マルチメディア通信と分散処理 91－11
（1999. 1. 22）

分散環境におけるオブジェクトデータ格納方式

井口 昭人† 諏訪 健一† 福士 公三‡ 滝沢 誠† 龍 忠光‡

† 東京電機大学理工学部経営工学科

‡ シーエーアイ株式会社

E-mail {igu, suwa, taki}@takilab.k.dendai.ac.jp

QWN00117@nifty.ne.jp, fukushi@cai.co.jp

現在の情報システムは、通信ネットワーク技術の発展にともない、複数の計算機が相互接続された分散型の形態 [1] となっている。これらの分散システム内のアプリケーションは、 C++ や JAVA [2] などといったオブジェクト指向言語で記述されたオブジェクトの集合によって構成され、これらのオブジェクトが互いに協調動作を行う事によって実現されている [9, 11]。このような複数のオブジェクトで構成されるオブジェクト指向システムでは、多種多様のオブジェクトを効率よく利用するために、オブジェクトを二次記憶に格納する必要がある。本論文では、リレーショナルデータベースシステムに格納する方法について考える。また、分散環境における高速なアクセス方法として、索引とキャッシング機構について論じる。

Object Storage System in Distributed Environment

Akihito Iguchi†, Kenichi Suwa†, Kouzou Fukushi‡,
Makoto Takizawa†, and Tadamitsu Ryu‡

† Tokyo Denki University

‡ CAI Inc.

E-mail {igu, suwa, taki}@takilab.k.dendai.ac.jp

QWN00117@nifty.ne.jp, fukushi@cai.co.jp

This paper discusses how to store objects in relational database systems distributed on networks. In this paper, objects are stored in a table of each database system, where a tuple denotes an object identifier, an attribute, and a value in order to store objects of various schemes in a table. We discuss caching and indexing mechanisms to efficiently access the distributed objects.

1 はじめに

通信ネットワーク技術の発展にともない、現在の情報システムは複数の計算機が相互接続された分散型の形態となっている。これらの分散システム上のアプリケーションは、 C++ や JAVA などといったオブジェクト指向言語で記述されたオブジェクトの集合によって構成される。これらのオブジェクトは、互いに協調動作を行う事により、アプリケーションが実現されている。こうしたオブジェクトの概念に基づいたアプリケーションの枠組として、CORBA [4] 等が提案されている。このような複数のオブジェクトで構成されるシステムでは、多種多様なオブジェクトを効率よく永続的に利用することが必要となる [10]。このために、オブジェクトを二次記憶に格納する必要がある。近年、オブジェクトを格納するためのオブジェクト指向データベース (OODB) [3, 6] が利用されてきているが、標準的な商品とはなっていない。一方、*Oracle* [7]、*Sybase* [5] といったリレーショナルデータベースシステム (RDBS) [8] は、SQL の国際標準化にともない、各々の分野で広く普及している。このため、本論文では、オブジェクトを RDBS に格納する方法について考える。さらに、分散環境において、ネットワーク上に分散したオブジェクトを高速にアクセスする方法として、キャッシングと索引機構を考える。また、実装したシステムの性能評価を行う。

本論文の構成を示す。第 2章では、提案するシステムモデルを述べ、第 3章では、本論文で考えるオブジェクトとその格納方法を示す。また、第 4章では、分散されたデータベースにおける、高速なデータ検索の方法について示す。

2 システムモデル

2.1 分散オブジェクトモデル

分散オブジェクトシステムは、複数の計算機上に配置されているオブジェクトから構成される。これらのオブジェクトが互いに協調動作を行うことにより、分散アプリケーションが実現されている。オブジェクト間では、 TCP/IP により高信頼に通信が行える。

システムは、複数のサーバとクライアントから構成されている [図 1]。利用者は、まずクライアントの存在する計算機内のデータベースサーバ (リレーショナルデータベース) に対して検索を行う。これを、ローカルデータベースとする。ここで、このローカルデータベースにオブジェクトが存在しなければ、ネットワークにより相互接続されている他のデータベースに対して検索を行う。また、オブジェクトは多重化され、複数のデータベースサーバにレプリカが存在

