

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

早 田 宏[†] 渡 辺 美 樹[†]
田 中 圭^{††} 山 崎 伸 宏[†]

本論文では、オブジェクト指向データベース・エンジン（OODB エンジン）が応用プログラムの複数のシステム・アーキテクチャへ適応する 1 つの方法として、キャッシュの共有に着目した設計空間を提案する。設計空間は、応用プログラムのクライアント・サーバでのホストの位置関係、ホストあたりのプロセス数、およびプロセスあたりのスレッド数というシステム・アーキテクチャにおける 3 つの構成上の選択を軸としている。各軸は、キャッシュに関する 3 つの共有方式（server-client shared cache, multiple-clients shared cache, intra-process shared cache）の採用の有無と対応できる。提案する設計空間に基づき、8 通りのシステム構成の OODB エンジンを拡張可能な手法で実現する。複数のシステム構成の性能を測定し、システム構成による差異を示す。また、応用プログラムへの適用経験から、複数のシステム構成が負荷分散や並列処理の自由度を提供できる点で有効であることを述べる。

An Extensible Object-oriented Database Engine Earth: Its Design Space for Shared Cache

HIROSHI HAYATA,[†] YOSHIKI WATANABE,[†] KEI TANAKA^{††}
and NOBUHIRO YAMASAKI[†]

Goal of this paper introduces a design of an object-oriented database engine for its flexible adaptability to several types of application architectures. The adaptability means that object-oriented database engines can provide good performance on each of the architecture types. To achieve the goal, this paper proposes a design space which represents alternate features among the application types. Its three orthogonal axes are three selections; host location of client-server configuration, client number of each host, and threads number of each process. These selections can introduce three shared cache methods; server-client shared cache, multiple-clients shared cache and intra-process shared cache. The design space also represents design alternatives of shared cache methods. Based on the proposed design space, eight system configurations of object-oriented database engines can be implemented. Earth is an object-oriented database engine which provides the eight configurations with using an extensible implementation method. Performance benchmark results and real application examples are introduced to show the effectiveness of the proposal and the implementation.

1. はじめに

1.1 文書管理応用のためのデータベース・エンジン

今日、文書管理応用プログラムとして、構造化文書管理方式^{23),29)}、CORBA による異種文書管理統合方式²¹⁾、全文検索キーワードのインデックス管理方式¹⁷⁾、可変長・構造を持つ属性や動的に追加可能な拡張属性を含む文書属性管理方式¹⁴⁾などが注目されている。こ

れらの応用プログラムで利用されるデータベース・エンジンでは、複雑な構造のデータを効率的に管理すること、個人レベルからグループ・レベル、インターネット・ユーザのレベルまでの文書数や利用ユーザ数のサービス規模の広がりに対応することが要求される。

筆者らは、これらの要求の解決策として、文書管理応用プログラムに組み込んで利用するオブジェクト指向データベース・エンジン（OODB エンジン）の研究開発を進めている。OODB エンジン^{*}は、単純なオ

[†] 富士ゼロックス株式会社

Fuji Xerox Co., Ltd.

^{††} FX パロアルト研究所

FX Palo Alto Laboratory, Inc.

^{*} 文献 6) では、OODB エンジンは Object Manager として分類されている。

プロジェクト管理機構が必要な応用プログラムやデータベース管理システム (DBMS) のコアとして利用される⁶⁾。OODB エンジンの特性は、限定的なデータ管理機能と、限られた計算機資源での高性能である。たとえば、永続的なオブジェクトの管理機構や単純な並行制御は要求されるが、宣言的な問合せ言語の機能は要求されない。また、実効的な性能のために必要なメモリ占有量が少ないこと (small footprint) が望まれる。

OODB エンジンは、オブジェクト指向モデルにより複雑なデータ構造を効率的に管理できる。しかし、サービス規模の広がりへは十分に対応できていない。応用プログラムは、スタンドアローン構成、クライアント・サーバ構成、3 階層アーキテクチャ (Three-tier architecture) などの複数のシステム・アーキテクチャにより、サービス規模の広がりへ対応している。筆者らは、複数のアーキテクチャに適應する OODB エンジンにより、規模の広がりを考慮した応用プログラムの構築を支援できると考える。本論文では、OODB エンジンがアーキテクチャへの適應性を提供する 1 つの方法を提案する。

筆者らの提案する方法は、複数のシステム構成の OODB エンジンを実現し、応用プログラムの設計者による選択を可能とする方法である。複数のシステム構成を実現するために、応用プログラムのアーキテクチャ上の選択肢を軸として構成する設計空間を定義する。軸上での選択の組合せ (各アーキテクチャ) と OODB エンジンでのキャッシュの共有方式を対応させ、アーキテクチャに応じた性能を提供する。また、複数のシステム構成の OODB エンジンを効率的に実現するため、拡張可能な実現方法を採用する。実現している OODB エンジン Earth^{12),13)} は、システム構成が拡張可能な DB エンジンであり、C++ オブジェクトを永続管理する DB エンジンである。

1.2 キャッシュ共有の課題

オブジェクト指向データベース管理システム (OODBMS) は、永続オブジェクトを応用プログラムから直接参照可能なヒープ領域へ配置し操作する (main memory buffering) 方式により、高性能を実現している^{15),25)}。この方式に基づくクライアント・サーバ構成は、サーバが管理するデータをクライアント側へ転送し、クライアントにてデータを操作するデータ・ SHIPPING (data shipping) の構成となる。そのため、ヒープ領域と二次記憶の間での効率的なデータ転送が、高性能を実現するために重要である⁶⁾。

クライアントのヒープ領域とサーバの二次記憶間のデータ転送は、クライアントとサーバ間でのリモート・

プロシージャ・コール (RPC) によって実現される。RPC にて、長大なデータを転送することや、短いデータを何度も転送することは、性能への影響が大きい。効率的な転送を実現する方法として、転送されたデータをキャッシュし、複数の応用プログラムで共有する方法がある。この方法により、二次記憶への入出力やネットワークによる転送の総量を抑制できる。

