の考えを検証するために，長期間にわたる運用試験を今後予定している．

6.2 提案手法とディスクロギングとの比較がフェアである理由および比較する意味

本論文の目的は，連続的にシステムに到着するデータを全て

格納することができる高性能ロギング方式を提案することにあった．筆者の知る限り，これを実現するのは筆者の提案を除けばディスクロギングしかない．MMDB におけるロギングは 2. 節で述べたように，更新には適切だが，挿入処理には不適切である．従って MMDB のロギング方式と本論文の提案方式を比較することはフェアではない．それゆえ筆者で提案手法とディスクロギングとの比較はフェアであると考える．そして連続的にシステムに到着するデータを全て格納することができるロギング方式にはディスクロギングと提案方式しか考えられないため，両者を比較したことには性能を比較するという意味があった．

6.3 ストリームやセンサデータは保存する価値があるか？

ストリームやセンサデータが貴重か否かは立場によろうが，今後は重要性が増すと筆者は考えている．なぜならデータベースの使われ方がこれから変化すると考えるからだ．これまでの DB には人間が必ず使うデータが恣意的に格納され，DBMS の負荷は検索トランザクションが主であった．

しかし，今後はストレージの巨大化により，データ到着時には重要性のわからないデータをさしあたり DB に格納しておき，後で検索を行うようになると筆者は考えている．従って，一見無価値のように思えるストリームやセンサデータさえも瞬時に格納可能な DBMS アーキテクチャの重要性は増大すると，筆者は考える．

6.4 通信プロトコル

本研究では通信プロトコルに TCP を用いた．他に考えられるプロトコルには，UDP，T/TCP, あるいはゼロコピー通信の利用が考えられる．しかしこれらには，信頼性，プログラミングのしにくさ，低性能のいずれかが関係する．T/TCP はコネクション生成とデータ送信をいちどに行うため，予備評価にて TCP よりも低い性能しか得られなかった．UDP やゼロコピー通信は信頼性に問題がある．さらに，多重化及び信頼化処理を考えると，プログラミングが困難であることから，高性能を出せる可能性が低いと筆者は考える．従って本研究では通信プロトコルとして TCP を選び，応答時間を向上させるために TCP_NODELAY を指定して Nagle アルゴリズムをオフにした．

6.5 実験結果の意義

実験結果より，本研究により改善されたリモートロギングは平均 206 マイクロ秒のロギング時間を得た．このデータの意味について考える．一般的なロボットは 1 ミリ秒周期で，モーターデータ取得およびデータ解析を行う．それゆえロギング時間が 206 マイクロ秒ならば，少なくともロボットのモータデータ記録という，これまでは全く DBMS が使われることのなかった分野へ，DBMS を適用することが検討可能になる．

7. 結　　論

本研究の課題は，(1)TP 極大化，(2)LT 極小化，(3)BT 極小化，であった．提案機構は，ディスクロギングに比べて (1) は 10 倍程度，(2) は 17 倍程度優れた結果を示した．(3) は平均 4.38 マイクロ秒という極小値を得た．以上より提案機構は研究課題をほぼ解決したと考えられる．そして，実験が KRAFT 上で行われたことから，表 1 における「目標」欄が達成されたとの結論が示唆されよう．

文　　献

[1] Brian Babcock, Shivnath Babu, Mayur Datar, and Rajeev Motwani. Chain: Operator scheduling for memory mininization in data stream systems. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 253–264, 2003.

[2] Don Carney, Ugur Cetinternel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Monitoring Streams - A New Class of Data Management Applications. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

[3] Sang K. Cha and Changbin Song. P*TIME: Highly Scalable OLTP DBMS for Managing Update-Intensive Stream Workload. In *Proceedings of 30th International Conference on Very Large Data Bases*, pp. 1033–1044, 2004.

[4] Sirish Chandrasekaran and Michael Franklin. Remembrance of streams past: Overload-sensitive management of archived streams. In *Proceedings of 30th International Conference on Very Large Data Bases*, pp. 348–359, 2004.

[5] Jim Gray and Andreas Reuter. トランザクション処理，概念と技法, 第 13 章. 日経 BP, 2001.

[6] Svein-Olaf Hvasshovd, Øystein Torbjørnsen, Svein Erik Bratsberg, and Per Holager. The ClustRa Telecom Database: High Availability, High Throughput, and Real-Time Response. In *Proceedings of the 21th International Conference on Very Large Data Bases*, pp. 469–477, September 1995.

[7] Juchang Lee, Kihong Kim, and Sang K. Cha. Differential Logging: A Commutative and Associative Logging Scheme for Highly Parallel Main Memory Databases. In *Proceedings of the IEEE International Conference on Data Engineering*, pp. 173–182, 2001.

[8] Kwei-Jay Lin, Swami Natarajan, and Jane W.-S.Liu. Concord: A System of Imprecise Computation. In *Proceedings of the Eleventh Annual International Computer Software and Applications Conference (COMPSAC 87)*, pp. 75–81, 1987.

[9] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The Design of an Acquisitional Query Processor for Sensor Networks. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 491–502, 2003.

[10] C. Mohan. Repeating History Beyond ARIES. In *Proceedings of 25th International Conference on Very Large Data Bases*, pp. 1–17, 1999.

[11] Douglas Terry, David Goldbery, David Nicholas, and Brian Oki. Continual Queries over Append-Only Databases. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 321–330, 1992.

[12] 川島英之, 遠山元道, 今井倫太, 安西祐一郎. リモートメモリを用いたセンサデータストリームの永続化. 情報処理学会論文誌データベース (TOD19), Vol. 44, No. SIG12, pp. 98–109, 9 2003.

[13] 川島英之, 今井倫太, 遠山元道, 安西祐一郎. センサデータベースシステム KRAFT の設計と実装. 情報処理学会論文誌：データベース, Vol. 45, No. SIG14(TOD24), pp. 39–53, 2004.

[14] 小野田英樹, 波多野賢治, 宮崎純, 植村俊亮. ウェラブルコンピューティングのための追記型ファイルシステムの実装. In *DEWS2004*, 2004.

[15] 天海良治, 一二三尚, 小西隆介, 佐藤孝治, 木原誠司, 盛合敏. Linux 用ログ構造化ファイルシステム nilfs の設計と実装. 情報処理学会研究報告, pp. 61–68, 2005.