OGSA-WebDB: グリッド・サービス・アーキテクチャ上でWebデータベースをアクセス可能にするシステムの実現

サイド　ミルザ　パレビ† 　　小島　功†

† 産業技術総合研究所グリッド研究センター　〒305–8568 つくば市梅園 1–1–1
E-mail: †{mirza,kojima}@ni.aist.go.jp

あらまし　本稿では、グリッド上で SQL で Web データベースへの問合せを可能にする OGSA-WebDB システムの実現について述べる．OGSA-WebDB はプロキシデータベースとメディエータから構成されている．プロキシデータベースは Web データベースをグリッド内に代理し、メディエータは SQL 問合せをグリッドアプリケーションから受け取り、Boolean 問合せに変換した後 Web データベースに送り、結果をプロキシデータベースに挿入する．SQL 問合せ処理は Web データベースの特性を考慮しながら並行に行われる．また、システムの応答時間を短縮するために、プロキシデータベースに蓄積されたデータをキャッシュとして利用するアルゴリズムを提案する．最後に、OGSA-WebDB を Globus Toolkit (GT) と OGSA-DAI ソフトウェアを用いて実装したので、その実用性を検証する．

キーワード　グリッド、 web データベース、OGSA、メディエータ

OGSA-WebDB: An OGSA-Based System that Enables Accessing Web Databases from within the Grid

Mirza PAHLEVI SAID† and Isao KOJIMA†

† Grid Technology Research Center, AIST,　1–1–1 Umezono, Tsukuba 305–8568 Japan
E-mail: †{mirza,kojima}@ni.aist.go.jp

Abstract　In this paper, we propose an OGSA-based system, called OGSA-WebDB, that enables the grid (database) applications to query web databases using the standard database query language (SQL). OGSA-WebDB uses *proxy databases* that delegate web databases and a *mediator* that bridges the gap between the proxy databases and the delegated databases. The mediator accepts an SQL query from grid applications. It then transforms the query into one or more Boolean conditions sent to the target web databases, and inserts the returned result into the proxy databases. The SQL query processing is done in parallel taking into account the characteristics of web databases. Beyond that, we propose a simple algorithm to use data that has been retrieved into the proxy databases to shortern the system's response time. Finally, to show the applicability of OGSA-WebDB, we have fully implemented the system on the top of the Globus Toolkit and OGSA-DAI software components.

Key words　grid, web database, OGSA, mediator

1. はじめに

グリッド [18], [19] は科学や商用などのアプリケーションで広く使われつつある．これらの大規模アプリケーションは、大量・分散したデータの保存・管理の必要性があるため、グリッド環境の中でデータベースを統合利用する要求が高まってきている [27]．このために、国際的な団体である GGF（Grid Global Forum）でデータアクセスと統合サービスワーキンググループ（DAIS-WG）が構成され、グリッド上でのデータベース統合の標準仕様を策定しつつある．

グリッド上でデータ処理を行うユーザは、分散した複数の情報源からのデータを統合利用したい場合がしばしばある．そして、それらの情報源は Web 上に存在するものが多い．例えば、新しい薬を開発する生物学者が薬特許データベース [16] や生医学文献引用データベース [7] のインターネット上のデータをグリッド上にある DNA・蛋白質データベースのデータと統合して、データ処理をすることなどが考えられる．

一般に、ほとんどの Web データベースの問合せは Boolean 条件で表現され、HTTP リクエストで送られる．したがって、JDBC のようなデータベースドライバによるアクセスや SQL のようなデータベース言語での問合せは扱えない．さらに、データベースの所有者はデータ保護の理由などで自分のデータベー

スに関するメタデータを公開したくない場合も多い．

現在策定中の DAIS-WG 仕様 [6], [11], [13], [14] ではアプリケーションがデータベースをアクセスする場合は、あるデータベース言語で問合せを書き、特定のデータベースドライバで問合せを送る必要がある．また、その仕様はデータベーススキーマやドライバやサポートするデータベース言語などのデータベースメタデータを公開する必要がある．これらの「**グリッドデータベース要件**」は、実際には Web 上に存在する有用な情報へのグリッドからのアクセスを困難にしている．

