4L−7

データベース管理システムのためのベースシステムの実現

早田宏、木内康彦、小松原弘文、佐藤直人
富士ゼロックス(株) システム技術開発センタ

1.はじめに

近年、オブジェクト指向データベースをはじめとし、新たなデータベース管理システム(DBMS)の研究や開発が行われ、その実装が盛んに行われている。しかし、DBMS実現のためのライブラリは整備されておらず、その実現に多大な労力が費やされている。DBMS実現のためのライブラリをベースシステムとして整備し、実現したので報告する。本ベースシステムは実験的なDBMSの実現においても、実用的なDBMSの実現においても、有効なライブラリである。

2.想定するデータベース管理システム

本ベースシステムはDBMS実現のための各種機能を提供するが、ある仕様のDBMSを想定して実現されている。本ベースシステムの想定するDBMSの仕様は以下の通りである。

- UNIXワークステーションで実現
- マルチプロセスで構成
- マルチプロセスは1つのアドレス空間を共有
- 各プロセス毎にプロセス固有な専有領域
- C系の言語で実装

3.ベースシステムの構成と実現

3.1 ベースシステムの構成

本ベースシステムはプロセス管理部、入出力部(ネットI/O部及びディスクI/O部)、ランタイムシステム部からなる(図1参照)。プロセス管理部は、プロセスの生成/消滅/待機ならびに排他制御のためのインタフェースを提供し、マルチプロセス間での共有アドレス空間ならびに各プロセス毎の専有領域を実現する。入出力部はバイトストリームのネットワーク入出力ならびに高速なディスク入出力を実現する。ランタイムシステム部は、非局所飛び出し、エラーシステム、ハッシュテーブルなどの各種データ構造を実現する。

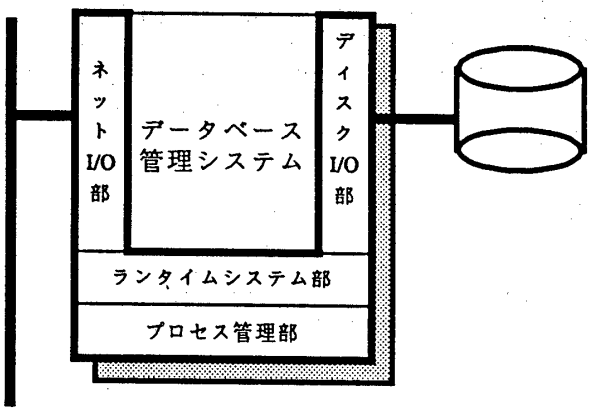

図1.ベースシステムの構成

3.2 プロセス管理部

プロセス管理部でマルチプロセス間での共有アドレス空間ならびに各プロセス毎の専有領域を提供する。この実現には、各種システムコールを利用する方式と所謂ライトウェイトプロセスライブラリを利用する方式とがある。前者では、各プロセスはUNIXの1プロセスとして実現され、共有アドレス空間(共有領域)は共有メモリ、プロセス毎の専有領域は各プロセスの領域、排他制御はセマフォを利用する。この方式では、プロセス毎の専有領域と共有領域の大きさを自由に設定できるが、プロセススイッチングや排他制御の性能に問題がある。後者では、各プロセスは1スレッドとして実現され、プロセスのアドレス空間はすべて共有となり、プロセス毎の専有領域はスレッドのスタック上にスペシャルコンテクストとして実現され、排他制御はモニタや条件変数によって実現される。ライトウェイトプロセスライブラリとしてSun Light Weight Processライブラリ[1]を利用すると、1プロセスで共有領域の資源を排他制御しながらアクセスする性能評価で後者は前者の約7倍の性能であった。ベースシステムでは、後者の方式をとり、ライトウェイトプロセスライブラリとしてSun Light Weight Processライブラリを利用する。

3.3 入出力部

入出力部はバイトストリームのネットワーク入出力ならびに高速なディスク入出力を実現する。ネットワーク入出力はTCP/IPのインターネットコミュニケーションをもとに実現している。ネットワーク入出力はSun Light Weight ProcessのNon Blocking I/Oライブラリを利用して実現している。

Implementation of Base System for Database Management System

Hiroshi HAYATA, Yasuhiko KIUCHI,
Hirofumu KOMATSUBARA, Naoto SATO
Fuji Xerox Co., Ltd. STDC