汎用的なクライアント・サーバ構成では、複数のクライアント・ホストがローカル・ネットワークにてサーバ・ホストと接続し、あるクライアント・ホストでは複数のクライアント・プロセスが稼動する。クライアント・ホストとサーバ・ホスト間でのデータ転送は避けられない。複数のクライアント・プロセスでのキャッシュの共有によりデータ転送を抑制しようとする、ホストごとの仲介役のキャッシュ・マネージャによりオーバーヘッドが発生する。

汎用的なクライアント・サーバ構成の OODB エンジンを提供することで、複数のアーキテクチャへ適應できる。しかし、汎用性によるオーバーヘッドのため、個々のアーキテクチャに適した性能を提供することができない。アーキテクチャに応じた効率的なデータ転送の達成が課題である。

筆者らは、提案する設計空間の軸であるアーキテクチャ上の選択肢に、キャッシュの共有方式の選択肢を対応させる。これにより、アーキテクチャのバリエーションに対応した共有方式が選択でき、各共有方式により効率的なデータ転送を実現できる。本論文では、この点について詳しく述べる。

まず、2 章で関連研究に対する位置づけを明らかにする。3 章で、クライアント・サーバ構成ならびにプロセス構造からアーキテクチャ上の選択肢を決定し、設計空間の軸として提案する。また、アーキテクチャ上の選択肢を、キャッシュ共有方式の選択と対応させる。4 章で、設計空間に基づく複数のシステム構成の OODB エンジンを説明し、3 レイヤのモジュール構成によって実現することを報告する。5 章で、モジュール構成を利用して実現した複数のシステム構成の性能評価と、応用プログラムへの適用での有効性について記述する。

2. 関連研究

提案する OODB エンジンは、OODBMS^{9),15),27)} のサブセットもしくは、カーネル部 (DB エンジン) として位置づけることができる。また、DB エンジンに関する研究は、Kala²⁴⁾ や、EXODUS の最下層である Wisconsin Storage System (WiSS)⁸⁾ などが知られ

ている。これらのOODBMSにおいても、DBエンジンにおいても、応用プログラムのアーキテクチャに柔軟に対応できるシステム構成は示されていない。ObjectStore¹⁵⁾やPERCIO²⁷⁾等は、ローカル・エリア・ネットワークで接続された複数のクライアントとサーバの構成を提供し、Kala²⁴⁾やMneme¹⁸⁾等は、ローカルホスト内のオブジェクト・サーバでの効率的なデータ管理を提供する。また、POET²²⁾やObjectStore PSE²⁰⁾のように、シングルユーザ版ならびにマルチユーザ版の両方の構成を提供するシステムもある。しかし、これらを実現するための設計空間と、その実現方法は明示的に示されていない。筆者らは、アーキテクチャ上の選択肢を表す設計空間を定義し、その選択肢をキャッシュ共有方式の選択と対応させている。さらに、複数のシステム構成のOODBエンジンを拡張可能な方法で実現する。

一方、柔軟な構成のDBMSを狙う拡張可能なデータベース管理システム(Extensible DBMS)が研究開発されている^{4),5),11),26)}。Extensible DBMSの実現方式には、大きく分けてコンポーネント方式とツールキット方式がある³⁾。コンポーネント方式は、Starburst¹¹⁾やPOSTGRES²⁶⁾に代表される。この方式では、Extensible DBMS内のインデクス構造などのコンポーネントを応用ごとに置き換えることができる。ツールキット方式は、部品群からシステムの合成機能を持つEXODUS^{4),5)}に代表される。コンポーネント方式では、DBMSにイメージ検索やテキスト処理などの新たな応用機能を追加することは容易であるが、キャッシュ管理などの基本機能レベルでの拡張ができない。ツールキット方式では、基本機能から応用機能までの柔軟な追加や選択が可能であるが、部品の使いこなしに課題がある³⁾。筆者らは、DBMSのカーネルに相当するDBエンジンにおいて、3レイヤのモジュール構成によるツールキット方式により、複数のシステム構成を提供することで、応用プログラムでの容易かつ有効なシステム構成の選択を可能とする。

また、ネットワーク上に分散したメモリの共有(分散共有メモリ)に着目して、複数のシステム構成を提供するWAKASHI²⁾がある。WAKASHIは、分散共有メモリ上の永続プログラミング言語システムである出世魚^{2),16)}の最下層で、3つのシステム構成(WAKASHI/B, WAKASHI/C, WAKASHI/D)が実現されている。しかし、これら構成は、分散共有メモリを前提として、共有のためのプロトコルを初期実験、集中制御、分散制御と拡張することで実現している。そのため、応用プログラムの複数のアーキテク

チャに適應するものではない。筆者らは、あるホスト内でのキャッシュ共有に着目した複数のシステム構成で、アーキテクチャの多様性に適應する。ホスト内でのプロセス間ならびにプロセス内でのキャッシュ共有を選択的に可能とすることで、8通りのシステム構成のOODBエンジンを実現している。

3. 設計空間とキャッシュ共有

3.1 設計空間の軸

応用プログラムのアーキテクチャ上の選択肢を軸とした設計空間を提案する。今日、多くの応用プログラムはクライアント・サーバ構成を基本としている。一般にクライアント・サーバ構成といっても、いくつものバリエーションが存在する。そのバリエーションは、応用プログラムごとに選択するクライアント・プロセスとサーバ・プロセスの位置関係や、プロセスの構造などに起因している。本論文で課題とするOODBエンジンのアーキテクチャへの適應性を、アーキテクチャのバリエーションに応じた性能が提供できるかという点で評価する。