グリッド上から Web データベースをアクセス可能にする一つの方法としてはメディエータ [29] の利用が考えられる．これは、グリッドアプリケーションからの問合せを受け取り、その問合せを Web データベースに送り、結果をアプリケーションに渡す．グリッド／ Web データベースを効果的に仲介できるために、メディエータは以下の「**グリッド特有要件**」を満たす必要がある．

（1） グリッドアプリケーションは Web データベースを統合的にアクセス可能にしなければならない．つまり、メディエータは Web データベースを上のグリッドデータベース要件を満たすようにサポートしなければならない．

（2） メディエータはプラットフォーム独立であり、かつグリッドの要求する仕様を満足しなければならない．

（3） メディエータはグリッドセキュリティ認証基盤（GSI）を用いて、認証を必要とする Web データベースをアクセス可能にしなければならない．

本稿では、グリッド・サービス・アーキテクチャ（OGSA）に基づいてグリッド上から Web データベースを統合利用可能にする OGSA-WebDB システムを提案する．OGSA-WebDB はグリッド特有要件を満たすように設計・実装されており、グリッド／ Web データベースの仲介を行うために、Web データベースの特徴を考慮した動的な問合せ実行アルゴリズムを使用する．このアルゴリズムは対象とする複数の Web データベースに問合せを並行に送り、結果を Web データベースを代理するプロキシデータベースに挿入する．グリッドアプリケーションは Web データベースに直接にアクセスする代わりに、プロキシデータベースをアクセスする．

システムの応答時間を短縮するために、プロキシデータベース上のデータをキャッシュとして利用するアルゴリズムを提案する．このアルゴリズムは意味的キャッシュ（semantic cache）の概念に基づいている．最後に、OGSA-WebDB を実装し、文献データベースや化学データベース統合のアプリケーションなどを実現し、その有効性を確かめた．

本稿の構成は以下の通りである． 2節では関連研究を取り上げる． 3節では現在のグリッド上でのデータベースのアクセスについて記述する． 4節では OGSA-WebDB のシステム構成と問合せ処理を詳しく説明する． 5節ではシステム実装について述べる． 6節では OGSA-WebDB の拡張として意味的キャッシュの利用について述べる． 7節でまとめと今後の課題について述べる．

2. 関 連 研 究

Web ／データベースの分野では長い間、異種分散情報源統合利用の研究が行われている．基本的に、情報源統合利用の方法には *materialized/warehousing* アプローチと *virtual* アプローチがある [17]．Materialized アプローチ [31], [32] では、複数の情報源からのデータは中央ウェアハウス／データベースにロードされ、問合せはそのウェアハウスのデータに対して作成・実行される．このアプローチは短い応答時間を持つが、データの新鮮さを維持するのは重要な課題となっている．

一方、Virtual アプローチでは、問合せはある特定のデータ統合アプリケーションの仮想リレーションからなる情報源の統合ビュー (mediated schema) に対して作成・実行される．データは中央ストレージにロードされないので、問合せはいくつかのサブ問合せに分解され、リモート情報源に送られる．このアプローチはデータが頻繁に変わる情報源と自立的に管理される情報源に対して適している．このために、Web データベースと既存のデータベースの情報統合の研究はこのアプローチに焦点を置いている [20], [22], [25]．また、このアプローチは生医学データのようなより特殊なデータをもつ情報源の統合にも使われた [10], [21], [30]．

これらのシステムは先のグリッド特有要件を満たさないので、そのままではグリッド／ Web データベースのメディエータとして使用できない．例えば、いくつかのシステムはグラフ構造あるいはオブジェクト指向概念に基づく言語 [20], [32] を使ったり、関数プログラム言語を使ったりする [10], [21]．また、グリッド認証基盤に対応しているものは知られていない．

