5W－2　関係データベースと分散ファイルシステム上に構築したオブジェクトサーバの概要とその応用

福田 剛志,　森本 康彦,　何 千山,　小坂 一也
日本アイ・ビー・エム(株) 東京基礎研究所

1　はじめに

近年、マルチメディア情報管理システムの様にさまざまな型を持つ複雑なデータ構造を取り扱う要求が増えてきている。従来の関係データベース (RDBMS) に代わって、オブジェクト指向データベース (OODBMS) がその一つの解になりつつある。

OODBMS は RDBMS に比べて、特に細粒度のオブジェクトの巡行アクセスに関して非常に高い性能を持つが [1]、OODBMS が一般に広く使われるには至っていない。我々は、これは次のような理由によると考える。(a) ほとんどの OODBMS がページサーバ方式を採用しており、大規模データベースに対する質問処理は必ずしも高速でない。(b) アプリケーション作成後に新たに追加された型 (クラス) を使用できない。(c) 新しいアプリケーションでも、RDBMS に格納されているデータ資産に対するアクセスが必要である。

これらの問題に答えるため、我々は COSMOS[2] と呼ばれるプロジェクトにおいて、RDBMS と分散ファイルシステム (DFS) の上に次のような特徴を持つオブジェクト管理層を作った。

1. ほとんどの OODBMS と同様、シームレスな永続 C++ 言語インターフェイスを提供する。
2. 細粒度オブジェクトのクラスタリングにより、パフォーマンスを改善する。
3. メソッドを COSMOS のデータベースに格納し、必要に応じて実行時に結合する。
4. オブジェクトの複製を複数サーバに配置することにより、応答時間の短縮と負荷分散を行なう。

この論文では現在の COSMOS の概要とクラスの動的結合について述べる。

2　システム構成

COSMOS は図 1 に示す様に、既存の RDBMS と DFS のクライアントモジュール上に作られたオブジェクト管理層である。従って、RDB に格納されたデータ資産を使用する際には、これまで通り SQL によるインターフェイスが利用可能である。また、COSMOS 自体は、標準的な RDB の API と、DFS に対する File I/O のみを用いて C++ 言語で記述されているため高い移植性を持っている。実際、COSMOS はアーキテクチャの異るワークステーション上で複数種の RDBMS, DFS 上に移植され稼働している。

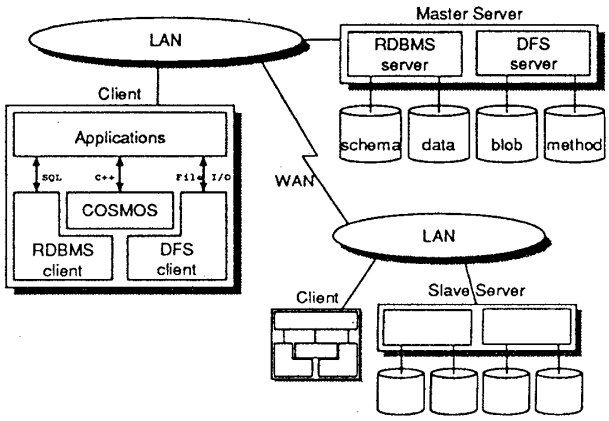

図 1: システム構成

3　オブジェクトモデル

COSMOS のオブジェクトの永続性は、C++ のクラスシステムを用いて、言語拡張を一切せずに実現されている (図 2参照)。COSMOS が定義する永続基底クラス (dbPObj) を継承するクラスは永続クラスとなり、そのインスタンスはデータベースに格納可能となる。それ以外のクラスは一時クラスで、そのインスタンスは通常の C++ と同様、プログラムセッション内でのみ有効なオブジェクトである。

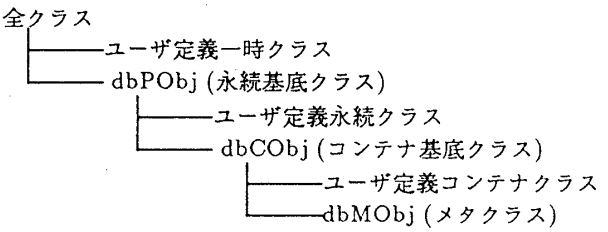

図 2: COSMOS のクラス階層

システム定義のクラス dbCObj のサブクラスのイン

An object server based on relatoinal database systems and distributed file systems: overview and applications
Takeshi Fukuda, Yasuhiko Morimoto, Qinshian He, and Kazuya Kosaka
Tokyo Research Laboratory, IBM Japan Ltd.