個々のアーキテクチャを決定する選択肢として、クライアント・サーバ構成でのホストの位置関係の選択、その構成でのクライアント数の選択、動作するプロセス構造の選択を採用する。これらの選択肢を直交軸とした設計空間を定義する(図1参照)。選択肢の組合せを示す各点が、1つのアーキテクチャである。設計空間は、アーキテクチャの8通りのバリエーションを表現できる。各軸について記述する。

クライアントの位置の軸 クライアント・サーバ構成でのホストの位置関係において、「クライアントとサーバが同一ホストに存在する(LocalClient)か、異なるホストに存在する(RemoteClient)か」を選択する。

ホストあたりのプロセス数の軸 クライアント・サーバ構成でのクライアント数の観点において、「ホストあたりの並列プロセス数が単一(SingleProcess)か、複数(MultipleProcess)か」を選択する。

プロセスあたりのスレッド数の軸 利用するプロセス構造の観点から、「プロセスあたりの並列スレッド数が単一(SingleThread)か、複数(MultipleThread)か」を選択する。

上記の3軸から構成する設計空間は、応用プログラムのアーキテクチャ上の選択肢を表している。

3.2 キャッシュの共有方式

アーキテクチャのバリエーションに応じた性能を提供することが課題である。筆者らは、提案する設計空

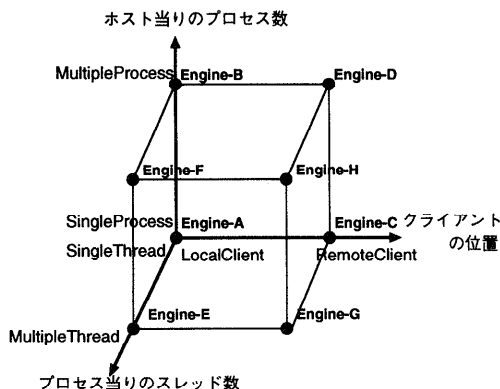


図1 アーキテクチャ上の選択肢を軸とする設計空間

Fig. 1 A design space for alternatives on architecture.

間の各軸が、単一ホスト内でのキャッシュの共有方式に対応できる点に着目する。各軸での選択肢が、複数の共有方式での採用の有無に対応できる。すなわち、アーキテクチャのバリエーションを特定することで、それに適したキャッシュの共有方式が選択でき、バリエーションに応じた性能を提供できる。

単一ホスト内のキャッシュでは、ホスト内の複数プロセス間の共有 (inter-process shared cache) と、プロセス内の複数スレッド間の共有 (intra-process shared cache) が可能である。さらに、クライアント・サーバ構成でのプロセス間での共有では、サーバとクライアントのプロセス間の共有 (server-client shared cache) と、複数クライアントのプロセス間の共有 (multiple-clients shared cache) が可能である。複数のキャッシュの共有方式を同時に実現することで、キャッシュしたデータが有効となる可能性を大きくできる。キャッシュが有効に利用できればできるほど、コストのかかる二次記憶への入出力やデータのネットワーク転送をより抑制できる。しかし、共有方式によっては、キャッシュのコヒーレンスを保証するために新たな管理プロセスや利用プロトコルが必要となる。方式の採用では実行コストも考慮しなければならない。キャッシュの共有方式を以下にまとめる。

inter-process shared cache 同一ホスト内で順次ならびに並列に実行される複数のプロセス間で、キャッシュを共有する。たとえば、マップド・ファイルによるメモリ共有で実現できる。

server-client shared cache サーバとクライアントのプロセス間で、キャッシュを共有する。たとえば、サーバがマップド・ファイル上でキャッシュしたデータを管理し、クライアントにその共有を許可する。採用すれば、

サーバとクライアント間で、実際のデータは転送しない、コストの軽いデータ転送プロトコルが実現できる。

multiple-clients shared cache 複数のクライアント間で、キャッシュを共有する。たとえば、キャッシュ・マネージャ・プロセスがマップド・ファイル上でキャッシュしたデータを管理し、各クライアントにその共有を許可する。キャッシュ・マネージャ・プロセスは、異なるホストにあるサーバとも連動したキャッシュの有効性の判定や、変更や無効化の伝播をすることで、複数ホスト間でのキャッシュ・コヒーレンスを維持する。コヒーレンス維持のプロトコルは、クライアントとキャッシュ・マネージャ間ならびに、キャッシュ・マネージャとサーバの間の2段階となる。複数クライアント間での共有を望まなければ、キャッシュ・マネージャは不要となり、コヒーレンス維持のプロトコルもクライアントとサーバ間の1段階のみで実現できる。

intra-process shared cache 同一のプロセス間で順次ならびに並列に実行される複数のスレッド間でキャッシュを共有する。スレッド間でのヒープ領域の共有機構で実現できる。データベースのキャッシュだけでなく、ヒープ領域内のすべてのデータを共有し、操作できる。そのため、データ操作においては、キャッシュしたデータに関してトランザクションによる並行制御を実施するだけでなく、スレッド間での同期プロトコルが必要となる。

設計空間の各軸はキャッシュの共有方式に対応できる。応用プログラムのアーキテクチャ上の選択肢を表していた設計空間は、OODB エンジンにおけるキャッシュ共有のための設計空間でもある。

4. 設計空間に基づく Earth の実現

筆者らは、提案する設計空間に基づき、8通りのシステム構成から1つが選択可能な OODB エンジン Earth を実現する。Earth のキャッシュ構成の概要を示し、8通りのシステム構成を記述する。さらに、この複数のシステム構成を拡張可能な方式で実現することを記述する。

4.1 Earth のキャッシュ構成

Earth のキャッシュは、ページ・キャッシュとオブジェクト・キャッシュから構成されている。各々のキャッシュを分離し、独立管理する方式で、データベース中

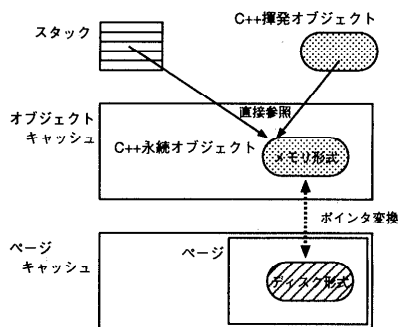


図2 Earthのキャッシュ構造
Fig. 2 Cache structure of Earth.

