拡張可能ＤＢＭＳにおける部品の管理と呼び出しの一方法

増永浩二† 宝珍輝尚† 都司達夫†

拡張可能データベース管理システムでは，利用分野ごとに特化した処理をカーネルの一部として実行させたいという要求がある．しかし，ＤＢＭＳを柔軟にし必要な機能を付加可能とすると，そのためのオーバーヘッドが生じてしまうという問題がある．本論文では，ダイナミックリンクライブラリを用いて部品を使用した際に発生する実行時性能オーバヘッドについて考察し，(1)部品の性能要求を満たすための複数のハッシュを用いた部品の管理法，ならびに，(2)部品使用時のハッシュ値の衝突を禁止して部品情報の検査を省いた高速実行と，部品情報の検査を行なう通常実行という２種類の部品の呼び出し法を提案する．実機による部品実行性能評価の結果，ハッシュ関数を複数用い，第ｎのハッシュエントリ中にキーに加えて第 n+1 のハッシュ値を格納し衝突時のキーの比較をこのハッシュ値の比較で代用するのは有効であることを明らかにした．これは，ハッシュのキーである部品情報の比較処理がハッシュ値の比較よりも重いためである．また，高速実行は通常実行に比べ処理時間が短縮されるので高速実行は有効であることを明らかにした．

A Method of Managing and Invoking Software Parts in an Extensible DBMS

KOJI MASUNAGA,† TERUHISA HOCHIN† and TATSUO TSUJI†

In an extensible DBMS, an application-specific function is required to be executed as a part of the DBMS kernel. The overhead in invoking DBMS parts can not be neglected when DBMS is made flexible by enabling application-specific functions to be added to the DBMS. This paper studies the run-time overhead in invoking DBMS parts through the Dynamic Link Library supported by operating system. This paper proposes the method of managing DBMS parts by using more than one hash function. As the collisions of hash values seldom occur, the performance overhead in invoking DBMS parts can be decreased. This paper also proposes the quick invocation of DBMS parts. A DBMS part is invoked without any checks about it in the quick invocation. The invocation with checks about the DBMS part is called the normal invocation for the purpose of distinguishing it from the quick one. In order to make the quick invocation possible, the DBMS part must be registered into the system without any collisions of hash values made from the information of the part. The evaluations of the execution times of DBMS parts clarify the benefits of the proposed methods as follows. It is effective to use more than one hash function in order to decrease the performance overhead. This is caused by storing $n+1$th hash values as well as keys, which are the information of parts, into nth hash entries, and comparing the $n+1$th hash values rather than the keys. The reason is that comparing keys is more expensive than comparing hash values. The quick invocation is effective because the execution time through the quick invocation is about 70% of that through the normal invocation.

1. はじめに

最近，CAD/CAM，LSI設計，CASE，ネットワーク管理，マルチメディアシステムなど，様々な分野でデータベースを利用したいという要求が強まっている．ところが，従来の帳票処理を対象としたＤＢＭＳでは様々なデータベースの応用分野に対応できず，CADデータやビデオデータなどを管理するには，それらの分野に適したＤＢＭＳが望まれる．しかし，それぞれの分野用にＤＢＭＳを構築することは，製作の工数やコストから見て現実的ではない．そこで，ＤＢＭＳを拡張可能としておき，必要な機能のみを作成・付加することで，それぞれの分野に特化したＤＢＭＳを迅速に構築しようという拡張可能ＤＢＭＳの研究が行なわれてきている[1)~18)]．

拡張可能ＤＢＭＳの研究アプローチは，大きく，ＤＢＭＳを構成するモジュールを組み合わせることでカ

† 福井大学工学部
Faculty of Engineering, Fukui University

スタムDBMSを作り上げるDBMSジェネレータ／ツールキットアプローチと，DBMSの機能を全て持ったDBMSに機能追加を可能とする完全機能DBMS拡張アプローチに分けられる[1)2)]．著者らが開発中のCOMMONはDBMSジェネレータ／ツールキットアプローチを採る拡張可能DBMSである[18)]．COMMONでは，システムをかなり粒度の細かいモジュールに分割し，演算の統一化を図ることで，モジュールの理解を向上させようとしている．

しかし，COMMONでは，DBMSに部品の追加・削除を行なう場合に，DBMS本体を再コンパイルしなければならず，システムの停止・再起動を必要とする．このようなことが頻繁に起こり得るDBMSではとても実用には耐えられない．また，必要な機能を付加可能としているためオーバヘッドが生じてしまう．特にDBMSの場合，データ数に応じた処理が必要となる機能に関しては，1呼び出し当りの性能オーバヘッドが大きくなくてもトータルでは大きなオーバヘッドとなることがあり，性能の良いDBMS部品の実行が求められる．

そこで，本論文では，高度な機能拡張を可能としながら，DBMSとして実行性能の良い拡張可能DBMSの実現を目的として，DBMS部品を管理するプログラム管理部における部品管理法，ならびに，部品呼び出し法を提案する．プログラム管理部では部品情報を管理し，ダイナミックリンクライブラリを用いて部品の実行を行なう．部品情報はハッシュを用いて管理するが，ハッシュ値の衝突が性能劣化を引き起こす．そこで，複数のハッシュを用いることで，なるべくハッシュ値の衝突が発生しないようにする．また，部品呼び出し時には，ハッシュ値の衝突の可能性があるため，部品情報の確認が必要であるが，これが大きなオーバヘッドとなりうる．そこで，ハッシュ値が衝突する場合は部品の登録を許可せず，その代わり，呼び出し時には部品情報の確認なしで部品を呼び出す高速実行を，通常の実行と併用することにする．これにより，データ数に応じた部品呼び出しが必要であるといった，少しでも部品実行のオーバヘッドを減らしたい場合に対処可能とする．

