解　説

2. オブジェクト指向データベースシステムの実現技術

2.2 オブジェクト指向データベースシステムの記憶構造†

加　藤　和　彦††

1. はじめに

オブジェクト指向データベースシステム（以下OODBSと略す）以前のデータベースシステムが，比較的に単純で規則的なデータの集まりを操作の対象としていたのに対し，OODBSでは，実世界の情報構造を直接的に反映させた，より複雑なデータ構造をもつ永続オブジェクトを操作する機能を提供する．このためOODBSを実現させるためには，従来型のデータベースシステムにはなかった二次記憶構造を扱う必要がある．

本稿では，OODBSにおいて，オブジェクトに対する物理的なアクセス効率に最も大きな影響を与える二次記憶の管理技術に関する解説を行う．二次記憶管理に関して，関係データベースシステムに代表される従来のデータベースシステムとOODBSの差異に着目して解説する．

以下，2.では，OODBSの記憶構造を考える上で最も基本となるクラスタリング技術について述べる．3.では，データベースの検索処理を高速化する方法として従来のデータベースシステムにおいて広く利用されてきたインデクシング技術をOODBSで用いる方法について述べる．4.では，OODBSにおいて二次記憶アクセス回数を減じるのに有効な技術である，複製を用いた二次記憶技術について述べる．

2. クラスタリング技術

データベースに格納すべきデータに対するアクセス処理が高速に行えるように，データの二次記憶上での配置を決めることを**クラスタリング**(clustering)と呼ぶ．

OODBSにおけるクラスタリングには，二つのレベルが考えられる．第1レベルのクラスタリングは，内部に構造をもつデータ型の値や，ほかのオブジェクトへの参照を内部に含む**複合オブジェクト**(complex/composite object)を，二次記憶上での格納に適したレコードの集合（すなわちファイル）に分解する際のクラスタリングである．第2レベルのクラスタリングは，ファイル中のレコードを，ファイル内で位置づける際のクラスタリングである．以下では，前者を**論理クラスタリング**，後者を**物理クラスタリング**と呼ぶことにする．

従来のデータベースシステムでは，データモデル自体が二次記憶上での格納に適したものであったため，論理クラスタリングはほとんど問題とはされず，クラスタリングに関する議論は物理クラスタリングに集中していた．これに対して，OODBSでは，二次記憶の物理構造に縛られずに，実世界における情報構造を直接的に表現するモデリング能力をもつために，論理クラスタリングが重要な役割を果たす．本章ではOODBSにおける論理クラスタリングに焦点を当てて解説する．

オブジェクトの論理クラスタリングを考える上で問題となるのは，内部に構造をもつようなオブジェクトをいかにして，いくつかのレコードの集合に分割するかということである．オブジェクトの論理クラスタリングの基本的な方法は次の三つである[14]．

- **直接クラスタリング** (direct clustering)
- **正規化クラスタリング** (normalized clustering)
- **分解クラスタリング** (decomposed clustering)

実際のOODBSで使用されている論理クラスタリング法もこれらの組合せとして理解することができる．以下，本章では，これら三つの基本的な論理クラスタリング法を述べた後，実際のOODBSで用いられている論理クラスタリング法を解説する．

† Storage Management Techniques of Object-Oriented Database Systems by Kazuhiko KATO (University of Tokyo).

†† 東京大学理学部情報科学科

2.1 直接クラスタリング

オブジェクトを二次記憶媒体上のデータにマッピングする上で最も単純で直接的な方法が，直接クラスタリングである[14]．この方法では，内部に構造をもつ一つのオブジェクトを，一つの連続的な二次記憶上データに展開して表現する．このときの展開法としては，たとえば，深さ優先探索によるトラバース法を用いる．三つの基本的論理クラスタリング法の中では最もクラスタリングの単位が大きい．

例 1 次のような型をもつオブジェクトを考えよう．

```
講師型 ::=[
  講師名: string,
  電話番号: string,
  講義: setof 講義型
]
講義型 ::= [
  講義名: string,
  講義室: int,
  単位数: int
]
```

講師型のオブジェクトは，講師名，電話番号，そして構造をもつ型である講義型の集合を属性としてもつ．講義型のオブジェクトは，講義名，講義室，そして単位数という属性をもつ．