のデータに対する密なアクセスでは、1回のページ操作で複数オブジェクトを操作できる。また、疎なアクセスでは、必要なオブジェクトのみをオブジェクト・キャッシュに残して、ページ・キャッシュ中のページを入れ替えることができる^{*}。

Earthのクライアント・サーバ構成では、サーバはページ単位でクライアントへデータ転送するページ・サーバ¹⁰⁾である。クライアントは、C++のプログラム・ライブラリとして、応用プログラムとリンクされる。クライアントは、応用プログラムのヒープ領域内に、ページ中の永続オブジェクトをポインタ変換し、応用プログラムから直接操作できるC++オブジェクトを提供する。このポインタ変換はページ・キャッシュとオブジェクト・キャッシュを利用するコピー変換方式¹⁸⁾である(図2参照)。

4.2 8通りのシステム構成

図3にて、各OODBエンジンのシステム構成を説明する。クライアント・サーバ構成でのホスト上の位置関係やクライアント数の選択から、Engine-AからEngine-Dの4つの構成が実現できる。それらの各々についてプロセス構造を複数スレッド化することで、Engine-EからEngine-Hの4つの構成が実現できる。

Engine-Dが汎用性の高いクライアント・サーバ構成であるが、Engine-Dに対してホスト上の位置関係やクライアント・プロセス数を限定することで、他のシステム構成が可能となる。サーバならびにクライアントが同一のホスト上で動作すると限定するならば、Engine-Bが可能となる。Engine-Bはserver-client shared cache方式を採用し、サーバとクライアント間でのデータ転送が効率化できる。また、あるホストでは単一のクライアントのみが動作すると限定するなら

ば、Engine-Cが可能となる。Engine-Cはmultiple-clients shared cache方式を採用していない。共有の可能性は限定されるが、キャッシュ・マネージャを介さないキャッシュ・コヒーレンシのプロトコルが可能となる。また、Engine-BとEngine-Cの両方の限定を合わせた単一ホスト、単一プロセスでの動作では、最も簡単なスタンドアローン構成のEngine-Aが可能となる。Engine-Aでは、複数による共有の考慮はなく、並行処理制御機能も必要なくなる。

Engine-AからEngine-Dの構成では、プロセス内は単一スレッドに限定している。一般に、プロセス内での複数スレッドにより、非同期に発生する処理の応答性の向上、複数処理でのメモリ資源の節約、マルチプロセッサの活用が可能となる。4つの構成を複数スレッド化することで、各々に対応したEngine-EからEngine-Hの構成を実現できる。たとえば、Engine-Bを複数スレッド化するには、クライアント・プロセスに対してintra-process shared cache方式を導入する。これにより、クライアント間でオブジェクト・キャッシュを共有するEngine-Fの構成が可能となる。Engine-Fでは、ヒープ領域内のデータ操作における同期プロトコルが必要となるが、単一のオブジェクト・キャッシュを複数の応用プログラムで利用できる。

4.3 3レイヤのモジュール構成による実現

8通りのシステム構成のOODBエンジンを、個別に実現するのではなく、拡張可能なOODBエンジンとして実現する。実現方式としては、3レイヤでのモジュールから構成するツールキット方式を採用する(図4参照)。これらの複数のシステム構成は、コンポーネント方式では拡張可能な実現ができない。なぜならば、DBMSの基本機能であるキャッシュの共有方式の選択に自由度を持たせることが必要であり、機能の入替えや新たな機能の追加での機能拡張では対応できないからである。

3レイヤは、管理するデータの抽象度のレベルに応じて設定する。各レイヤは、固定長のバイトデータを管理するページ・レイヤ、可変長のバイトデータを管理するレコード・レイヤ、永続化するC++オブジェクトを管理するエンティティ・レイヤである。レイヤ内のモジュールは、必要な機能によって置換えや除去による組合せが可能なモジュールである。3レイヤの構造は、レイヤの独立性を保障して機能モジュールの組合せを容易とすることを狙っている。

前述の複数OODBエンジンの構築方法を説明する。設計空間における“クライアントの位置”の軸での相違は、ページ転送モジュールの置換えで実現する。また、

^{*} ページ・キャッシュとオブジェクト・キャッシュを独立管理する方式により、ワーキングセット・サイズがキャッシュ・サイズを越えた場合でも、性能の大幅な低下を抑制できる。

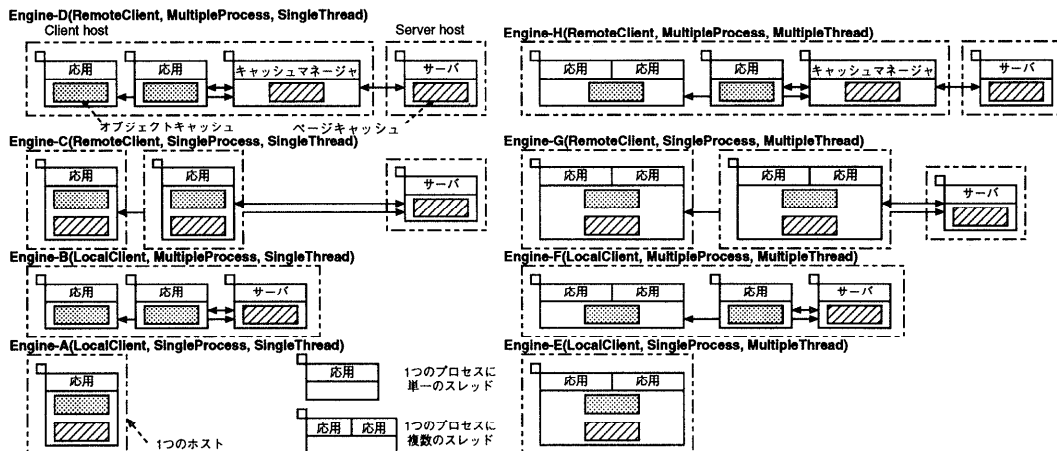


図3 複数のシステム構成の OODB エンジン

Fig. 3 Multiple system configurations of OODB engine.