以下，2で拡張可能DBMS：COMMONについて概説し，3でプログラム管理部の設計について述べる．4で部品の使用方法を示し，5で提案法を用いた部品実行の性能評価を行なう．6で関連する研究について言及し，最後に，7でまとめる．

2. 拡張可能DBMS：COMMON

2.1 COMMONの概要

あるシステムを拡張可能とするためには，システムをいくつかのモジュールに分割し，モジュールの機能を単純化する方法が一般的である．DBMSでもこの方法が適用できる．COMMONでは，データベースの管理に必要な様々なデータの各々を別々のデータベースとして管理するという考え方に基づいてモジュールの分割をしている[18)]．あるモジュールでは，ある特定のデータを管理し，データ操作の基本はSELECT, INSERT, DELETE, UPDATEとしている．データ操作をこのような基本的な演算に留めることで，モジュールの大きさを限定するとともに，モジュールを理解しやすくすることで，DBMS構築者の負荷を軽減している．

この方針に従って得られたモジュールをDBMSコアモジュールと呼び，その機能は部品によって実現される．各モジュールには少なくとも一つの部品があり，部品は手続きの集まりである．各手続きはSELECT, INSERT, DELETE, UPDATEといった基本的な演算を実現する．モジュールが複数の部品を持っている場合，それらの部品は互いに代替部品として機能する．

基本的な拡張方法は，モジュールに用意されている部品を選択することである．モジュールの機能はアプリケーションプログラムに最適な部品を選択することによって変更することができる．適切な部品が無い場合は，必要とする機能を持つ新しい部品を作成しなければならない．

2.2 要　　求

拡張可能DBMS：COMMONは，前述したように高い拡張性を備えている．しかしながら，実際に機能を拡張する際にはいくつかの問題点がある．以下に問題点を示す．

- 現在，部品の追加・削除にはDBMS本体を再コンパイルし，稼働しているDBMSを停止させ，新しいDBMSを再起動させる必要がある．DBMSの停止・再起動が頻繁に起こり得るようでは，とても実用には耐えられない．
- DBMSを拡張可能とすると，そのためのオーバヘッドが生じてしまう．特にデータ数に応じた処理が必要な機能に関しては，1呼び出し当たりの性能オーバヘッドは大きくなくても，トータルでは大きなオーバヘッドとなるという問題がある．

以上の問題から，以下の要求が得られる．

要求1　新たな機能をシステムの停止・再起動させる

こと無く，動的に追加・削除できること．

要求2　性能要求の厳しい部品は，実行オーバヘッドを少なくすること．

これらの要求を実現するため，COMMONにおいてその機能を担当するプログラム管理部の設計・実装を行なった．次にこれについて述べる．

3. プログラム管理部

要求1は，ダイナミックリンク機能を用いることで，動的な機能拡張を可能とし，要求2に対しては，部品の管理，呼び出し方法を細かく制御することによって，実行性能オーバヘッドを抑制する．以降で，プログラム管理部について詳しく述べる．

3.1 設　　計

プログラム管理部の主な構成部分を以下に示す．

- インターフェース部分
- プログラム管理部の初期化・終了処理の部分
- 部品情報を登録・削除する部分
- 部品の処理に関する部分
- 部品の動的な更新の部分
- サーバ(DBMS)の更新の部分

以下，これらについて説明する．

3.1.1 インターフェース部分

COMMONにおいては，システム本体と，モジュールとのインターフェースは，その形がほぼ統一されている．プログラム管理部もその形に従っている．インターフェース部分では呼び出し側から2つの引数を受け取る．第1引数は，そのモジュールに対する操作の種類であり，第2引数は，そのモジュールが使用する引数が格納された構造体へのポインタである．インターフェース部分では，第1引数によって操作の種類を判断し，必要な関数を呼び出す．その際に，呼び出した関数への引数として，第2引数を与える．

モジュールに対する操作の種類を**表1**に示す．

3.1.2 プログラム管理部の初期化・終了処理の部分

プログラム管理部の初期化の部分では，変数の初期化と，部品の初期登録を行なう．部品の初期登録は，初期設定用のファイルが存在する場合のみ行なわれる．初期設定用のファイルには，部品情報が記されており，ファイルから部品情報を読み込み，登録，オープンの処理を行なう．

終了処理の部分では，まず，その時点でオープンされている部品を全てクローズする．次に，新たに初期設定用のファイルを作成し，登録されている部品情報をファイルに書き込む処理を行なった後，システムを終了させる．これにより，システムを再起動させたときに，終了した時点の状態から再開することができる．古くなった初期設定用のファイルは，初期化の部分で消去される．

表1　プログラム管理部の操作の種類
Table 1　Operations of program manager.

操作の種類	機能
INIT	プログラム管理部の初期化
TERM	プログラム管理部の終了処理
INSERT	部品情報の登録
DELETE	部品情報の削除
UPDATE	部品の動的な更新
OPEN	部品のオープン
EXECUTE	部品の通常実行
FEXECUTE	部品の高速実行
CLOSE	部品のクローズ
NEWMAKE	DBMS 更新用 MAKEFILE の変更
NEWEXEC	DBMS の動的再起動
ALTERLINK	リンクタイプ変更の設定

表2　部品情報の種類
Table 2　Part information.