OGSA-WebDB は、これらのアプローチの技術を組み合わせることで、グリッド特有要件を満たすことができる．例えば、OGSA-WebDB では、Web データベースを関係スキーマとしてモデル化し、プロキシデータベースとして materialize している．これによって、Web データベースとグリッド内のデータベースを統一的に SQL でアクセスでき、グリッド上での DBMS の問合せ処理能力を使用できる．

IBM DB2 Information Integrator [22] はグリッド／ Web データベースのメディエータとしても使用できる．このシステムは DB2 に Web データベースを取り入れることができ、もし DB2 はグリッド環境上にあれば、グリッドクライアントは取り入れた Web データベースをアクセスできる．しかし、Web データベースをアクセスする応用では、このメディエータの使用はプラットフォーム独立かつ効率的ではないと考えられる．これは、Information Integrator はグリッドアプリケーションに対して特定の DBMS（つまり IBM DB2）からだけの Web データベースのアクセスを許すからである．また、できるだけ多くの DBMS に対応するグリッドデータベース仕様の目的とも一致していない(注1)．

グリッド上での分散問合せ処理の研究は既に行われている．

(注1)：現在、OGSA-DAI は RDBMS（MySQL、IBM DB2、Oracle）と XML DBMS (Xindice) をサポートしている．

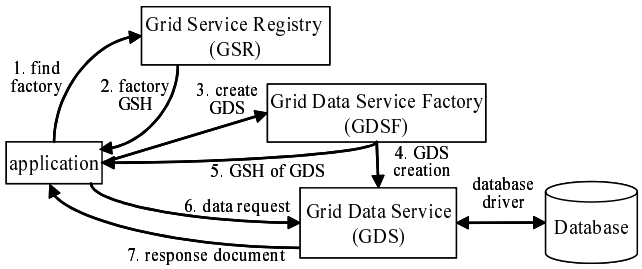

図 1　OGSA-DAI のシステム構成

OGSA-DQP [26] はその例である．このシステムは Polar* システム [28] に基づいて作られているシステムであり、グリッド上にある多様な DBMS を統合するものである．OGSA-WebDB は Web データベースをグリッド上からアクセス可能にするシステムで、OGSA-DQP と補完的に組み合わせることができる．

3.　グリッド上におけるデータベースのアクセス

OGSA [19] は GGF で提案されているソフトウェア構成モデルであり、あるサービスを OGSA に準拠するために、グリッドサービス基盤（OGSI）と呼ばれる仕様が定義された．Globus プロジェクトは現在 OGSI 仕様を実装しており、Globus Toolkit (GT) と呼ばれるオープンソース・ソフトウェアを公開している．

DAIS-WG によって定義されたグリッドデータベースサービス仕様 [11] は OGSA に基づくグリッド上でのデータアクセスと統合の仕様である．UK e-Science Programme プロジェクトで開発された OGSA Data Access and Integration (OGSA-DAI) [6] はその仕様を実装するソフトウェアであり、GT の上で動作する．OGSA-DAI はデータベースをアクセスするために、グリッドデータサービス（GDS）を定義し、アプリケーションは GDS でデータベースとのデータのやり取りを行う．

図 1 はグリッド上でのデータベースアクセスを示す．まず、アプリケーションはグリッドサービスレジストリ（GSR）に、接続先となる GDS を生成するためのグリッドデータサービスファクトリ（GDSF）を要求する（ステップ１）．アプリケーションは GDR からその GDSF のグリッドサービスハンドル (GSH) を受け取った後（ステップ２）、GDS 作成要求をその GDSF に送る（ステップ３）．GDSF は GDS を作成し（ステップ４）、新しく作られた GDS の GSH をアプリケーションに渡す（ステップ５）．最後に、アプリケーションはその GSH でデータベースとデータのやり取りを行う（ステップ６）．