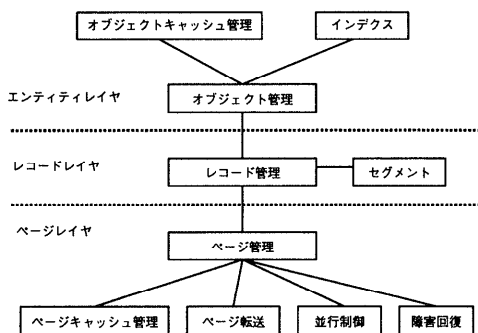


図4 3レイヤのモジュール構成

Fig. 4 Three-layered modules.

“ホストあたりのプロセス数”の軸での相違は、ページ・キャッシュ管理モジュールの置換えで実現する。特に、Engine-A に関しては、並行制御機能が必要ないため、並行制御モジュールも置き換える。また、“プロセスあたりのスレッド数”の軸での相違は、オブジェクト管理モジュールの置換えで実現する☆。

5. 複数のシステム構成の評価

5.1 クライアント・サーバ構成での性能特性

4.2 節で示したとおり、クライアント・サーバ構成の位置関係やクライアント数を限定することで、4 種類のシステム構成 (Engine-A から Engine-D) が実現できる。これらの構成でのキャッシュの共有には、

server-client shared cache ならびに multiple-clients shared cache 方式の採用の有無に差があり、それによりデータ転送やコヒーレンシ維持のプロトコルが異なる。各システム構成上で Object Operation version 1 (OO1) ベンチマーク⁷⁾を動作させ、共有方式の相違による実行性能の差を明らかにする。

共有方式の相違による性能差を明確化するため、各システム構成が適応するハードウェア構成で性能測定するのではなく、最も単純なシステム構成が動作する単一ホスト上で性能を測定する。測定環境は、Sun Sparc 330 (主記憶 40 MB) で、設定したキャッシュの大きさは、ページ・キャッシュ 4 MB、オブジェクト・キャッシュ 4 MB である。OO1 ベンチマークの小規模データベースをローカル・ディスクに作成し、インデクスを利用した条件検索である Lookup 操作、オブジェクト間の巡回操作である Traverse 操作、複数オブジェクトを追加する Insert 操作を実行する。各操作について、データがまったくキャッシュされていない Cold 状態 (キャッシュ Cold) の実行時間と、その後の 10 回の連続操作でキャッシュされたデータを利用する Warm 状態 (キャッシュ Warm) での平均実行時間を得る。図 5 は、キャッシュ Cold とキャッシュ Warm の各々での Lookup, Traverse, Insert 操作の総和を示す。縦軸は、最も効率的な Engine-A の実行時間に対する相対実行時間である。

各システム構成とも、キャッシュされたデータの操作自体にかかるコストは一定である。性能差の要因は、二次記憶上のデータベースファイルから、クライアントのメモリ領域までの転送コストと、キャッシュされたデータの正当性を確認するための検証コストにある。

☆ 現在の実装においては、これらのモジュールの置換えにともない、他のモジュールの一部で、マクロ文によるコンパイル時の切替えが必要である。これは、実装上の課題であり、モジュール間のインタフェースを整備することで、より高い独立性のモジュールは実現できる。

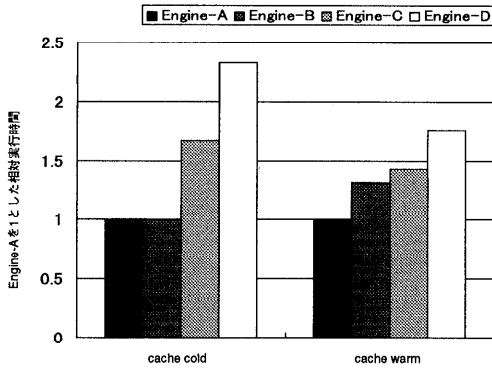


図5 複数のクライアント・サーバ構成での性能
Fig. 5 Performance on selection of client-server configuration.

両コストとも、キャッシュCold, Warmにかかわらず発生する。しかし、操作に必要なデータの転送量は異なる。データがまったくキャッシュされていないキャッシュColdでは、データの転送量が多く、転送コストの影響が大きく表れる。キャッシュされたデータを操作するWarmでは、データの転送は完全にキャッシュされるまでと、Insert操作での書き戻しに限定され、転送コストと検証コストの両者の影響が表れる。

Engine-AとEngine-Bの共有方式の相違は、multiple-clients shared cache方式の採用の有無である。この差は、Warmでの検証コストの差として表れている。その差は、並行制御方式の相違による。キャッシュを利用するクライアント数が単一のEngine-Aでは、他の構成と異なり並行制御が不要である。並行制御はサーバ主導での二相施錠方式で実現しているため、他の構成では、ロック操作やコミット操作において、プロセス間のRPCが必要となる。また、データ転送コストに関しては、ともに、server-client shared cache方式により転送コストを抑制しているので、キャッシュColdの性能がほぼ同じとなっている。

Engine-AとEngine-Cの共有方式の相違は、server-client shared cache方式の採用の有無である。この差は、転送コストの差として表れる。特に、キャッシュColdでは、クライアントとサーバ間で大量のデータをリモート転送する際の影響が表れている。また、キャッシュWarmでは、転送コストの影響と、前述の並行制御での検証コストの影響が合わさって、性能差として示される。