部品情報の種類	内容
manager	部品の属するマネージャ名
parts	部品名
type	部品の分類名
path	オブジェクトファイルがあるパス
function	部品に対応する関数の名前
linktype	部品のリンクタイプ(動的／静的)
entrytype	部品の登録タイプ(通常／高速)

3.1.3 部品情報を登録・削除する部分

機能の説明の前に，まず部品情報について説明する．部品情報の種類を**表2**に示す．

部品の属するマネージャ名，部品名，部品の分類名の3つの名前は，部品の識別で使用され，オブジェクトファイルがあるパス，部品に対応する関数の名前，部品のリンクが動的なものか静的なものかの3つは，ダイナミックリンク機能を用いるときに使用される．部品の登録タイプとして，通常実行登録と高速実行登録を設けている．通常実行登録と高速実行登録については**3.1.4**の部品の実行で詳述する．

部品情報を登録する部分では，ハッシュを用いて部品情報を登録している．部品情報の登録で使用する部品の属するマネージャ名，部品名，部品の分類名の3つの文字列の文字を加算し，それをハッシュテーブルのサイズで割った余りをハッシュ値として使用する．まず登録タイプが，通常実行登録か高速実行登録かどうかを調べる．通常実行登録の場合，登録したい部品情報の部品の属するマネージャ名，部品名，部品の分類名の3つ全てが既存のものと同じものが無いかどうかをチェックしてから登録する．高速実行登録の場合

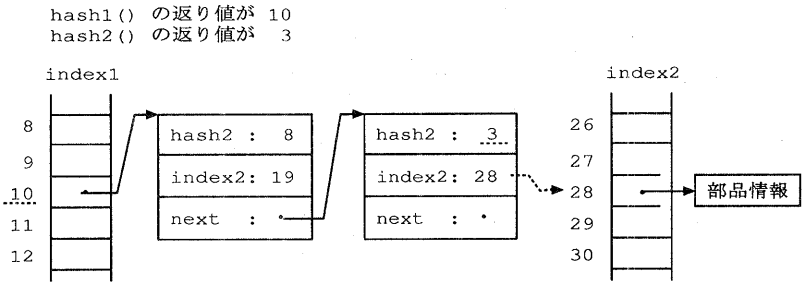


図 1 複数のハッシュを用いた部品情報の管理
Fig. 1 Management of the information of DBMS parts through two hash functions.

は，ハッシュ値を計算して，その値が既存のものと同じになった場合には登録を許可しない．したがってこの場合，部品の属するマネージャ名，部品名，部品の分類名のいずれか，または，すべてを変更する必要がある．問題が無ければ，部品情報が登録される．

部品情報を削除する部分では，削除したい部品が登録されているかどうかチェックしてから削除する．その部品がオープンされたままであった場合には，クローズしてから部品情報を削除する．

3.1.4 部品の処理に関する部分

部品の処理に関する部分では，ダイナミックリンクライブラリを用いて，部品のオープン，クローズ，実行の処理を行なう．

部品のオープンでは，まず，部品情報が登録されているかどうかをチェックする．次にリンクのタイプを調べ，動的なリンク，静的なリンクのそれぞれに対応した方法でオブジェクトファイルをオープンする．オープンに成功したならば，部品に対応する関数へのポインタを求め，部品情報に登録する．この時点では，部品はまだメモリ上に配置されず，部品が初めて実行されるときに配置される．部品のクローズ要求が来るまで部品はメモリ上に配置される．

部品のクローズでは，部品情報が登録されているかどうかを調べ，その後，オープンの処理のときに得た値である **handle** に正常な値が入っているかどうか調べてから，部品をクローズする．

部品の実行に関しては，要求2を満足するため，部品の実行を通常実行と高速実行の2種類に分けている．

通常実行では，使用したい部品の部品情報(引数)からハッシュ値を求め，その値を基に，登録されている部品情報にアクセスする．ここでは，ハッシュ値の衝突を極力避けるために，複数のハッシュ関数を用いている．**図 1** にその方法を示す．図 1 では，第1ハッシュ値が 10，第2ハッシュ値が 3 の場合を示している．第1ハッシュ値を用いて，index1 のエントリにアクセスし，リストをたどって，第2ハッシュ値を持つエントリを捜す．このエントリの index2 の値を用いて index2 のエントリにアクセスし，求める部品情報を得る．その後，部品情報が正しいか検査し，正しければ部品を実行する．

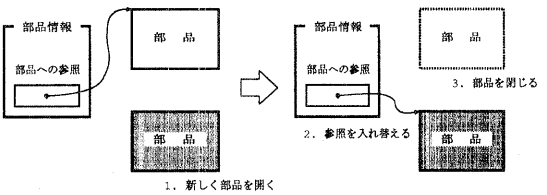


図 2 部品の動的な更新
Fig. 2 Dynamic reconfiguration of software parts.

高速実行の場合は，前述の通り，登録の際にハッシュ値が既存のものと衝突した場合，登録を許可しない．ハッシュ値に衝突が無いことを保証するので，部品の属するマネージャ名，部品名，部品の分類名の3つが確かに合致しているか検査する必要は無く，この検査を行わずに部品を実行する．

3.1.5 部品の動的な更新の部分

部品の動的な更新の処理は，既に登録されている部品を，部品の属するマネージャ名，部品名，部品の分類名の変更なしに，新しい部品に入れ替える機能である．**図 2** に処理のイメージを示す．ここで変更される部品情報は，オブジェクトファイルがあるパスと，部品に対応する関数名のみである．

3.1.6 DBMS (サーバ) の更新の部分

前述した機能によって，DBMS機能の動的な追加は可能になったが，DBMS (サーバ) そのものに対しては，拡張・変更の手段が無く，依然として柔軟性が低いままである．例えば，ダイナミックリンクで登録してある部品を静的にリンクしたい場合や，DBMSの基本的な部分を変更したい場合には，結局DBMSを停止させることになる．しかし，この場合にもシステムを実質的には停止させること無く，新しいシス

テムを起動させる機能を実現した．

ダイナミックリンクで登録してある部品を静的にリンクしたい場合には，まず，静的なリンクにしたい部品に，リンクタイプを変更するための設定をする．この機能は，変更したい部品が登録されているかどうかチェックした後，部品情報に変更後のタイプを記入する．静的なリンクから動的なリンクに変更したい場合にも，この機能が利用できる．部品を使用するために必要な情報に変更を加えるわけではないので，この処理を行なっても，部品の使用には何の支障もない．変更したい部品の全てにこの処理を行なった後，ＤＢＭＳ (サーバ) 更新用 MAKEFILE ☆の変更を行なう．ここで，もともと静的にリンクされている部品は，そのまま静的にリンクされるよう，動的なリンクから静的なリンクに変更したい部品も，静的にリンクされるよう，MAKEFILE に変更が行なわれる．変更が終了した後，その MAKEFILE を使用して，新しいシステムをコンパイルし，システムの再起動処理に入る．