直接クラスタリングによって，講師型のオブジェクトに対して論理クラスタリングを行うと，次のようになる．ここで {…} は，…部の任意個の繰り返しを表すこととする．

```
講師ファイル ::= {
  [ 講師名: string,
    電話番号: string,
    { [ 講義. 講義名: string,
        講義. 講義室: int,
        講義. 単位数: int
      ] }
  ]
}
```

（例終り）

この方法は一つの複合オブジェクト全体に対するアクセスが頻繁に起こる場合に最も適している．短所は，オブジェクトの一部のみに対してアクセスしたい場合でも，一つのオブジェクト全体にアクセスしなければならない点である．また，複数のオブジェクトが一つのオブジェクトを共有する場合は，その共有されるオブジェクトの複製を作り，それを共有する側の複数のオブジェクト内に保持しなければならない．この場合，複製を保持するために，より多くの二次記憶容量が必要である．また，共有されているオブジェクトの属性値が更新されたときに，全ての複製にその更新を伝播させなければならないというオーバヘッドが生じる．

2.2 正規化クラスタリング

直接クラスタリングのように，概念的なオブジェクトを単純にそのまま二次記憶上にマッピングするのではなく，オブジェクトに適当な正規化を行って規則的なデータ構造を単位として二次記憶上に格納する方法が正規化クラスタリングである[14]．このときの正規化は，関係データベースにおける第一正規形と同じように，内部に構造をもたない平坦 (flat) なタプルを単位として，二次記憶上に格納する．三つの基本的論理クラスタリング法の中で，正規化クラスタリングの単位は中間的な大きさである．

例 2 例1の講師型，講義型に加えて，次のような学生型が定義されていることとする．

```
学生型 :: = [
  学生名: string,
  電話番号: string,
  講義: setof 講義型
]
```

講師は一個以上の講義を講義し，学生は一個以上の講義を受講する．講義はそれ自身が identity をもつと考えられ，講義型オブジェクトが存在し，講義型オブジェクトは講師型のオブジェクトと学生型のオブジェクトの両方から参照され，共有されるものとする．このとき，講師型オブジェクトと学生型オブジェクトから始まる有向グラフ構造が形成される．講師型，学生型，講義型のオブジェクトに対して正規化クラスタリングを行うと次のようになる．

```
講師型ファイル ::= {
  [ 講師 ID,
    講師名: string,
    電話番号: string,
    講義: 講義集合 ID ] }
```

```
学生型ファイル ::= {
  [ 学生 ID,
    学生名: string,
    電話番号: string,
    講義: 受講集合 ID ]
講義型ファイル ::= {
  [ 講義 ID,
    講義名: string,
    講義室: string,
    単位数: int ] }
講義集合 Join Index ファイル ::= {
  [ 講義集合 ID,
    講義 ID ] }
受講集合 Join Index ファイル ::= {
  [ 受講集合 ID,
    講義 ID ] }
```

講師型，学生型，講義型のそれぞれについてオブジェクトが格納されるファイルとして，それぞれのオブジェクトに対応するファイルを用意する．オブジェクトの属性値として集合を含む場合は，集合を Join Index[13] と呼ばれる二項レコードのファイルに格納することによって，レコードの平坦化を行うことができる．講義集合 Join Index ファイルには，講師が講義を行う講義の集合を表す二項レコードの集合，また，受講集合 Join Index には学生が受講する講義の集合を表す二項レコードの集合を格納する．

一つの講師型オブジェクトから，講義を行う講義の集合を得る場合は，その講師型オブジェクトの講義属性値として格納された講義集合 ID を検索キーとして講義集合 Join Index ファイルを検索することにより，目的とする講義型オブジェクトを指し示す講義 ID の集合が得られる．学生型オブジェクトから，受講する講義の集合を得る場合も，同様に，受講集合 ID をキーとして受講集合 Join Index を検索することにより得られる．

このように，正規化クラスタリングでは，一つの複合オブジェクト全体にアクセスするには，レコード識別子からそのレコード識別子が示すレコードへの**航行** (navigation) 操作が必要となる．

(例終り)

この方法の最大の特徴は，内部に構造をもつオブジェクトを平坦なレコードの集まりとして管理できる点にある．このために，リレーショナルデータベースのために開発されたさまざまな二次記憶管理技術をオブジェクトの管理に適用することが可能となる．