する場合もある。

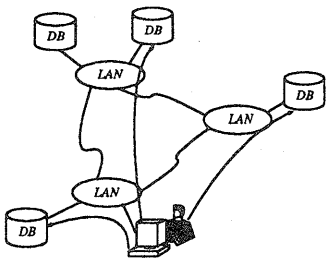

図 1: システムモデル

2.2 オブジェクト

オブジェクトには、クラスとインスタンスの二種がある。クラス C_i は属性の組 $\langle A_{i1}, \cdots, A_{im_i} \rangle$ によって表される。ここで、A_{ij} は属性である $(j = 1, \cdots, m_i)$。$dom(A_{ij})$ は、属性 A_{ij} の定義域であり、A_{ij} の取り得る値の集合である。各属性 A_{ij} に、定義域 $dom(A_{ij})$ 内の値を与えたものが、クラス C のインスタンスである。本論文では、オブジェクトという用語は、インスタンスを示すものとする。クラス C_i のオブジェクト o_i は、以下のように示される。

$o_i = \langle [A_{i1}{:}\{v_{i1}\}], \cdots, [A_{im_i}{:}\{v_{im_i}\}] \rangle$

各属性 A_{ij} は、$dom(A_{ij})$ 内の一つ以上の値を取ることが出来る。各オブジェクト o_i は、システム全体で一意なオブジェクト識別子 $oid(o_i)$ を持つ。

3 永続オブジェクトの格納方法

プロセスにより生成されたオブジェクトは、プロセスの終了に消滅する。このため、オブジェクトを再利用するために、データベースに格納しておき、永続化する必要がある。ここでは、動的に生成、消滅されるオブジェクトを、どのようにデータベース、特にリレーショナルデータベースに格納するかについて述べる。

3.1 オブジェクトデータの特徴

種々のオブジェクトがシステム内で生成、消滅される。従って、データベースは、オブジェクトの生成、消滅に柔軟に行わせる必要がある。リレーショナルデータベースシステムは、データの高速なアクセス方式、複数のアプリケーション下でのデータベースの一致性を保つ同時実行制御、障害復旧等の機能を保持している。このようなオブジェクトをリレーショナルデータベースシステム (RDBS) に格納する場合、以下に述べる 2 通りの方法が考えられる。

- 汎 (*universal*) テーブルに格納。
- クラス毎にテーブルを作成して格納。

以下にこれらの 2 通りの方法におけるオブジェクトの格納法について考える。

3.1.1 汎テーブル

まず汎テーブルを用いる方法を考える。汎テーブルとは、全オブジェクトの全属性から構成されるテーブルである。データベースは、一つの汎テーブル U を持つ。テーブルの各行は、一つのオブジェクトを示す。オブジェクトは異った属性構成を持つ。例えば、新しくオブジェクトが生成された場合、オブジェクトの属性が汎テーブルに存在しない場合があり得る。このとき、汎テーブル U に対して属性の追加が必要となる。また、オブジェクトが消滅する場合は、生成とは逆にテーブルの属性を削除する必要がある。

ここで、以下に示すオブジェクトを格納する場合を考える。

*Object*1 :
⟨[作者:*Taro*], [タイトル:*Networks*], [発行年:1998]⟩
*Object*2 :
⟨[作者:*Hanako*], [タイトル:*Databases*], [出版社:*TDU*]⟩

これらのオブジェクトを格納した汎テーブル U を図 2に示す。図で "–" は、空値を示す。

作者	タイトル	発行年	出版社
Taro	*Networks*	1998	—
Hanako	*Databases*	—	*TDU*

図 2: 汎テーブル U

汎テーブル U は、全オブジェクトの全属性から構成される。従って、オブジェクトに存在する属性が、別のオブジェクトには存在しない場合がある。図 2に示した例では、オブジェクト *Object*1 は 属性 "出版社" を持っておらず、逆に *Object*2 は、 属性 "発行年" を持っていない。このため、これらのオブジェクトをテーブルに格納する場合、その属性の値は空値 (*NULL*) となる。このように、汎テーブルには、オブジェクトの種類 (クラス) が増加するにつれて、空値が多く含まれ記憶効率が良くない。