システム再起動の処理は，それまでにＤＢＭＳが受け付けていたトランザクションが全て終了した後，新たなトランザクションを受け付ける前に行なう．

システム再起動の処理では，プログラム管理部の終了処理をした後，コンパイルによって新しく出来上がった実行ファイルを，exec 関数によって起動させる．起動に失敗した場合は，それまでに稼働していたシステムを再起動させる．

4. 部品の使用例

部品を登録・使用するときには，プログラム管理部に引数として，プログラム管理部の操作の種類と，部品情報を格納した構造体のポインタを渡す．プログラム管理部の操作の種類は表 1 の通りであり，部品情報の内容は表 2 の通りである．

図 3 に，部品の使用例を示す．この例では，“datamng” というモジュールに “fetchindex” という部品を登録し，オープン，実行，クローズしている．

5. 評　　価

プログラム管理部を用いた部品実行の性能評価を行なった．ここでは，以下の実行性能を測定した．

- データの挿入・検索にかかる処理時間
- 部品情報の，部品の属するモジュール名，部品名，部品の識別名の総文字数の違いによる部品選択時間
- 部品選択時の衝突回数の違いによる部品選択時間

☆ この MAKEFILE はもともと，これを使ってコンパイルすれば，現在稼働中のシステムが作成できるような記述がされている．

```
/* 部品の登録 */
pmarg1.manager   = "datamng";
pmarg1.parts     = "fetchindex";
pmarg1.path      = "../../datamng/datamng.o"
pmarg1.function  = "datafetidx"
pmarg1.entrytype = NORMAL;
/* pmarg1.entrytype = FAST で高速実行登録 */

progmng( INSERT, &pmarg1 );

/* 部品のオープン */
pmarg2.manager   = "datamng";
pmarg2.parts     = "fetchindex";
pmarg2.entrytype = NORMAL;

progmng( OPEN, &pmarg2 );

/* 部品の実行 */
pmarg3.manager  = "datamng";
pmarg3.parts    = "fetchindex";
pmarg3.pknd     = INIT;
pmarg3.parg     = &dataarg;
    /* dataarg は部品が使用する引数を格納した構造体 */
progmng( EXECUTE, &pmarg3 );
/* progmng( FEXECUTE, &pmarg3 ); で高速実行 */

/* 部品のクローズ */
progmng( CLOSE, &pmarg2 );
```

図 3　部品の使用例
Fig. 3　An example of using a DBMS part.

以上は，部品情報をアクセスする際に用いるハッシュ関数に影響されるので，ここでは，(1)**3.1.4** で述べたように，部品の属するモジュール名，部品名，部品の識別名の全ての文字を加算し，それをハッシュテーブルのサイズで割った余りをハッシュ値とする関数 (通常ハッシュ関数) と，(2) 部品の属するモジュール名，部品名，部品の識別名の中から 4 文字をピックアップして加算し，ハッシュテーブルのサイズで割った余りをハッシュ値とする関数 (簡易ハッシュ関数) を使用して測定を行なった．第 2 ハッシュも同様の関数を用いる．

測定は，ワークステーション Ultima1(Axil)，メモリ 256MB，Solaris 2.5.1 を用いて行なった．また，性能条件が厳しい部品環境での利用を考慮し，データはメモリ上に配置し，二次記憶装置への入出力は行なわない状況で測定した．

5.1 データの挿入・検索にかかる処理時間

ここでは，データ件数 1000 ～ 14000 までを 1000 ずつ増やしながら，通常実行と高速実行のそれぞれに対して，データの挿入・検索にかかる処理時間を計測した．それぞれ 300 回計測し，平均値を求める．通常ハッシュ関数を用いた場合の計測結果を，データ 1 件当たりの処理時間で図 4 に示す．実線が通常実行の結果で，破線が高速実行の結果である．図 4 より，高速実行は通常実行に比べ，4 割近く処理時間が短縮されている．また，通常実行，高速実行のどちらも，デー

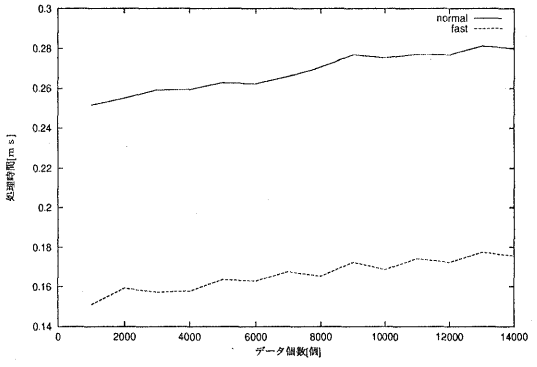


図 4 データ1件当たりの挿入・検索時間 (通常ハッシュ関数)
Fig. 4 Effect of the number of data on the insertion and retrieval time by using a normal hash function.

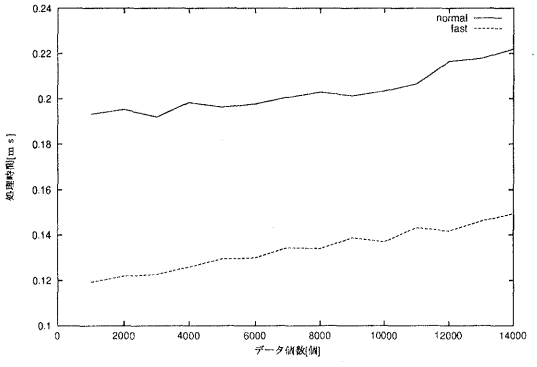


図 5 データ1件当たりの挿入・検索時間 (簡易ハッシュ関数)
Fig. 5 Effect of the number of data on the insertion and retrieval time by using a simple hash function.