Engine-AとEngine-Dの共有方式の相違は、server-client shared cacheとmultiple-clients shared cacheの両方式の採用の有無である。Engine-Dは、前述のプロセス間でのデータ転送コストならびに並行制御の

検証コストが、Engine-Cと同様に発生している。これに加えて、Engine-Dでは、クライアント・ホスト内の複数クライアント間でのキャッシュのコヒーレンスを維持するキャッシュ・マネージャの影響で、転送コストも検証コストもさらに大きなものとなる。転送コストに関しては、サーバからキャッシュ・マネージャ、キャッシュ・マネージャからクライアントへの2段階での転送が発生し、その影響が特にキャッシュColdの性能に表れている。

5.2 プロセス構造での性能特性

また、4.2節で示したとおり、複数スレッド対応により、さらに4種類のシステム構成(Engine-EからEngine-H)が実現できる。単一スレッド対応と複数スレッド対応でのキャッシュ共有には、intra-process shared cache方式の採用の有無に差があり、それによりヒープ領域内のデータ操作における同期プロトコルの必要性が異なる。Engine-Bと、その複数スレッド対応であるEngine-Fにおいて、前述のOO1ベンチマークを動作させ、キャッシュ共有の相違による実行性能の差を明らかにする。

複数スレッド対応における性能差を明確化するために、OO1ベンチマークで定義している操作を単一ホスト上で並列実行させて、性能測定する。測定環境は、IBM PC 互換機(Pentium 100 MHz, 主記憶32 MB)上のWindows-NT 3.51で、設定したキャッシュの大きさは、ページ・キャッシュ2 MB、オブジェクト・キャッシュ2 MBである。OO1ベンチマークの小規模データベースをローカル・ディスクに作成し、Lookup操作とTraverse操作の各々について複数クライアントで並列実行する。Engine-Bでは、複数クライアントは複数のプロセスとして動作し、各プロセスがオブジェクト・キャッシュを占有し、操作する。Engine-Fでは、複数クライアントは単一プロセス内の複数のスレッドとして動作し、全スレッドは単一のオブジェクト・キャッシュを共有し、操作する。図6は、各操作において、並列動作するクライアント数を変化させた場合の、終了時間^{*}の推移を示す。クライアント数1では、Engine-BとEngine-Fでの終了時間は同じである。図6の縦軸は、クライアント数1での終了時間に対する相対時間である。

Engine-BとEngine-Fでは、intra-process shared cache方式の採用の有無によって、占有メモリ量が異なる。Engine-Bでは、プロセスごとにオブジェクト・

^{*} 同時に複数のクライアントで、操作を開始し、すべての操作が終了するまでの時間である。

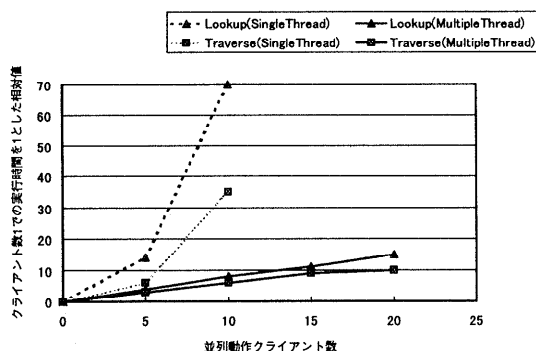


図6 プロセス構造の相違による並列処理の性能

Fig. 6 Performance on selection of process structure.

キャッシュを保有するために、2 MB のクライアント数倍を占有する。各プロセスは、ヒープ領域内のデータを、他のプロセスの動作とは独立に操作できる。Engine-F では複数スレッドがキャッシュを共有するために、クライアント数にかかわらず 2 MB のみを占有する。各スレッドは、ヒープ領域内のデータ操作において、同期プロトコルの実行が必要である。

同期プロトコルの実行の有無よりは、占有するメモリ量の差が、並行動作での性能差に表れる。Engine-F では、並行動作するクライアント数倍に終了時間がとどまっている。Intra-process shared cache により、Engine-F のメモリ共有がすすみ、メモリ使用量が複数クライアント並行動作時でも、1 クライアントのみの動作時とほぼ同じにとどまっている。しかし、Engine-B では、並列クライアント数が増加するに従って、それに比例する以上に終了時間が長くなってしまふ。10 クライアントでの並行動作においては、1 クライアント動作時の約 50 倍の時間がかかる。10 クライアントでの Engine-F のスループットに対しては、約 1/7 である。これは、クライアントごとでオブジェクト・キャッシュ内のデータを操作するコスト（たとえばポインタ変換のコスト）が必要な他に、占有するメモリ量が増加するにともなう実メモリを圧迫し、プロセスのページングが多発することが影響している。

一方で、Insert 操作のような書き込み操作を並列したスレッドで動作させる場合、Engine-F においては問題が発生する。複数スレッド対応の OODB エンジンにおいては、トランザクションによる並行制御によるロック操作と、ヒープ領域のデータを操作するための同期操作の間でデッドロックが発生する。複数スレッド対応では、デッドロックを回避するために、トランザクションの利用に制約を設ける¹⁹⁾か、発生するデッドロックを自動的に解消する機構が必要である。

5.3 応用での複数システム構成の利用

複数のシステム構成を持つ OODB エンジンを利用した 2 つの応用プログラムの事例を紹介する。1 つは、イントラネットにおける文書検索サービスであり、もう 1 つは、文書管理ミドルウェアでの属性管理機構である。

イントラネットにおける文書検索サービスを、OODB エンジンを利用して実現した。検索サーバは、3 階層アーキテクチャでの応用サーバ (application server) に位置づけられる。ユーザは、WWW クライアントから検索サービスを利用する。検索サーバは、検索要求に合わせた問合せを OODB エンジンへ発行する。OODB エンジンは、語と URL の対応や URL の属性などを管理し、発行された問合せを処理する。