この方法の長所，短所は直接クラスタリングと対照的である．オブジェクト全体よりも，オブジェクトの部分に対するアクセスに適している．また，オブジェクトの共有を，複製ではなく，オブジェクト識別子を参照することによって実現できる．このため，共有されているオブジェクトが更新されたときは，そのオブジェクトのみを更新すればよい．

正規化クラスタリングを採用した場合，一つのオブジェクト全体に対するアクセスの効率は，直接クラスタリングを採用した場合に比べて劣る．例 2 で述べたように，一つのオブジェクト全体にアクセスするには，オブジェクトを構成する要素オブジェクトの数およびオブジェクトのネスト構造の深さに応じた航行操作が必要となり，多くの二次記憶アクセス回数が必要である．複製を用いた技法により，この航行操作の回数を減少させる方法を **4.3** で述べる．

2.3 分解クラスタリング

分解クラスタリングは，正規化クラスタリングにおける正規化操作を最大限に押し進め，オブジェクトの属性値とオブジェクト識別子の二項をもつレコードによりオブジェクトを管理する方法である[3]．これまでに述べた基本的論理クラスタリング法の中では，最も論理クラスタリングの単位が小さい．

例 3　例 1 で定義した講師型のオブジェクトに対して分解クラスタリングを行うと，次のように，講師名，電話番号，そして講義の三つの属性に対応して三つのファイルが生成される．

```
講師型ファイル 1 ::= {
  [ 講師 ID,
    講師名: string ] }
講師型ファイル 2 ::= {
  [ 講師 ID,
    電話番号: string ] }
講師型ファイル 3 ::= {
  [ 講師 ID,
    講義: string ] }
```

分解クラスタリングでは，属性がすべて分解され，属性値と，属性値の属するオブジェクトの識

別子のペアが各ファイルに格納される．

（例終り）

分解クラスタリングは，属性の分解が最大限になされているために，オブジェクト中の少数の属性のみがアクセスされる場合に適している．

分解が最大限になされていることを物理クラスタリングの際に生かすことができる．属性値に対する内容検索操作が多いのか，航行操作が多いのかに応じて，属性値かレコード識別子のいずれに関して物理クラスタリングすればよいのかを決定する．もし，両方の操作が多い場合は，その属性値を格納したファイルの複製を作り，一方を属性値で，他方をレコード識別子で物理クラスタリングすることにより，両方の操作を高速化できる．

分解クラスタリングの短所は，オブジェクト全体もしくは1オブジェクト中の比較的多くの属性がアクセスされる場合に，多くのファイルをアクセスしなければならないために性能が低下してしまうという点である．

2.4 実際のシステムにおける論理クラスタリング

2.1～2.3 で述べた基本的論理クラスタリング法は，オブジェクトを分解するときの単位が直接クラスタリング，正規化クラスタリング，分解クラスタリングの順で大きく，この単位の大きさに起因する長所と短所を併せもっていた．これら三つの基本的な論理クラスタリング法を組み合わせて用いることにより，OODBS の利用状況に合わせた，それぞれの短所を補完した論理クラスタリングを行うことができる．たとえば，1オブジェクト型の中にある属性値群は，同時にアクセスされることが多いということがあらかじめ分かっている場合は，それらの属性群が一つのファイルに格納されるような直接クラスタリングを行い，それ以外の属性については正規化クラスタリングを用いるとよい．これは，正規化クラスタリングに直接クラスタリングの概念を取り入れたものになっていると考えられる．

実際のシステムでは，ORION[8),10)]のようにシステムが一つの論理クラスタリング法のみ（ORION は基本的に正規化クラスタリングのみ）を提供するというシステムもあるが，データベース設計者が複数の論理クラスタリング法の中から選択できるようになっているものがある．その例として ENCORE と O_2 でとられている方法を紹介する．