タ数が増えると1件当たりの処理時間が長くなる傾向が見られる．これは，検索時に全データのフルサーチがあるからであろうと考えられる．性能劣化が大きくないのは，全データがメモリ上に配置され，二次記憶装置との I/O がないためであろうと考えられる．

簡易ハッシュ関数を用いた場合の測定結果を，データ1件当たりの処理時間で図 **5** に示す．図5では，両方とも処理時間が短縮され，通常実行と高速実行の差は3割ほどになっているが，図5にも図4と同様の傾向が見られる．

5.2 総文字数の違いによる部品の選択時間

ここでは，部品情報の部品の属するモジュール名，部品名，部品の識別名の総文字数の違いによる選択時間の差の比較を行なった．部品の属するモジュール名，部品名，部品の識別名のそれぞれの文字数を x, y, z とし，(x, y, z) と表現すると，(1, 0, 0)，(1, 1, 0)，(1, 1, 1)，(2, 1, 1)，(2, 2, 1)，…というように増加させた．それぞれの文字数毎に，100 万回部

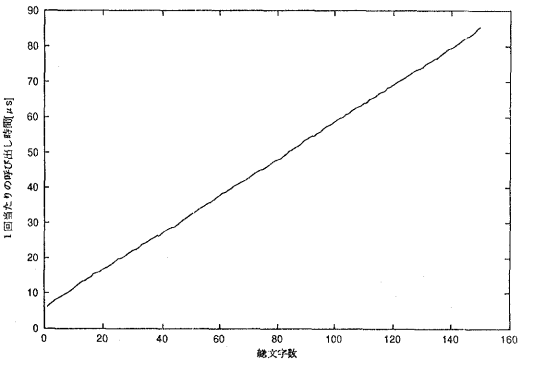


図 6 総文字数の違いによる選択時間の差の比較 (通常ハッシュ関数)
Fig. 6 Effect of the total number of characters of names on the time in invoking a software part by using a normal hash function.

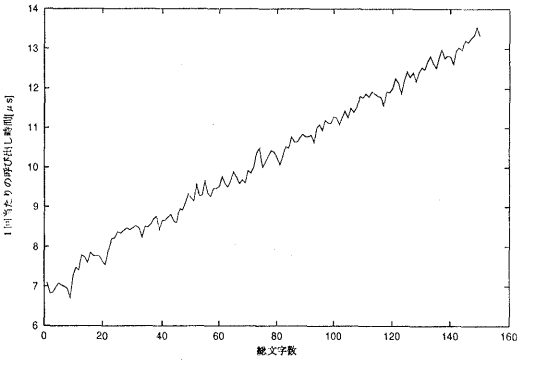


図 7 総文字数の違いによる選択時間の差の比較 (簡易ハッシュ関数)
Fig. 7 Effect of the total number of characters of names on the time in invoking a software part by using a simple hash function.

品を呼び出すまでの時間を 150 回計測し，その平均によって値を求めている．なお，この値の計測は，通常実行で行ない，部品選択時の衝突は起こらないようにしている．

通常ハッシュ関数を使用した場合の測定結果を，部品の1回当たりの呼び出し時間で図 **6** に示す．図6より，文字数に比例して検索時間が増加しているのが分かる．また，簡易ハッシュ関数を用いた場合の測定結果を図 **7** に示す．この場合，1文字と１５０文字で差が7 μ 秒ほどしかなく，部品の属するモジュール名，部品名，部品の識別名の総文字数は，部品の使用にほとんど影響を与えないということが分かる．

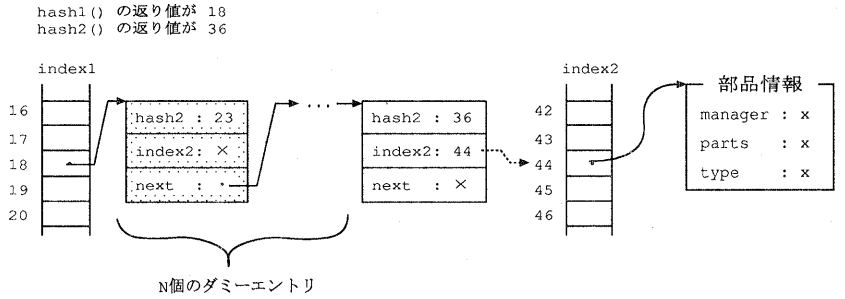


図 8 第2ハッシュの衝突なし
Fig. 8 An illustrative example of no collision of the second hash values.

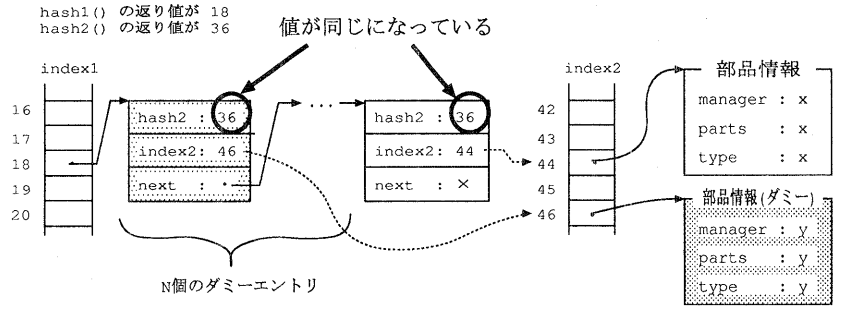


図 9 第2ハッシュの衝突あり
Fig. 9 An illustrative example of collisions of the second hash values.