次に、汎テーブル U において、以下に示す新しいオブジェクトが生成された場合について考える。

*Object*3 :
⟨[作者:*Jiro*], [タイトル:*Logics*], [発行年:1997]⟩

汎テーブル U では、新しいクラスのオブジェクトが生成されるたびに、テーブルのスキーマの変更が必要となる。このことは、テーブルの管理の複雑さと処理速度の低下につながる。そのため、テーブルを作成する際には、全てのオブジェクトの属性をあらかじめ知っておく必要がある。ここで、新しく生成されたオブジェクト *Object*3 を追加したテーブルを図 3に示す。

作者	タイトル	発行年	出版社
Taro	*Networks*	1998	—
Hanako	*Databases*	—	*TDU*
Jiro	*Logics*	1997	—

図 3: 汎テーブルにおけるオブジェクトの追加

次に、図 3の例において *Object*2 が消滅した場合を考える。汎テーブル U では、各行で一つのオブジェクトを示しているため、*Object*2 が消滅した場合には、テーブル U から *Object*2 に対応する一行が削除される。これにともなって、そのオブジェクトしか持っていなかった属性も削除する必要がある。そのためオブジェクトの生成と同様に、テーブル U のスキーマの変更が必要となる。ここで、*Object*2 が消滅した後のテーブルを図 4に示す。

作者	タイトル	発行年
Taro	*Networks*	1998
Jiro	*Logics*	1997

図 4: 汎テーブルにおけるオブジェクトの消去

3.1.2 クラス毎にテーブルを作成

次に、クラス毎にテーブルを作成する方法を考える。ここでは、同じスキーマのオブジェクトを集めて一つのテーブルとする。汎テーブル U とは異り、各テーブルには空値は含まれない。一方、新しいクラスのオブジェクトが生成される度にテーブルの作成が必要になる。逆に、オブジェクトが消滅する度に、該当するテーブルの削除が必要になる。本方法は、汎テーブルに比べ単純ではあるが、オブジェクトの生成、消滅にともない、テーブルの作成、削除が発生するために処理時間及び使用記憶領域が増大してしまうという欠点がある。

汎テーブル U の例で用いたオブジェクト *Object1* と *Object2* をクラスごとにテーブルを作成して格納した場合を図5に示す。

Object1	作者	タイトル	発行年
	Taro	*Networks*	1998
	Jiro	*Logics*	1997

Object2	作者	タイトル	出版社
	Hanako	*Databases*	*TDU*

図 5: クラス毎のテーブル

3.2 オブジェクト格納方式

汎テーブルとクラス毎にテーブルを作成する方法があった。汎テーブルは記憶量が増大する点が問題となり、クラス毎のテーブルでは、オブジェクトの生成、消滅に対して、処理負荷が増大する問題点がある。

オブジェクト $o = \langle A_1{:}v_1, \cdots, A_m{:}v_m \rangle$ の格納方式について考える。オブジェクト o は、識別子 $oid(=oid(o_i))$、属性 A_i とその値 v_i との組 $\langle$ *oid*, A_i, $v_i \rangle$ の集合 $\{ \langle$ *oid*, A_1, $v_1 \rangle, \cdots, \langle$ *oid*, A_m, $v_m \rangle \}$ としてで表す事が出来る。図6に示すテーブル *Object*(*oid*, *attribute*, *value*, *no*) により、オブジェクトを格納することを考える。

例として、本を示すオブジェクト *Object1* を考える。本 *Networks* の第一作者 "*Taro*" と第二作者 "*Makoto*" であるとする。*Object1* = $\langle$ *oid*:1, 作者:⟨*Taro*, *Makoto*⟩, タイトル: "*Networks*", 発行年: 1998⟩。図6に *Object1* をテーブル *Object* に格納した例を示す。1 は、*Object1* の *oid* である。作者の値として *Taro* と *Makoto* がある。*no* は、作者の値として、二つある事を示している。