検索サーバへのアクセス数や検索対象文書の規模に応じて、OODB エンジンのシステム構成が決定された。10 万文書程度までの小規模なサービスにおいては、検索サーバと、OODB エンジンが単一のホストで共存できた。この場合、Engine-A のシステム構成が選択された。しかし、検索サービスの規模が拡大し、検索対象を数百万文書程度まで増加させ、サービスのターンアラウンド・タイムを向上させることが要求された。この要求に対しては、応用サーバが検索要求を複数のホストヘディスパッチするアーキテクチャが適正と考えられた。そのため、複数ホストへの負荷分散が可能な Engine-C の構成が選択された。

文書管理ミドルウェアのサーバ側での属性管理を、OODB エンジンを利用して実現した。サーバはグループレベルで利用する文書群を管理する。OODB エンジンは、文書群に対する基本ならびにユーザ拡張の属性情報を管理する。属性としては、構造を持つデータや可変長のデータの格納も可能である。OODB エンジンは、単一のサーバホストで動作し、ユーザ管理などの他のサブシステムと共存している。

ミドルウェア・サーバの並列処理の規模に応じて、OODB エンジンのシステム構成が決定された。並列処理数が限定されていた際は、単一ホストで複数クライアントが動作する Engine-B の構成が選択された。しかし、単一ホストのままで並列処理数を向上させることが要求された。この要求に対しては、並列数に依存せずメモリ占有量を抑制できる複数スレッド対応が適正と考えられた。そのため、複数スレッドでの並列処理が可能な Engine-F の構成が選択された。

6. ま と め

本論文での課題は、応用プログラムのアーキテク

チャへの OODB エンジンの適応性である。適応性は、アーキテクチャのバリエーションに応じた性能が提供できるかという点で評価している。アーキテクチャ上の選択肢を表す設計空間を提案し、アーキテクチャ上の選択肢をキャッシュの共有方式と対応させることで、アーキテクチャに応じた性能を実現することを提案している。バリエーションに対応する 8 通りのシステム構成を可能とする OODB エンジン Earth を実現し、バリエーションの差異、すなわちキャッシュの共有方式の差異による性能の影響を示すことができた。

キャッシュの共有方式の差異により、性能に有意な差があることを、OO1 ベンチマークの結果が示している。クライアント・サーバ構成における共有方式の差は、キャッシュ Cold で 2 倍以上の性能差をもたらす。また、プロセス構造における方式の差は、10 クライアントの並列処理で約 7 倍の性能差をもたらしている。データ規模やデータのアクセスパターン、共有方式の実装の変化により、これらの性能差はある程度は変動すると予測する。しかし、データ・ SHIPPING 方式のクライアントサーバ構成の DBMS や DB エンジンにおいて、キャッシュの共有方式を選択可能とする考え方は広く利用可能であるといえる。

3 レイヤのモジュール構成による拡張可能な実現方式は、複数プログラミング言語対応という観点からも有効と考える。Earth は、C++ オブジェクトを永続管理する OODB エンジンとして設計し、C++ 言語にて実装している。今日、ネットワーク環境でのプログラミング言語としての Java 言語に注目が集まり、Java オブジェクトの永続管理に関する研究¹⁾が進められている。C++ 対応の OODB エンジンを、Java 対応へと拡張する²⁸⁾際も、3 レイヤのモジュール構成を利用して、多様な実現が可能である。今後は、Java 対応を含めて、実用的な OODB エンジンを目指した課題の解決を試みていきたい。

謝辞 オブジェクト指向データベース・エンジンの研究開発に関して、貴重な意見やフィードバックを下さった富士ゼロックス株式会社のドキュメント工学研究所の滝口孝一所長をはじめとした皆様に感謝いたします。また、本論文に対して助言を下さった九州大学の牧之内顕文教授に感謝いたします。

参考文献

- 1) Atkinson, M.P., Jordan, M.J., Daynes, L. and Spence, S.: Design Issues for Persistent Java: A type-safe, object-oriented, orthogonally persistent system, *Proc. Seventh*

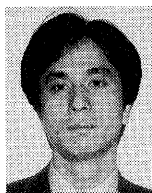
Int'l. Workshop on Persistent Object Systems (1996).