データベースをアクセスするためには、GDSF と GDS が対象データベースのメタデータを知る必要がある．そのメタデータにはデータベーススキーマやデータベースドライバなどが含まれる．また、アプリケーション側もデータベース言語で問合せを作成するため、対象となるデータベースのスキーマを知る必要がある．

4.　OGSA-WebDB

4.1　システムの概要

本稿では、Web データベースの検索インタフェースは少なくとも単純な連言問合せ（AND オペレータで結ばれるキーワード）をサポートすると仮定する．また、検索インタフェースはフィールド検索をサポートしても良いし、ユーザ認証を要求するものでも良い．また、問合せや認証情報などは HTTP リクエストで送られると仮定する．

本システムでは関係データベース（RDBMS）を対象とする(注2)．これは、RDBMS は広く普及したデータベースであり、多くのデータは RDBMS で管理できるからである．また、問合せは SQL の SELECT 文による検索（以降、SQL 問合せと呼ぶ）を考える．

グリッド特有要件を満たすグリッド／ Web データベース仲介システムを実装するために、**プロキシデータベース**と**メディエータ**からなるシステムを構築する．プロキシデータベースは、グリッド上に設けられるデータベース（グリッドアプリケーションからアクセスできるデータベース）であり、Web データベースを代理する役割を持つ．各 Web データベースはプロキシデータベースの中にある１個の**プロキシリレーション**に代理される．したがって、グリッドアプリケーションは Web データベースを直接にアクセスする代わりに、その Web データベースのプロキシリレーションをアクセスする．

一方、メディエータはプロキシデータベースと代理される Web データベースとの間を仲介する．メディエータはグリッドアプリケーションから送られる SQL 問合せを受け取り、それを Web データベースに処理可能な Boolean 問合せに変換する．検索インタフェースの問合せ処理能力は非常に限定されるので、メディエータは検索インタフェースが処理可能な問合せの部分だけを抽出・変換し、残り（処理不可能な部分）の処理はプロキシリレーション上で行う．メディエータは問合せの結果を受けると、その結果を該当するプロキシリレーションに挿入し、グリッドアプリケーションはそれをアクセスする．

プロキシデータベースを使用することで次のような利点が得られる：(1) グリッド上から様々な Web データベースと、グリッド内のデータベースを統一的に SQL でアクセスできる．(2) 検索インターフェースが処理不可能な問合せの部分はプロキシデータベースを管理する DBMS に委ねることができ、全体の問合せ処理の時間を短縮できる [12]．(3) プロキシデータベースはグリッド上にある任意の DBMS で実装できるので、OGSA-WebDB はプラットフォーム独立となる．

OGSA-WebDB はラッパからのソーススキーマはグリッド上の DBMS で materialize され、Web データベースからの検索結果をそのリレーションにロードされる．また、グリッドアプリケーションはそのスキーマに基づいて問合せをつくり、大部分の SQL 問合せはロードされたデータに対して処理される．このアプローチは Materialized アプローチに近いが、一部の SQL 問合せを Web データベースで処理されるのは Materialized アプローチと異なる．また、OGSA-WebDB は Virtual アプローチの問合せ処理も使用する．SQL 問合せから Web データベースが処理可能な条件を抽出し、それらの条件（サブ問合せ）を