ENCORE システム[5)]では，データベース設計者がオブジェクトごとに次の4つの論理クラスタリング法を指定できるようになっている．4つの論理クラスタリング法とは，（a）分解クラスタリング（**2.3** 参照），（b）直接クラスタリング（**2.2** 参照），（c）型によるクラスタリング，（d）属性値によるクラスタリングである．（c）の型によるクラスタリングとは，同じ型のオブジェクトを同じクラスタに格納するという方法である．（d）の属性値によるクラスタリングとは，同じ属性値をもつオブジェクトを同じクラスタに格納するという方法であって，たとえば，「青い」という属性をもつ自動車オブジェクトは，オブジェクトの型（すなわち車種）が異なっても同じクラスタに入れるという方法である．ENCORE はオブジェクトの共有を複製により行う．すなわち，一つの子オブジェクトが複数の親オブジェクトの構成要素となる場合は，それぞれの親オブジェクトに複製を保持することによりオブジェクトの共有を実現する．ENCORE では，共有を複製によって実現しているために，親オブジェクトごとに適当なクラスタリングが行えるという長所があるが，反面，子オブジェクトの属性値が変更されたときは複製に対する更新も行わねばならないという短所がある．

O_2[4)]は，基本的には直接クラスタリングによる論理クラスタリングを行うが，直接クラスタリングを行う単位をデータベース管理者が自由に指定できるようになっている．O_2 では，オブジェクトの共有を複製によらず，オブジェクト識別子による参照関係により行う．このため，オブジェクトの実体は二次記憶上で物理的に一つの場所に格納されるだけである．このときに，オブジェクトをどのオブジェクトと同じファイルに格納するかという指示を，クラスごと，**配置木** (placement tree)[1)] と呼ばれる概念スキーマを表す木の部分木によって，データベース管理者が指示する．明示的な指示が与えられない場合は，システムが自動的に決定する．配置木を後から更新することも可能であり，その変更に応じて論理クラスタリングも動的に変更される．

3. インデクシング技術

インデクシング (indexing) は，内容検索処理を高速化するために，データベースに格納されているデータに補助的なアクセス情報を付加する技術である．本章では，OODBS におけるインデックスの基本的管理法と，内部に構造をもつ複合オブジェクトに対するインデクシング技術について述べる．

3.1 インデックスの管理法

関係データベースシステムにおけるインデクシングがリレーションの属性ごとに行うのが基本となっているのに対し，OODBS におけるインデクシングはクラスの属性に対して行うのが基本となる．このときにクラス階層を利用せずにインデックスを管理する方法を**単クラスインデクシング** (single-class indexing)，クラス間の階層関係を利用する方法を**クラス階層インデクシング** (class-hierarchy indexing) と呼ぶ[9]．

単クラスインデクシングでは，クラス階層に関する情報を使わずにクラスごとにインデックスを管理する．たとえば，上位のクラス A において属性 a が定義されていて，A のサブクラス B が属性 a を継承している場合を考える．クラス A, B の両方について，属性 a にインデックスを付加する必要がある場合，単クラスインデクシングでは，クラス A, B のそれぞれに別々に属性 a のインデックスを管理する．

いま，属性 a を継承するクラスが n 個あり，その n 個のクラスにおいて属性 a にインデックスを付加するとしよう．インデックスの大きさが p ページとすると，属性 a に付加されたインデックスの総容量は np となる．インデックスの構造としてB木を採用したとすると，属性 a についてある値をもつ全オブジェクトの集合を求めるのに必要な平均アクセスページ数は $n*\log p$ となる．

クラス階層インデクシングでは，インデックスの管理にクラス階層に関する情報を利用する．ある属性に付加されるインデックスは，クラス階層全体でただ一つのみが管理される．先ほどの例でいえば，クラス A, B の両方について属性 a にインデックスを付加する必要がある場合でも，保持される属性 a のインデックスは一つのみである．属性 a に関するインデックス総量は一般に np よりかなり小さい値で済み，検索時のアクセスページ数は $\log np$ (多くの場合 $\log np < n*\log p$) でよい．

ある属性にインデックスを付加する場合に，限られた数のクラスに属するオブジェクトにのみインデックスを保持すればよい場合には，単クラスインデクシングのほうが適しており，それ以外の場合はクラス階層インデクシングが適している．

ORION システムでは，性能評価の結果，クラス階層インデクシングが採用されている[10]．O_2 システムにおいても，クラス階層インデクシングに若干の改良を加えた方式が用いられている[4]．