この方法では、オブジェクトが生成され、新しい属性が追加されたとしても、テーブルのスキーマを変更をすることなく、オブジェクト (属性) を追加することが可能になる。同様に、オブジェクトの削除にともなう、属性の削除も容易に行うことが可能となる。

Object	*oid*	*attribute*	*value*	*no*
	1	作者	*Taro*	1
	1	作者	*Makoto*	2
	1	タイトル	*Networks*	1
	1	発行年	1998	1

図 6: *Object* テーブル

3.3 データの ID 化

提案手法では、オブジェクトはオブジェクト識別子 (*oid*)、属性 *att*、値 *v* の組 $\langle$ *oid*, *att*, *v* $\rangle$ の集合として格納される。このため、テーブル *Object* 内には、同じ属性名、値が冗長に存在する。オブジェクト数の増加にともない、記憶量が単調増加してしまう。格納するオブジェクトが少量である場合や情報量が小さければ記憶領域に与える影響は少ない。しかし、大量のオブジェクトやマルチメディアオブジェクト等のように情報量が大きいオブジェクトを扱う場合は、記憶量の増加を抑える事が必要となる。本論文では、オブジェクトを構成する属性及び値が冗長に記憶されることを避けるために、属性名、値もオブジェクトとして扱い、属性と値の関係をこれらのオブジェクト識別子により表す事を考える。このために、以下の二つのテーブルを用いる。

Primitive (*pid*/*OID*, *value*, *type*)

Obj (*wid*/*OID*, *aid*/*OID*, *vid*/*OID*)

ここで *OID* は、オブジェクトの識別子の定義域である。*pid*, *wid*, *aid*, *vid* は、定義域が *OID* である属性である。テーブル *Primitive* には、属性名と値が属性 *value* の値として記憶される。各属性名と値がおのおのオブジェクトとなり、オブジェクト識別子が与えられる。*type* は、値 *value* が属性 (*A*) か値 (*V*) かを示す。図7に、*Object1* と *Object2* を格納した例を示す。

Primitive	*pid*	*value*	*type*
	1	作者	*A*
	2	タイトル	*A*
	3	発行年	*A*
	4	出版社	*A*
	5	*Taro*	*V*
	6	*Networks*	*V*
	7	1998	*V*
	8	*Hanako*	*V*
	9	*Databases*	*V*
	10	1997	*V*
	11	*Makoto*	*V*
	12	*TDU*	*V*

Obj	*wid*	*aid*	*vid*	*no*
	1001	1	5	1
	1001	1	11	2
	1001	2	6	1
	1001	3	7	1
	1002	1	8	1
	1002	2	9	1
	1002	4	12	1

図 7: データの ID 化

テーブル *Obj* のスキーマには、オブジェクトを識別する *wid*、属性を示す *aid*、その値を示す *vid* から構成される。テーブル *Primitive* にはオブジェ

クトを構成する属性及び値の内容が格納される。各々の属性名、値に対して、オブジェクト識別子(*oid*)が与えられている。テーブル *Obj* には、属性と値の関係が、*oid* を用いて格納される。

4 システム構成

4.1 索引

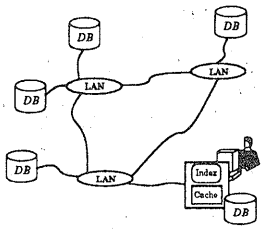

図 8: システム構成図

オブジェクトは、分散されたデータベースに存在する[図 8]。アプリケーションが要求するオブジェクトが、どのデータベースに存在するか知る必要がある。このために、オブジェクトを検索するためには、全てのデータベースに対して検索を行うと、通信と処理の負荷が問題となる。そこで、本論文ではオブジェクトがどこのデータベースに存在するかを明らかにするために、索引を用いる方法を考える。

索引 *Index*(*oid*, *address*) には、システム内のオブジェクト識別子(*oid*)と、そのオブジェクトがどこのデータベースに存在するかを示す情報(*address*)を格納する。*address* は、IP アドレスとポート番号から構成される。オブジェクトが多重化されて複数のサーバに存在する場合には、*address* にシステム上のサーバのアドレスが記憶される。図 9に *Object*1 と *Object*2 を格納したときの索引の例を示す。ここで、*Object*1 は *Server*1 に、*Object*2 は *Server*2 に格納されている。