(注2)：OGSA-DAI は RDBMS の他に XML データベースもサポートする．

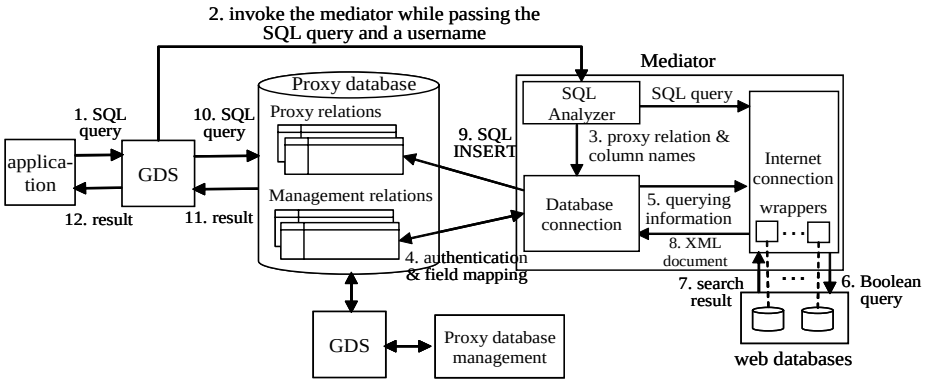

図 2　OGSA-WebDB のシステム構成と Web データベースアクセスの手順

該当する Web データベースに送る．しかし、OGSAWebDB は、スキーマの materialization の他に、大部分の SQL 問合せの要素／節（Join, Group-by, Order-by など）をメディエータに処理されず、グリッド上での DBMS に委ねることは Virtual アプローチと異なる．

グリッド認証基盤はプロキシデータベースの中にある**管理リレーション**で保持される．管理リレーションは Web データベースの認証情報を管理し、メディエータは認証を必要とする Web データベースをアクセスする時に、管理リレーションを参照する．

4.2　システム構成とデータベースアクセスの手順

図 2 は OGSA-WebDB のシステム構成及びデータベースアクセスの手順を示す．GDS は SQL 問合せを受けると（ステップ１）、その問合せと DBMS にログインするためのユーザ名を渡しながらメディエータを起動する（ステップ２）．

SQL 解析モジュールは渡された SQL 問合せとユーザ名を受け取り、問合せからプロキシリレーション名とカラム名を抽出し、それを**データベースコネクションモジュール**に渡す（ステップ３）．データベースコネクションモジュールはそれらの情報で Web データベースアクセスのための情報をプロキシデータベースの中にある管理リレーションから得る（ステップ４）．特に、データベースコネクションモジュールは管理リレーションに対して以下の処理を行う．

（1） 対象とする Web データベースを決定し、そのラッパクラス名を要求する．

（2） 対象とする Web データベースは認証を必要とするものであれば、認証情報を獲得する．

（3） SQL 問合せに記載されるカラム名を適当な名前にマッピングする．

以下は管理リレーション及びそのスキーマである．但し、カラム名の前に付けてある'#'は主キー、'*'は外部キーであることを示す．

- $User(\#uid, localName, description)$：DBMS にログインするための情報を管理する．$ID$ (UID) とローカルユーザ名 $(localName)$ を持つ．
- $Authentication(*uid, extName, passwd, *wid)$ ：Web データベースにログインするための認証情報を管理する．$extName$ と $passwd$ はそれぞれログイン名とパスワードである．wid と uid はそれぞれ Web データベースとローカルユーザ名の ID である．
- $Resource(\#wid, relName, url, wrapper, login)$ ：Web データベースに関する情報を管理する．$relName$ と url はそれぞれ Web データベースのプロキシリレーション名と URL であり、$wrapper$ は Web データベースを包むラッパクラス名、$login$ は認証が必要であるかどうかを示す'y'/'n'フラグである．
- $Fieldmapping(*wid, localColumn, extField)$：プロキシリレーションのカラム名を Web データベースのフィールド検索に使われるフィールド名にマッピングする情報を管理する．$localColumn$ と $extField$ はそれぞれそのカラム名とフィールド名である．

管理リレーションは**プロキシデータベース管理モジュール**によって管理される．例えば、新しい Web データベースを登録する場合、$Resource$ リレーションにラッパ名やプロキシリレーション名などを挿入する．また、その Web データベースがユーザ認証を必要とするものであれば、該当するユーザの ID とその認証データを $Authentication$ リレーションに挿入する．このモジュールはグリッドサービス（OGSA）に対応したクライアントであり、データベースとのやり取りをする時に GDS を使用する．