5.3 部品選択時の衝突回数の違いによる部品選択時間

部品選択時の衝突回数の違いによる選択時間の計測は，第2ハッシュが衝突しない場合と，衝突する場合の2種類に対して行なった．この計測は通常実行のみである．

第2ハッシュが衝突しない状況を作り上げるために，複数のダミーエントリを用意して測定した．この様子を図 **8** に示す．図8では，ダミーエントリの第2ハッシュの値と使用したい部品の第2ハッシュの値は異なるので，使用したい部品の第2ハッシュ値を持つエントリが見つかるまで，リストをたどることになる．

第2ハッシュが衝突する状況は，ダミーエントリの第2ハッシュの値を同一にすることで作り出した．ダミーエントリの第2ハッシュの値が使用したい部品の第2ハッシュの値と同じになっているので，リストを1つたどるたびに部品情報にアクセスし，照合を行なうことになる．

通常ハッシュ関数を用いた測定結果を1回の呼び出し当たりにかかる時間で図 **10** に示す．ここで使用している部品情報の総文字数は3である．実線は，第2ハッシュの値が衝突しない場合で，破線は衝突する場合の結果である．また，簡易ハッシュ関数を用いた場合の測定結果を図 **11** に示す．通常ハッシュ関数を用いた場合と比較して処理時間が短縮されているが，同様の傾向を示している．

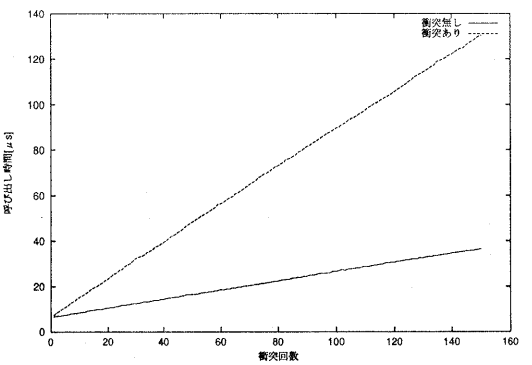


図 10 部品選択時の衝突回数の違いによる部品の選択時間 (通常ハッシュ関数)
Fig. 10 Effect of the number of collisions on the time in invoking a software part by using a normal hash function.

5.4 考 察

5.1 で見たように，高速実行は通常実行に比べ，処理時間が約3～4割短縮されている．ダイナミックリンク機能を使用した場合，部品のコードは部品が最初

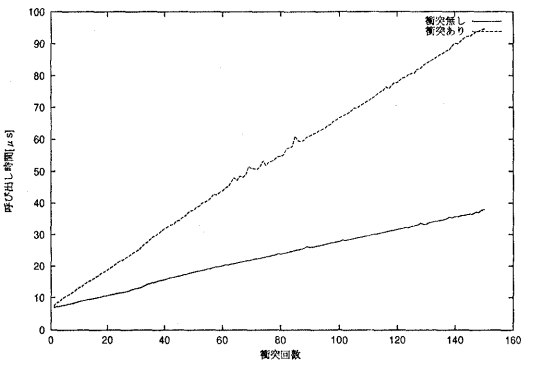


図 11 部品選択時の衝突回数の違いによる部品の選択時間（簡易ハッシュ関数）

Fig. 11 Effect of the number of collisions on the time in invoking a software part by using a simple hash function.

に実行されたときにメモリ上に配置（ロード）される．いったんメモリ上にロードされてしまえば，静的にリンクされたコードと同等のオーバヘッドで部品を実行することができる．ここで，例えば，ある部品を繰り返し1000回使用することを考えてみる．1回目の部品の呼び出しの際に部品のコードのメモリ上へのロードが起こる．しかし，残りの999回の呼び出しの際には部品のコードのロードは起こらない．従って，部品のコードのロードの平均オーバヘッドは，ロードに要する時間／呼び出し回数となる．従って，同一部品の呼び出し回数が多く，かつ，コードがメモリからスワップアウトされない場合は，部品のコードのロードの平均オーバヘッドは非常に少ないものとなる．一方，部品の検査は部品の呼び出しごとに行うのでこのオーバヘッドは無視できなくなる．**5.1** のデータの挿入・検索にかかる処理時間の測定では，1データ当たり5種類の部品を挿入と検索のそれぞれで呼び出している．高速実行と通常実行の処理速度の差は，合計10回の部品情報の検査の有無に起因するものと考えられる．従って，高速性が要求される処理では，部品の検査を行わずに部品の実行を可能とする方法は有効であると考えられる．

次に，部品情報の総文字数について考察する．図6と図7から分かるように，文字数に比例して部品の選択に要する時間が増加する．部品情報の検査のみについて考えるならば，部品の呼び出しが1回のみのときは部品情報の総文字数が多くても全体に対する影響は大きくないと考えられる．しかし，部品の呼び出しが多数回の場合，1回の部品の呼び出しにかかる時間は短くても全呼び出しにかかる時間が長くなる恐れがあるので注意する必要がある．また，総文字数があまりに多くなると処理時間に大きく影響する可能性が高いので，なるべく総文字数を少なくすべきであろう．

最後に，部品選択時の衝突回数に関して考察する．図10と図11から分かるように，部品選択時の衝突回数に比例して部品呼び出し性能が劣化し，「衝突無し」と「衝突あり」の呼び出し時間の差も大きくなっている．ここで，「衝突無し」は，衝突が生じた場合に同じハッシュ値を持つキーを連結リストを用いて連結しておく直接チェイニング法を採用し，キー（ここでは部品情報）のみを格納☆するのではなく，第2ハッシュの値をも格納して部品管理を行う場合と考えられる．一方，「衝突あり」は，第2ハッシュの値を同じにしているので，直接チェイニング法を使用してキー（部品情報）のみを格納して部品管理を行った場合と考えられる．「衝突無し」と「衝突あり」の部品呼び出し性能に差が現れているのは，部品情報の比較の方が第2ハッシュの値の比較と比べて処理が重いからであろう．このように，ハッシュのキーの比較処理が重くなる場合には，第2ハッシュの値をハッシュエントリに格納し，この値の比較でキーの比較を代用することでオーバヘッドを軽減することができると考えられる．