- 2) Bai, G. and Makinouchi, A.: WAKASHI/D : A Distributed Paged-Object Server for Storage Management of New Generation Databases, *ADTI '94*, pp.137-151 (1994).
- 3) Carey, M.J. and DeWitt, D.J.: Of Objects and Databases: A Decade of Turmoil, *Proc. 22nd VLDB*, pp.3-14 (1996).
- 4) Carey, M.J., DeWitt, D.J., Graefe, G., Haight, D.M., Richardson, J.E., Schuh, D.T., Shekita, E.J. and Vandenberg, S.L.: The EXODUS Extensible DBMS Project: An Overview, *Readings in Object-Oriented Database Systems*, Zdonik, S.B. and Maier, D. (Eds.), pp.474-499, Morgan Kaufmann (1990).
- 5) Carey, M.J., DeWitt, D.J., Richardson, J.E. and Shekita, E.J.: Object and File Management in the EXODUS Extensible Database System, *Proc. 12th VLDB*, pp.91-100 (1986).
- 6) Cattell, R.G.G.: *Object data management: object-oriented and extended relational database systems*, Rev. ed., Addison-Wesley (1994).
- 7) Cattell, R.G.G. and Skeen, J.: Object Operations Benchmark, *ACM Trans. Database Syst.*, Vol.17, No.1, pp.1-31 (1992).
- 8) Chou, H.-T., Dewitt, D.J., Hatz, R.H. and Clug, A.C.: Design and Implementation of the Wisconsin Strage System, *Software-Practice and Experience*, Vol.15, No.10, pp.943-962 (1985).
- 9) Deux, O., et al.: The O₂ System, *Comm. ACM*, Vol.34, No.10, pp.34-48 (1992).
- 10) DeWitt, D.J., Fattersack, P., Maier, D. and Véléz, F.: Three Alternative Workstation-Server Architectures, *Building an Object-Oriented Database System, The Story of O₂*, FrançoisBancilhon, C.D. and Harrus, G. (Eds.), pp.411-446, Morgan Kaufmann (1992).
- 11) Haas, L.M., Chang, W., Lohman, G.M., McPherson, J., Wilms, P.F., Lapis, G., Lindsay, B., Piraresh, H., Carey, M.J. and Shekita, E.: Starburst Mid-Flight: As the Dust Clears, *IEEE Trans. Knowledge and Data Eng.*, Vol.2, No.1, pp.143-160 (1990).
- 12) 早田 宏, 佐藤直人, 小部正人: OODBMS Earth における Core の設計と実装, 情報処理学会データベース研究会資料, 92-DBS-89, pp.59-68 (1992).
- 13) 早田 宏, 渡辺美樹, 田中 圭, 山崎伸宏: 組み込みに適した拡張可能なオブジェクト指向データベース・エンジン Earth の実現, 富士ゼロックステクニカルレポート, No.10, pp.98-107 (1995).
- 14) 窪野哲光: Wrapper 方式に基づくハイパーメディア型文書管理システムの検討, 情報処理学会

- データベース研究会資料, 96-DBS-108, pp.73-80 (1996).
- 15) Lamb, C., Landis, G., Orenstein, J. and Weinreb, D.: The ObjectStore Database System, *Comm. ACM*, Vol.34, No.10, pp.50-63 (1992).
 - 16) 牧之内顕文: マルチメディアデータベースのためのオブジェクト指向永続プログラミング言語族, 情報処理学会データベース研究会資料, 91-DBS-84, pp.201-208 (1991).
 - 17) Moffat, A. and Zobel, J.: Efficient Information Retrieval, *DASFAA '97 Tutorial Notes* (1997).
 - 18) Moss, J.E.B.: Working with Persistent Objects: To Swizzle or Not to Swizzle, *IEEE Trans. Softw. Eng.*, Vol.18, No.8, pp.657-673 (1992).
 - 19) Object Design, Inc.: ObjectStore C++ API User Guide Release 4 (1995).
 - 20) Object Design: <http://www.odi.com/products/products.html> (1997).
 - 21) Paepcke, A., Cosins, S.B., Gracia-Monlina, H., Hassen, S.W., Ketchpel, S.P., Roscheisen, M. and Winograd, T.: Using Distributed Objects for Digital Library Interoperability, *IEEE Computer*, pp.61-68 (1996).
 - 22) POET Sofyware: <http://www.poet.com/techover/> (1997).
 - 23) Sacks-Davis, R., Arnold-Moore, T. and Zobel, J.: Database Systems for Structured Documents, *ADTI '94*, pp.272-283 (1994).
 - 24) Simmel, S.S. and Godard, I.: The Kala Basket, *Proc. OOPSLA '91*, pp.230-246 (1991).
 - 25) Singhal, V., Kakkad, S.V. and Wilson, P.R.: Texas: An Efficient, Portable Persistent Store, *Proc. 5th Int'l. Workshop on Persistent Object Systems* (1992).
 - 26) Stonebraker, M., Rowe, L.A. and Hirohama, M.: The Implementaion of POSTGRES, *IEEE Trans. Knowledge and Data Eng.*, Vol.2, No.1, pp.125-142 (1990).
 - 27) 鶴岡邦敏, 木村 裕, 波内みさ, 安村義孝: オブジェクト指向データベース管理システム PERCIO の開発と今後, 電子情報通信学会論文誌, Vol.J79-D-I, No.10, pp.587-596 (1996).
 - 28) 渡辺美樹, 早田 宏: Java のためのデータベース・エンジンの設計課題, 情報処理学会データベース研究会資料, 97-DBS-113, pp.125-130 (1997).
 - 29) 吉川正俊: 構造化文書とデータベース, *Proc. Advanced Database Symposium '95*, pp.49-57 (1995).
 - 30) Zdonik, S.B. and Maier, D. (Eds.): *Readings in Object-Oriented Database Systems*, Morgan Kaufmann (1990).

(平成 9 年 12 月 25 日受付)

(平成 10 年 4 月 3 日採録)



早田 宏 (正会員)

昭和 35 年生。昭和 60 年京都大学大学院工学研究科情報工学専攻修士課程修了。同年富士ゼロックス (株) 入社。Lisp プログラミング環境の開発, オブジェクト指向データベースの研究開発に従事。現在ドキュメント工学研究所副主任研究員。ACM 会員。



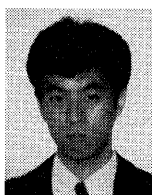
渡辺 美樹 (正会員)

昭和 38 年生。昭和 63 年筑波大学大学院修士課程理工学研究科修了。同年富士ゼロックス (株) 入社。オブジェクト指向分析設計支援ツール, 構造化文書データベース, オブジェクト指向データベースの研究に従事。現在, ドキュメント工学研究所に所属。日本ソフトウェア科学会会員。



田中 圭

昭和 42 年生。平成 2 年九州大学工学部応用原子核工学科卒業。同年富士ゼロックス (株) 入社。オブジェクト指向データベースの研究に従事。平成 8 年より FX パロアルト研究所研究員。日本ソフトウェア科学会会員。



山崎 伸宏 (正会員)

昭和 44 年生。平成 3 年筑波大学第 3 学群情報学類卒業。同年富士ゼロックス (株) 入社。オブジェクト指向データベースの研究に従事。現在, ドキュメント工学研究所に所属。