GemStone では，前述のいずれの方法でもないインデックスの管理法が採用されている．GemStone ではインデックスをクラスごとに管理するのではなく，データの集まり (collection；GemStone では Bag および Set 型) ごとに管理するようになっている[11]．文献 11) によれば，この第一の理由は，インデックスを使用しないプログラムの優先度を第一に考えて，インデックスの管理にともなうオーバヘッドが，インデックスを必要としないプログラムに影響を与えないようにするため，としている．第二の理由は，データの集まりごとにインデックスを与えるようにしたほうが，検索結果にインデックスを与えるなど，より柔軟にインデックスを利用できるため，としている．

3.2 複合オブジェクトのインデクシング技術

内部に，別のオブジェクトまたは構造化されたデータをもつ複合オブジェクトに対するインデクシング技術が文献 2) において提案されている．提案は，**入れ子インデックス** (nested index)，**パスインデックス** (path index)，そして**マルチインデックス** (multiindex) の三つのインデクシング法からなっている．これら三つのインデクシング法を，次のようなスキーマをもつ自動車データベースを例として説明しよう．

例 4

```
自動車 ::= [
    色: string,
    製造会社: 会社
]
会社 ::= [
    会社名: string,
    部署集合: setof 部署
```

]
部署 ::= [
部署名: string
]

自動車型のオブジェクトは会社オブジェクトを参照しており，会社オブジェクトはさらに部署オブジェクトを参照している．データベース中のデータには次のような値をもつオブジェクトが格納されているものとする（部署1，2，…，6の値は省略した）．

自動車1 =["白", 会社2]
自動車2 =["赤", 会社2]
自動車3 =["白", 会社1]
会社1 =["トヨタ",
{部署1，部署3，部署5}]
会社2 =["ニッサン", {部署2，部署4}]
会社3 =["ホンダ", {部署6}]

この例を用いて，三つのインデクシング法は次のように説明される．

入れ子インデックスでは，オブジェクトの参照関係の先頭と最後の参照関係のみをインデックスとして保持する．たとえば，自動車データベースで，'自動車.製造会社.社名' というパスについて入れ子インデックスを保持する場合は，車名と自動車オブジェクトの集合が保持され，次の二つのレコードがインデックスとして保持される．

["トヨタ", {自動車3}]
["ニッサン", {自動車1，自動車2}]

パスインデックスでは，オブジェクトの参照関係の先頭と最後だけではなく，参照関係の途中のオブジェクトも含めて，インデックスとして保持する．自動車データベースの例では，次の三つのレコードがインデックスとして保持される．

["トヨタ", {自動車3.会社1}]
["ニッサン", {自動車1.会社2,
自動車2.会社2}]

マルチインデックスは，パスインデックスと同等の情報を，入れ子インデックスの集合により保持する．自動車データベースの例では，'自動車.製造会社' というパスに対する次のような入れ子インデックスと，

[会社1，{自動車3}]
[会社2，{自動車1，自動車2}]

'会社.会社名' というパスに対する次のような入れ子インデックスを保持する．

["トヨタ", {会社1}]
["ニッサン", {会社2}]
["ホンダ", {会社3}]

（例終り）

これら三つのインデクシング法の詳しい解析と評価が文献2）においてなされている．その結果によれば，検索性能は入れ子インデックスが最も良く，パスインデックス，マルチインデックスがこの順で続くとしている．また，更新性能はマルチインデックスが最も良く，入れ子インデックスとパスインデックスは複合オブジェクトのネストの深さや，どのネストレベルで更新が起きるかにより性能の順序が異なるとしている．

4. 複製技術

従来のデータベースシステムでは，更新時のオーバヘッドを軽減するために，一つのデータベース内のデータの複製はできるだけ保持しないような戦略がとられてきた．OODBS では，従来のデータベースシステムに比べて，複雑な概念スキーマを直接的に表現する能力を有するために，複製を管理するオーバヘッドを考慮に入れた上でもなお，複製を保持することにより，性能を向上させることが可能となる場合がある．

これまでに提案されている OODBS における複製技術の利用方法には次の三つがある．

1. 複製クラスタリング
2. 情報圧縮効果
3. 航行操作の高速化

本章では，これら三つの利用法と複製管理のオーバヘッドを更新の遅延により軽減する方法について述べる．

4.1 複製による複数クラスタリング

2. で述べたように，OODBS におけるオブジェクトのクラスタリングにはいくつもの方法があり，それぞれ長所，短所がある．そこで，オブジェクトの**複製** (replication) を作り，複製ごとに異なるクラスタリング法を用いれば，それぞれのクラスタリング法の長所を組み合わせて利用することができる．実際に，この方法は ENCORE システム[5]で提供されている（2.4 参照）．