認証マッピングは OGSA-WebDB で重要な機能である．これは、ユーザが異なる Web データベースに対して異なるログイン名を持つ可能性があるが、OGSA-WebDB はグリッドアプリケーションなので、グリッドの認証との対応を取る必要があるからである．システムの実装には OGSA-DAI ソフトウェア [6] を使用する．OGSA-DAI はアプリケーションに与えられた X509 証明書を、接続されるデータベースシステムのログイン名とパスワードにマッピングする．OGSA-WebDB はそのユーザ名を受け取ってさらに Web データベースのログイン名とパスワードにマッピングし、全体として一貫性を実現する．

データベースコネクションモジュールは得られたマッピング結果などを**インターネットコネクションモジュール**に渡す（ステップ５）．このモジュールは対象となる Web データベースが処理可能な問合せ条件を SQL 問合せから抽出し、それをラッパに渡す．また、もしその Web データベースはユーザ認証を

必要とするものであれば、認証情報もラッパに渡す.

ラッパはWebデータベースの問合せ結果から必要とするデータを抽出するモジュールである．抽出されたデータをXMLに変換し、データベースコネクションモジュールに渡す（ステップ８）．データベースコネクションモジュールはその問合せ結果を該当するプロキシリレーションに挿入する（ステップ９）.

最後に、プロキシリレーション内にはWebデータベースのデータが挿入された後、GDSはSQL問合せをプロキシデータベースに送って処理して（ステップ１０）、問合せ結果をGDSに渡す（ステップ１１）.

4.3 問合せ処理

問合せ処理の主な目的は`WHERE`節の条件にマッチするWebデータベースとローカルリレーションからのデータを求めることである．Webデータベース検索をするために、`WHERE`節の条件をそのWebデータベースの処理可能な形に変換しなければならない．しかし、前に述べたように検索インタフェースの問合せ処理の能力は極めて限定されるものであるので、問合せ変換は簡単ではない．例えば、結合条件の処理や範囲問合せの処理などは多くの検索インタフェースでは処理できない.

本稿でのアプローチはSQL問合せを２つの段階に分けて処理する．最初のパス（第一パス）では、`WHERE`節の条件に含まれる全てのWebデータベースからのデータを該当するプロキシリレーションに挿入する．`WHERE`節に検索条件を持つWebデータベースの場合は、その条件をWebデータベースにサポートされるBoolean条件に変換して検索を行い、結果を該当するプロキシリレーションに挿入する．一方、検索条件を持たないWebデータベースの場合は、その条件をローカルリレーションのデータか検索条件を持つWebデータベースのデータで生成し、検索を行う．検索条件の生成方法は次の節で説明される.

次のパス（第二パス）では、最初のパスでまだ処理されなかった（無視された）条件や他のSQL問合せ要素／節を処理する．このパスはプロキシデータベースを管理するDBMSによって行われる.

4.3.1 第一パス問合せ実行

本稿では、次のような`WHERE`節を考える：`WHERE` JC `AND` C．JCは結合条件の連言 $jc_1 \wedge \ldots \wedge jc_l$（$l \geqq 0$）、また$C$は以下のような条件である.

（１） $c_1 \wedge \ldots \wedge c_n$ ($n \geqq 1$)．但し、c_iはSQL問合せ条件 $r.col\ op\ value$ である．$r.col$はリレーションrのカラム名、opはSQLオペレータである．また、rはプロキシリレーションかローカルリレーション(注3)である.

（２） rがプロキシリレーションで、かつrに対する条件が存在すれば、少なくとも1個の条件は次のような形を持たなければならない：$r.col$ LIKE %kw% あるいは $r.col = kw$．kwはキーワード、"LIKE“ はパターンマッチングオペレータ、また"%“記号は任意の文字にマッチするワイルドカードである．簡単のため、この条件を「**キーワードベース条件**」と呼ぶことにする.