Index	*oid*	*address*
	1	*Server*1, *Server*2
	2	*Server*1, *Server*2
	3	*Server*1
	4	*Server*2
	5	*Server*1
	6	*Server*1
	7	*Server*1
	8	*Server*2
	9	*Server*2
	10	*Server*2
	11	*Server*1
	12	*Server*2
	1001	*Server*1
	1002	*Server*2

図 9: 索引テーブル

4.2 キャッシング

データベース内には、オブジェクトが格納されている。これらのオブジェクトの中から、必要となるオブジェクトを効率よく検索する方法が必要となる。特に、よく利用されるオブジェクトは、必要とされる度に検索を行うこととなってしまう。さらに分散されたデータベースに存在するオブジェクトを検索することを考えると、ネットワークを用いた通信が必要となる。そこで、一度検索を行ったオブジェクトをクライアントのメモリにキャッシングすることを考える。キャッシングされたオブジェクトを検索する場合には、ネットワーク上のデータベースを検索する必要はなく、キャッシュに存在するオブジェクトを参照することによって、検索時間の短縮を行う。クライアント上の計算機のメモリは有限であるため、検索を行って得たオブジェクトを全て記憶できない。そこで、頻度の低いオブジェクトはキャッシュから削除する。

キャッシュを行うことによって、オブジェクトの値による検索時間を短縮するだけでなく、オブジェクト識別子からオブジェクトを検索する場合も検索時間を短縮させることができる。オブジェクト識別子から検索を行う場合には、索引を参照して、そのオブジェクトがどこのデータベースに存在するかを調べる。クライアントは、データベースを検索した結果、そのオブジェクトをキャッシュに登録し、そのオブジェクトの値を得る。ここで、索引からキャッシュの場所が参照可能なように、索引にキャッシュのアドレス追加する。こうする事によって、次にクライアントが、そのオブジェクト識別子を検索しようとした場合、まず索引を参照し、そのオブジェクト識別子からキャッシュを参照できる。また、クライアントがオブジェクトの値を検索したい場合には、まずキャッシュを参照し、キャッシュにオブジェクトが存在しなければ、データベースの検索することを行う。

4.3 実装

図 10に索引とキャッシュの検索の手順を示す。

索引とキャッシュは、アクセスの高速のためにハッシュを用いて実装されている。

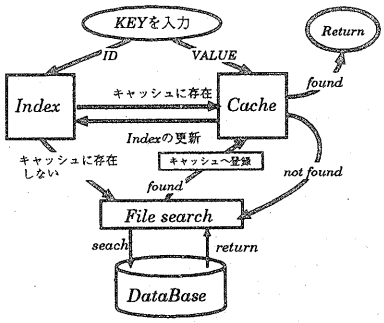

図 10: 索引とキャッシュ

5 評価

索引とキャッシュを用いてデータの検索を行い、その検索時間についての評価を表 1に示す環境で行う。

テーブルへの検索を考える。本評価では、*Primitive* には、表 2に示すような約 23 万行のデータを持つ。本評価では、*Primitive* テーブルは、*Unix* のファイルとして実装されている。ファイルは、逐次的に検索される。また検索するデータは、乱数を発生させ、その値によって検索するデータを前半部分(0 ～ 233925 件)、後半部分 (233936 ～ 236925 件) に

わけ発生させる。ランダムに ID を発生させてからその値を見つけて返すまでの時間を検索時間とする。

ここで、オブジェクトを検索し、そのオブジェクトをキャッシュする時に、既にキャッシュ可能な数(5000件)を超えてしまっていた場合には、キャッシュされているオブジェクト内で一番参照頻度の低いオブジェクトを削除し、検索して得た新しいオブジェクトをキャッシュへの登録を行った。