複製による複製クラスタリングは，検索処理の高速化に有効であるが，更新処理時には，更新を

全ての複製に伝播させねばならないというオーバヘッドがともなう．このためにこの方法は，更新処理が相対的に少ない場合に検索処理を高速化するのに有効である．

4.2 複製による情報圧縮

さまざまなオブジェクトに散在する情報の複製を一カ所に集めて格納すると，実質的な情報濃度の濃い物理クラスタを作ることができる．この物理クラスタは，少ない二次記憶ページ数のなかに，ある処理を遂行する上で必要な情報が圧縮して格納されていると考えられ，少ないディスクアクセス回数で有効な情報を得ることができる．文献12) ではこの方法を**分離複製** (separate replication) 法と呼んでいる．この方法も**4.1**と同様，更新処理が検索処理に比べて少ない場合に有効である．

4.3 複製による航行操作の高速化

ほとんど全てのOODBSにおいてオブジェクトindentityの概念[7]がサポートされている．そのようなシステムでは，各オブジェクトにはシステム内でユニークなオブジェクト識別子が与えられ，その識別子を参照し合うことにより，オブジェクトの参照関係を直接的に表現することができ，さらに，オブジェクト間の共有関係をも表現することができる．オブジェクトidentityの概念をサポートするシステムでは，オブジェクトAが別のオブジェクトBを参照するということは，オブジェクトBの識別子がオブジェクトAの属性値として格納されることによって表現される．オブジェクトAから，オブジェクトA内に格納されたオブジェクトBの識別子をたどって，オブジェクトBにアクセスすることをオブジェクトAからオブジェクトBへの**航行** (navigation) 操作という．

オブジェクト識別子による参照がある属性値を格納した二次記憶ページ内に，参照するオブジェクトの全体もしくは，オブジェクトの属性値の一部を格納しておくことにより，参照している側のオブジェクトにアクセスするだけで，航行にともなうオーバヘッドなしに航行操作と同じ効果を得ることができる．この技法を文献12) では**埋め込み複製** (in-place replication)，文献6) では**永続キャッシング** (persistent caching) と呼んでいる．

このときの複製は，参照するオブジェクトの属性値のみならず，参照するオブジェクトに関する任意の情報を格納することができる．その一例を示そう．オブジェクト識別子として物理アドレスとは独立な識別子を採用しているシステムを考える．そのようなシステムでは，オブジェクト識別子から物理アドレスを得るために，二次記憶上のマッピングテーブルへのアクセスオーバヘッドが必要である．オブジェクトの物理アドレスの複製を，参照するオブジェクトが格納されたページ内に置いておくことにより，参照先のオブジェクトの物理位置が変更されるまでは，マッピングテーブルにアクセスすることなしにオブジェクト識別子から物理アドレスが得られる．

オブジェクト間の参照関係が多段 (n 段) になっていて，始点オブジェクトがO_0，終点オブジェクトがO_nとなっている場合を考えよう．**2.2**で述べた正規化クラスタリングが採用されている場合では，一般にn回の航行操作が必要である．文献12) では，多段参照の先のオブジェクトの情報を参照元のオブジェクトの属性が格納されている二次記憶ページ内に保持することにより，O_0にアクセスするだけで，O_nの属性の一部にアクセスする方法を示している．

このようにして複製により航行操作を高速化する場合も，更新処理が少ない場合に有効である．

4.4 更新の遅延

4.1～**4.3**で述べたように，複製技術はOODBSの効率化に有効であるが，更新処理が多くなるにつれてその効果が薄れ，更新処理が検索処理よりも多い場合や，一つのオブジェクトの複製の数が多過ぎる場合は，複製をしない場合に比べて性能が大きく劣化してしまう．この性能劣化の原因は，オブジェクトの属性値の更新処理時に，更新情報を全ての複製に伝播させねばならないことに起因する．

複製に対する更新の伝播を，複製に対する参照の要求があるまで遅延することにより，更新処理がある程度頻繁に起きても，処理性能の劣化を免れることが期待できる．文献6) では，更新の複製への伝播を