また，直接チェイニング法ではなく，ハッシュ値の衝突時に別の場所を捜すオープン・アドレス指定法を用いる場合には，別の場所にあった部品が所望の部品か否かを検査する必要がある．所望の部品ではない時はさらに次の場所を捜して部品情報の検査を行う必要がある．従って，衝突が多い場合には，直接チェイニング法を直接用いた（「衝突あり」の場合）と同様の部品呼び出しオーバヘッドとなると考えられる．ここでも，部品情報を比較するのではなく，第2ハッシュの値を格納しておいてハッシュ値を比較することにより，ここで示した効果と同等の効果が得られるであろうと考えられる．

さらに，1種類のハッシュ関数では，部品情報をもとに汎用的にハッシュ値を分散させるのはかなり困難である．また，ハッシュテーブルのエントリ数が2Lの1種類のハッシュ関数を用いる場合と，ハッシュテーブルのエントリ数がLの2種類のハッシュ関数を用いる場合では，ハッシュテーブルの総エントリ数が同じにもかかわらず後者の方が衝突を少なくできる可能性が高いと考えられる．

以上より，図10と図11は，ハッシュ関数を複数用

☆ 実際は，部品情報を直接ハッシュエントリに格納してはおらず，部品情報へのポインタを持った形になっているが，実質的には格納したのと同じである．

いて衝突を極力避け，ハッシュ値を格納しておいて部品情報の比較を回避することの有効性を示していると考えられる．

以上の結果により，実行性能要求の厳しい部品の実行オーバヘッドを少なくするという要求を満足できたと考えられる．また，部品を使用する際の部品選択にかかる処理時間は，ハッシュ関数の性能に大きく影響する．ハッシュ関数の内容によっては，通常実行でもより高速な処理が行なえる可能性がある．

6. 関連する研究

拡張可能DBMSは，DBMSカーネルと呼ぶ，データの格納，バッファリング，排他制御やリカバリを行う固定的なサブシステムを使用するか否かによって分類できる．DBMSカーネルを使用するシステムにはEXODUS[3]，GENESIS[4]，DASDBS[9] 等がある．DBMSカーネルを使用することにより，なるべく性能を劣化させず，また，DBの安全性を確保することができる．しかし，DBMSカーネルの提供する機能には拡張性を持たせることができない．

DBMSカーネルを使用しないシステムには，Open OODB[5]，MODUS[10]，AXIS[11]，Earth[12]，PRIMA[13] がある．Open OODB や MODUS では，機能部品を追加することができるが，機能部品に相当する関数をコンパイルする必要がある．Earth や PRIMA では，アーキテクチャ等を選択可能とすることで様々な分野に適用可能としている．しかし，あらかじめ実現された機能以外の機能を持たせることはできない．AXIS は部品間の呼び出しに RPC(Remote Procedure Call) を用いており，ネットワークにわたる部品の実行が可能である．しかし，RPC による部品呼び出しのオーバヘッドは非常に大きく，特に，データの数に応じて部品が呼び出されるような場合には使用に耐えない．

これに対して，本論文で述べた方法を採用すると，DBMSカーネルに相当する機能に対しても拡張性を持たせることができる．また，性能的に耐えられる程度のオーバヘッドで部品呼び出しが可能である．これにより，高度な拡張性を有するDBMSの実現を可能とすることができると考えられる．

7. まとめ

本論文では，拡張可能DBMSにおける機能部品の管理と，呼び出し方法について検討した．拡張可能DBMSには，部品の追加・削除において，DBMS本体を再コンパイルし，稼働しているシステムを停止させ，新しいシステムを再起動させる必要があるといった問題や，DBMSを拡張可能とすると，そのためのオーバヘッドが生じてしまうといった問題がある．

本論文では，ダイナミックリンクライブラリを利用した部品の使用によって動的な機能拡張を可能とし，また，DBMS (サーバ) そのものの柔軟性も考慮して，動的なDBMS (サーバ) の更新も可能にした．また，必要な機能を付加可能とした際に生じる実行性能オーバヘッドを削除するために，(1) 複数のハッシュを用いた部品の管理，(2) 通常実行と高速実行の2種類の実行方法を提案した．また，性能評価によって，(1) 高速実行は通常実行に比べ，約3～4割処理時間が短縮されること，(2) 部品の属するモジュール名，部品名，部品の識別名の総文字数があまりに多くなると処理時間に大きく影響する可能性が高いので，なるべく総文字数を少なくすべきであること，(3) キー（部品情報）の比較処理が重い場合に，ハッシュ関数を複数用い，第nのハッシュエントリ中にキーに加えて第n+1のハッシュ値を格納し衝突時のキーの比較をこのハッシュ値の比較で代用するのは有効であることを明らかにした．

近年のネットワーク環境の進歩に伴い，ネットワーク上の機能部品を利用したいという要求が強くなっており，RPC を用いた部品の実行[11] やエージェントを用いた部品の実行[19] が提案されている．このようなネットワークを考慮した部品の実行は今後の課題である．また，性能の良いハッシュ関数の検討，実際のDBMSへの適用も今後の課題である．