ディスク入出力部はページ単位で入出力を行う。UNIXでは、高速かつ高信頼のディスク入出力としてraw I/Oを提供している。しかし、raw I/Oを利用すると、入出力要求においてそのプロセス全体がブロックする問題がある。ベースシステムでは、ノンブロッキングでかつ非同期な入出力を高速かつ高信頼におこなうために、raw I/Oを専用に行う入出力プロセス(IOP)を設ける。DBMSスレッドはIOPへディスク入出力をノンブロッキングで要求し、IOPはraw I/Oを使って実際のディスク入出力を行い、DBMSスレッドへ入出力結果を送る。

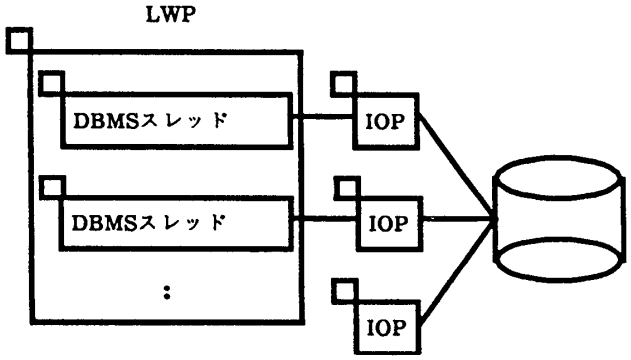

図2.ディスク入出力部の構成

3.4 ランタイムシステム部

ランタイムシステム部は、非局所飛び出し、エラーシステム、各種データ構造を実現する。この仕様は主にCommonLispでの言語仕様[2]に影響されている。非局所飛び出しはUNIXにおいてsetjmp/longjmpとして提供されているが、非局所飛び出しが行われた際の後処理の記述はできない。ベースシステムではCommonLisp風のcatch/throw/unwind_protectを提供する。

```
catch(tag_name, return_type, body)
throw(tag_name,result_type,v_count,value)
unwind_protect(result_type,protected_form,cleanup_form)
    /* cleanup_formはprotected_formの正常終了時にも非局
    所飛び出し発生時にも必ず実行 */
```

非局所飛び出しの管理のための制御スタックを設ける。スタックの要素は1つのcatchまたはunwind_protectに対応する。この制御スタックは各DBMSスレッド毎に固有であるため、スタックのアドレスはスレッド毎の専有領域で管理される。

エラーシステムはエラー条件をハンドリングするための機構である。DBMSでエラーを発生させると、そのエラーの種類に応じてエラーオブジェクトを生成し、ある位置で特定のエラーをキャッチして、特定の処理を実行させる。発生したエラーオブジェクトは各DBMSスレッド毎に固有であるため、発生したエラーオブジェクトへのポインタはスレッド毎の専有領域で管理する。ベースシステムではインタフェースとして以下のものを提供する。

```
error(error_type)
handler_case(body,error_type1,handler1,error_type2,...)
    /* handler1はerror_type1のエラーが発生した場合実行 */
```

データ構造として、配列、リスト、ハッシュテーブルを提供する。配列はC系の言語においてもサポートされているが、領域の拡張を自動的に行うような配列が有効である。また、不定長の要素を効率良く実現する手段としてリストは有効である。リストの不要となった要素の自動回収はベースシステムではサポートしていない。明示的な解放をするインタフェースは提供している。ハッシュテーブルはDBMS内の1対1の対応をもつ資源の管理で有効なデータ構造である。

4. ベースシステムの利用

サーバクライアント方式のDBMSのサーバを本ベースシステムを利用して実現した。サーバは論理的なファイルシステムとページ単位のファイルアクセスをクライアントに提供している。ページ単位のトランザクション処理、クライアントでのページのキャッシング管理もサーバで行う。

トランザクションの並列制御はページ単位のtwo-phase lockingで実装されている。デッドロックは自動的に検出され、再実行される。トランザクションのアトミック性はログを利用して実現される。スレッドによって実現されたバックグラウンドプロセスにより、コミットしたトランザクションの更新内容がデータベースファイルに伝播する。

5. おわりに

本論文では、DBMSのためのベースシステムの実現について述べた。ベースシステムの有効性は、その利用によって実証されている。しかし、本ベースシステムはDBMS全体からみると、最下層に位置する。DBMS自体はまだまだ複雑であるため、DBMSのある機能の実験や改良を簡単に行うことは容易ではない。オブジェクト指向データベースなどの新たなDBMSの実装実験のためには、DBMSの部品化、実装のレイヤリングが必要である。

参考文献

[1] SunOSマニュアル System Servives Overview (1988)
[2] Guy L. Steele Jr. : COMMON LISP THE LANGUAGE, Digital Press (1990)
[3] Daniel Weinreb, et al: An Object-Oriented Database System to Support an Integrated Programming Environment, Data Enigineering Vol.11 No.2 (1988)
[4] C. J. Date : An Introduction to Database Systems, Addison-Wesley (1987)