- 複製作成の時刻印を複製に付加すること，そして，
- 原データの最新更新時間に関する情報を主記憶上のハッシュ表に保持すること

により減じる方法が提案され，その性能が解析的に評価されている．その結果によれば，主記憶上

のハッシュ表に十分大きさを確保できる場合は，更新の複製への伝播オーバヘッドが除去可能であるとしている．

5. おわりに

OODBS の研究はシステム開発が先行する形で進んできたこともあって，本文中に述べたようなOODBS 独自の二次管理技術がすでに研究開発されてきている．しかしまだ，OODBS の使用経験の蓄積が十分でないため，これらの技術が実際のアプリケーションにとってどれほど有効かという検証はほとんどなされていない．今後は，使用経験からのフィードバックを含めた技術進展が必要であろう．

謝辞 常日頃ご指導いただく東京大学 益田隆司教授に感謝いたします．また，日頃有益な議論をしていただく ICOT 演繹・オブジェクト指向データベースワーキンググループ，および，同 DBPL サブワーキンググループの皆さまに感謝いたします．

参考文献

1) Benzaken, V. and Delobel, C.: Dynamic Clustering Strategies in the O_2 Object Oriented Database System, Technical report, Altair (Oct. 1989).
2) Bertino, E. and Kim, W.: Indexing Techniques for Queries on Nested Objects, *IEEE Trans. Knowledge and Data Eng.*, 1(2), pp. 196-214 (June 1989).
3) Copeland, G, P. and Khoshafian, S.: A Decomposition Storage Model, in *Proc. of ACM SIGMOD*, pp. 268-279 (May 1985).
4) Deux, O. et al.: The Story of O_2, *IEEE Trans. Knowledge and Data Eng.*, 2(1), pp. 91-108 (Mar. 1990).
5) Hornick, M. F. and Zdonik, S. B.: A Shared, Segmented Memory System, *ACM Trans. Office Information Syst.*, 5(1), pp. 70-95 (Jan. 1987).
6) Kato, K. and Masuda, T.: Persistent Caching: An Implementation Technique for Complex Objects with Object Identity, Technical Report 89-021, Dept. of Information Science, Univ. of Tokyo (1989).
7) Khoshafian, S. and Copeland, G.: Object Identity. In *Proc. ACM Conf. on OOPSLA*, pp. 406-416 (Oct. 1986).
8) Kim, W., Banerjee, J., Chou, H, T., Garza, J. F. and Woelk, D.: Composite Object Support in an Object-Oriented Database System. In *OOPSLA '87*, pp. 118-125 (Oct. 1987).
9) Kim, W., Dalse, K, C. and Date, A.: Indexing Techniques for Object-Oriented Databases. In Kim, W. and Lochovsky, F. H. editors, *Object-Oriented Concepts, Databases and Applications*, chapter 15, pp. 371-394, Addison-Wesley (1989).
10) Kim, W., Garza, J. F., Ballou, N. and Woelk, D.: Architecture of the ORION Nextgeneration Database System, *IEEE Trans, Knowledge and Data Eng.*, 2(1), pp. 109-124 (Mar. 1990).
11) Maier, D., Stein, J., Otis, A. and Purdy, A.: Development of an Object-Oriented DBMS. In *OOPSLA 86*, pp. 472-482 (Sep. 1986).
12) Shekita, E. J. and Carey, M. J.: Performance Enhancement Through Replication in an Object-Oriented DBMS. In *Proc. of ACM SIGMOD*, pp. 325-336 (June 1989).
13) Valduriez, P.: Join In dices, *ACM Trans. on Database Systems*, 12(2), pp. 218-246 (June 1987).
14) Valduriez, P., Khoshafian, S. and Copeland, G.: Implementation Techniques of Complex Objects. In *Proc. 12th VLDB*, pp. 101-110 (Aug. 1986).

(平成3年2月18日受付)

加藤 和彦（正会員）

昭和37年生．昭和60年筑波大学第三学群情報学類卒業．昭和63年同大学院博士課程工学研究科中退．東京大学理学系研究科博士課程入学．現在，東京大学理学部情報科学科助手．オペレーティング・システム，データベースシステム，プログラミング言語などシステム・ソフトウェア全般に興味をもつ．ソフトウェア科学会，ACM，IEEEE 各会員．