スタンスは、性能改善のためのクラスタリングの単位であるコンテナとして扱われる。コンテナは永続オブジェクトの一種である。

全ての永続クラスについて、そのクラス自身を表すメタオブジェクトが、システム定義のクラス dbMObj のインスタンスとして作られる。メタオブジェクトは、コンテナとして実装されている。

4 クラスの動的結合

従来の OODBMS ではメソッドがアプリケーションに静的に結合されてしまうため、メソッドの変更や新しいクラスの追加などの際にアプリケーションの再コンパイル・リンクが必要であった。この問題を解決するためには、メソッドもデータベースで共有管理しなければならない。

最近の多くの OS には動的結合ライブラリ (DLL) または共有オブジェクトなどと呼ばれる機構が用意されており、実行中のプログラムの空間内に動的に手続きやデータを結合し利用することができる。しかし、DLL を呼び出すためには専用のシステムコールを使用するためプログラミングが面倒であった。

COSMOS ではデータベース中に格納されているクラスに属すメソッドを DLL としてデータベースに格納し、C++ の仮想関数と呼ばれるオブジェクトとメソッドの遅延結合の機構を用いることにより、プログラミング上新たな構文・意味を導入することなく、動的に結合されたクラスのメソッド呼び出しを実現した。

オブジェクトの作製などを行なうクラスメソッドの動的な呼び出しが必要になることが多いが、C++ 言語はこれをサポートしていない。そこで、COSMOS ではメタオブジェクトにメソッドの名前からアドレスを問い合わせる機能を与えることで、動的なクラスメソッドの呼び出しを可能にした。

クラスを動的に結合するケースは、データベースに未知クラスの質問をした場合、未知クラスメソッドを呼び出す場合、オブジェクト間のリンク (ポインタ) を辿った場合の 3 種類存在する。アプリケーションとクラスとの結合は、クラスを表すメタオブジェクトをデータベースから (明示的/暗黙に) 読み込む際に行なわれる。

4.1 明示的結合

COSMOS の質問とクラスメソッドの取得は、メタオブジェクトに対してメッセージを送ることによって記述する。メタオブジェクトはデータベースを表すオブジェクトにクラス名を指定して問い合わせる。

```
dbPDBObj* db = new dbPDBObj("dbname");
dbMObj* mobj = db→getMObj("MyClass");
dbQIter<MyClass> qiter =
    mobj→query("name == \"COSMOS\"");
while (qiter.more()) {
    MyClass* object = qiter.next();
    // ...
}
```

質問する対象のクラスは、このプログラムが作成された後で設計、データベースに登録されて構わない。メタオブジェクトをデータベースから読み込む際、メソッドがアプリケーションと結合される。

4.2 暗黙の結合

質問または巡行アクセスで入手した永続オブジェクトが持つ (他の永続オブジェクトへの) リンクを辿る際にも、未知のクラスを読み込む可能性がある。

```
MyClassA* objA = qiter→next();
MyClassB* objB = object→link_to_B;
objB→method();
```

この場合は、リンクを辿る際に自動的に必要なメタクラスがデータベースから読み込まれ、同時にメソッドも結合される。

どちらの場合も永続オブジェクトが読み込まれた際に C++ の遅延結合の機構である仮想関数表 (vtable) へのポインタが、DLL 内のメソッドを指すよう swizzle されるため、アプリケーションは通常の C++ のプログラム同様にデータベースに格納されていた仮想メソッドを呼び出すことができる。

5 おわりに

我々は RDB と DFS 上にオブジェクト管理層を提供することにより、従来のデータ資産を生かしながら複雑なデータ型を取り扱うようなアプリケーションの開発を容易にした。実際、マルチメディア型の商品開発支援情報システムが COSMOS データベースを中核として構築しされ、運用されている。

オブジェクトのデータだけでなくメソッドをデータベースに格納し動的に結合することにより、メソッドの変更や、新しいクラスの追加に対してアプリケーションを安定させることが可能となった。

参考文献

[1] R. G. G. Cattell and J. Skeen. Enginieering database benchmark. Technical report, Sun Microsystems, Apr. 1990.

[2] K. Kosaka, K. Kajitani, Q. He, Y. Morimoto, and T. Fukuda. オブジェクトサーバとその応用. 情報処理学会研究会報告, volume 92-DBS-89, pp. 19–28, July 1992.