謝辞 本研究は，一部，文部省科学研究費 (課題番号 09780258) による．

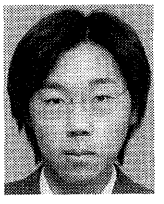
参考文献

1) Batory, D. and Mannino, M.: Panel on Extensible Database Systems, *Proc. of ACM SIGMOD*, pp. 187–190 (1986).
2) Carey, M. and Haas, L.: Extensible Database Management Systems, *SIGMOD RECORD*, Vol. 19, No. 4, pp. 54–60 (1990).
3) Carey, M. J., DeWitt, D. J., Frank, D., Graefe, G., Muralikrishna, M., Richardson, J. E. and Shekita, E. J.: The Architecture of the EXODUS Extensible DBMS, *Proc. of Int'l Workshop on Object-Oriented Database Systems*, pp. 52–65 (1986).
4) Batory, D. S., Barnett, J. R., Garza, J. F., Smith, K. P., Tsukuda, K., Twichell, B. C. and Wise, T. E.: GENESIS: An Extensible Database Management System, *IEEE Trans.*

on Software Eng., 14, 11, pp.1711–1730 (1990).
5) Wells, D. L., Blakeley, J. A. and W., T.C.: Architecture of an Open Object-Oriented Database Management System, *COMPUTER*, Vol. 25, No. 10, pp. 74–82 (1992).
6) Dittrich, K.R.: Database Technology Research at the University of Zurich: Using and Engineering Object-oriented, Active, and Heterogeneous DBMS, 信学技報 DE91–60, Vol. 91, No. 538, pp. 59–74 (1992).
7) Yaseen, R., Su, S. Y. W. and Lam, H.: An Extensible Kernel Object Management System, *Proc. of OOPSLA'91*, pp. 247–263 (1991).
8) Cooper, R. L., Atkinson, M. P., Dearle, A. and Abderrahmane, D.: Constructing Database Systems in a Persistent Environment, *Proc. of 13th VLDB Conf.*, pp. 117–125 (1987).
9) Scheck, H.-J., Paul, H.-B., Scholl, M. H. and Weikum, G.: The DASDBS Project: Objectives, Experiences, and Future Prospects, *IEEE Trans. on Know. and Data Eng.*, Vol. 2, No. 1, pp. 25–43 (1990).
10) 古瀬一隆, 石川佳治, 山口和紀, 北川博之, 大保信夫: 拡張可能ＤＢＭＳ　ＭＯＤＵＳの設計と実現技術, アドバンストデータベースシステムシンポジウム予稿集, pp. 139–148 (1992).
11) Kojima, I., Tanuma, H., Sato, Y., Ebihara, I. and Okada, Y.: Design and Implementation of an Object-Oriented Database System Using an Extensible Database Toolkit, *Proc. of 2nd Int'l Computer Science Conf.*, pp. 588–594 (1992).
12) 早田宏, 渡辺美樹, 田中圭, 山崎伸宏: オブジェクト指向データベース・エンジン Earth：キャッシュ共有のための設計空間, 情報処理学会論文誌, Vol. 39, No. 7, pp. 2240–2249 (1998).
13) Härder, T., Meyer-Wegener, K., Mitschang, B. and Sikeler, A.: PRIMA - a DBMS Prototype Supporting Engineering Applications, *Proc. of 13th VLDB*, pp. 433–442 (1987).
14) Stonebraker, M., Rowe, L. A. and M., H.: The Implementation of POSTGRES, *IEEE Trans. on Know. and Data Eng.*, Vol. 2, No. 1, pp. 125–142 (1990).
15) Haas, L. M., Chang, W., Lohman, G. M., McPherson, J., Wilms, P., Lapis, G., Lindsay, B., Pirahesh, H., Carey, M. J. and Shekita, E.: Starburst mid-flight: As the dust clears, *IEEE Trans. on Know. and Data Eng.*, Vol. 2, No. 1, pp. 143–160 (1990).
16) Buneman, P., Davidson, S. B., Hart, K., Overton, C. and L., W.: A Data Transformation System for Biological Data Sources, *Proc. of 21st VLDB*, pp. 158–169 (1995).
17) Linnemann, V., Kuspert, K., Dadam, P., Pistor, P., Erbe, R., Kemper, A., Sudkamp, N., Walch, G. and Wallrath, M.: Design and Implementation of an Extensible Database System Supporting User Defined Data Types and Functions, *Proc. of 14th VLDB*, pp. 294–305 (1988).
18) Hochin, T. and Inoue, U.: An Extensible DBMS Composed of Specific DBMSs, *Proceeding of Int'l Symposium on Next Generation Database Systems and Their Appl.*, pp.180–187 (1993).
19) 河信司, 市間健太郎, 高橋賢一郎, 家冨誠敏, 有澤博: スクラップ アンド ビルド型マルチメディアＤＢＭＳの構成方式, 情報処理学ＤＢＳ研究会, 116-62, pp. 273–280 (1998).

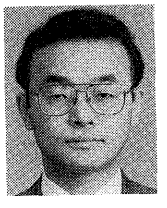
(平成11年3月20日受付)
(平成11年6月27日採録)

(担当編集委員 石川 博)



増永 浩二

平成11年福井大学情報工学科卒業. 同年, (株)ネスティ入社, 現在に至る. 在学中, データベース管理システムの拡張可能化の研究に従事.



宝珍 輝尚 (正会員)

昭和57年名古屋工業大学電気工学科卒業. 昭和59年同大大学院修士課程修了. 同年, 日本電信電話公社入社. 平成5年7月より福井大学工学部情報工学科助手. 現在, 情報・メディア工学科助教授. 博士(工学). マルチメディアデータ管理, 柔構造データベース, 拡張可能DBMS, グラフに基づくデータモデルの研究に従事. 電子情報通信学会, IEEE, ACM, 日本情報考古学会各会員.

都司 達夫 (正会員)

昭和48年大阪大学基礎工学部電気工学科卒業. 昭和53年同大大学院博士課程修了. 同年, 福井大学工学部情報工学科講師. 現在, 情報・メディア工学科教授. 工学博士. データベースシステム, プログラミング言語の研究に従事. 著書"Optimizing Schemes for Structured Programming Language Processors" (Ellis Horwood). 電子情報通信学会, IEEE, 日本情報考古学会各会員.