キャッシュを用いた場合と、キャッシュを用いなかった場合の、検索時間を図11に示す。また、このときのキャッシュのヒット率を表3に示す。これらの結果からも分かる通り、キャッシュに存在する可能性が高い(キャッシュ率が高い)場合には、キャッシュを用いた場合の方が明らかに検索時間が速い事が理解できる。しかし、キャッシュ率が20%を切る辺りからキャッシュを用いない方がキャッシュを用いた場合よりも速くなって来ている。これは、キャッシュへの登録、及びインデックスの更新などによる負荷のためであると考える。しかし、それらの場合でも、ほぼ0.1秒以下程度の遅れでしなないため、キャッシュを用いる有効性は十分あるものと考える。

次にデータの ID 化を行った場合と、行わなかった場合の記憶領域についての評価を行う。本評価では、約18万個のオブジェクトをテーブルに格納する。

ID 化を行わなかった場合と、ID 化を行った場合の記憶領域の比較を表4に示す。この結果より ID 化を行った場合の記憶領域は、行わなかった場合より、約4%減少していることが分かる。本評価では、格納するオブジェクトの情報量が小さいため、約4%の減少となったが、情報量の大きいオブジェクトを扱う場合には、ID 化する有用性はあるものと考える。

表 1: 評価環境

使用マシン	*Enterprise* 450 *Server*
CPU	300 Mhz × 2
Memory	512MB
全データ数	約23万件
キャッシュ数	5000件
データ検索数	10000件

表 2: ファイルの構造

0	*Book*
1	作者
2	タイトル
...	...
236923	*Computers*
236924	*Networks*
236925	*Databases*

表 3: キャッシュのヒット率

データ生成の割合	ヒット率	データ生成の割合	ヒット率
1:9	61.44%	6:4	17.26%
2:8	51.42%	7:3	12.87%
3:7	41.59%	8:2	11.39%
4:6	32.06%	9:1	11.90%
5:5	23.38%	10:0	11.90%

表 4: 記憶領域の比較

ID 化を行なわない		$16906KB$
ID 化を行う	*Primitive*	$4298KB$
	Obj	$11906KB$

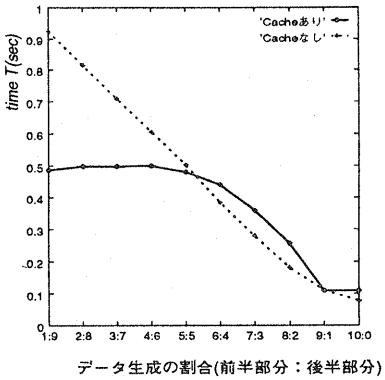

図 11: キャッシュの評価

6 まとめ

本論文では、多種多様な大量のオブジェクトを、ネットワーク上に分散して格納し、検索する方法について論じた。システムの実装、評価を行い、有効性を示した。

参考文献

[1] Date, C. J., "Introduction to Database Systems," *Vol.* 1, *Fourth edition*, *Prentice–Hall*, 1986.

[2] Gosling, J. and McGilton, H., "The Java Language Environment," *Sun Microsystems*, *Inc*, 1996.

[3] Kim, W. and Lochovsky, F.H., "Object Oriented Concepts," *ACM PRESS*, *Inc*, 1996.

[4] Object Management Group Inc., "The Common Object Request Broker," *Architecture and Specification*, *Revision* 2.1, 1997.

[5] "Sybase Adaptive Server Enterprise Transact–SQL ユーザーズ・ガイド Release 11.5.x," , 1997.

[6] "UniSQL/X データベース管理システムユーザーズマニュアル Release 3.0J,"

[7] 日本オラクル, "Oracle 実践アプリケーション開発技法," ソフトリサーチセンター, 1995.

[8] 滝沢 誠, "RDBMS 技術解説," ソフトリサーチセンター, 1996.

[9] 龍 忠光, 泉, 村川, "リアルタイム性を追求したオブジェクト思考設計法ＯＣＡ," 情報処理学会ソフトウェア工学研究会, 1993.

[10] 龍 忠光, 村川, 市川, 豊田, 足立, 戸島, "オブジェクト思考をベースとした部品の再利用について," 情報処理学会データベースシステムとオペレーティングシステム合同研究会, 1993.

[11] 龍 忠光, 泉, 戸島, "自律的オブジェクトのモデリング手法について, " 電子情報通信学会データ工学研究会, 1994.