キーワードベース条件はキーワードを用いた検索インタフェースのほとんどでサポートされている．したがって、この条件を満たす`WHERE`節の条件から、検索インタフェースがサポートするBoolean条件への変換は簡単である．ほとんどのWebデータベースは検索条件がないとアクセスできない．したがって、`WHERE`節条件の中に、Webデータベースに処理可能な条件（キーワードベース条件）が少なくとも１つ含まなければならない．このため、上の条件の制約２を設ける.

図3はこのパスのアルゴリズムの概要を示す．Rは`FROM`節に含まれるリレーションの集合である．メディエータは以下の問合せ関数とオペレータをサポートすると仮定する.

（１） **選択問合せ** $sq(r, \{c_1, \ldots, c_k\})$：$c_1 \wedge \ldots \wedge c_k$ の条件を満たすデータ項目をリレーションrから求める．rはプロキシリレーションであれば、与えられた条件で該当するWebデータベースに検索を行う．但し、この時キーワードベースでない条件が存在すれば、その条件は無視される.

（２） **半結合問合せ** $sjq(r, X, col)$：リレーションrからのカラムcolに該当するXのデータ項目でrを半結合する(注4)．r

```
METHOD: Main(R, JC, C)
1: C にある条件をリレーション名に基づいてグループ化する;
2: リレーション r_i を持つ各条件のグループ cg_i において、
3:    p_i ← sq(r_i, cg_i);
4:    r_i を R から削除する;
5: Exec(p_1, ..., p_k)、k は条件グループの数;

METHOD: Exec(p_1, ..., p_k)
6: paraOp(X_1 ⇐ p_1, ..., X_k ⇐ p_k);
7: IF X_i (1 ≦ i ≦ k) は既に実行完了であれば、
8:    r_i ← X_i のデータの持ち主;
9:    IF r_i は Web データベースであれば,
10:      r_i ← Web データベースのプロキシリレーション;
11:      insertOp(r_i, X_i);
12:   jcs ← r_i を含む JC にある結合条件;
13:   IF jcs は空であれば, return;
14:   jcs に含まれる全てのリレーションを R から削除する;
15:   jcs に含まれる全ての結合条件を JC から削除する;
16:   jcs にある結合条件を r' に基づいてグループ化する
        但し、r' は r_i と結合されるリレーション;
17:   r' にグループ化された各グループ {jc_1, ..., jc_m} において、
18:      sjq_1 ← sjq(r', X_i, col'_1), ..., sjq_m ← sjq(r', X_i, col'_m)、
           但し、col'_i は jc_i に含まれる r' のカラム;
19:      sjq_min ← 最小のコストを持つ半結合問合せ;
20:   Exec(sjq_min_1, ..., sjq_min_k)、k は条件グループの数；
```

図3　問合せ実行アルゴリズム

(注3)：ローカルリレーションとはプロキシ・管理リレーション以外にグリッド上で存在するリレーションを言う.

(注4)：実際に、半結合問合せは選択問合せの集合として実装されている．これは、まずカラムcolに該当するXから、異なる値の集合 $v = \{v_1, \ldots, v_k\}$ を抽出し、次に条件集合 $c = \{c_1, \ldots, c_k\}$（但し、$c_i =$ “$r.col = v_i$”、$v_i \in v$）を作成し、最後に各条件 $c_i \in c$ において、選択問合せ $sq(r, \{c_i\})$ を作る．選択問合せは3番目の並列オペレータで実行される.

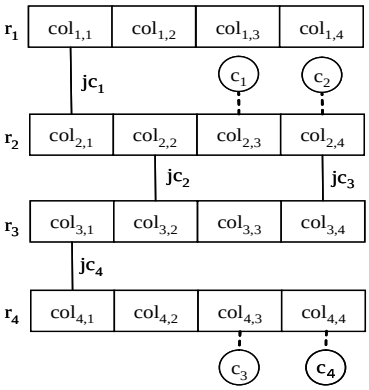

図 4　4つのリレーションを結合する条件の例

はプロキシリレーションであれば、データ項目を r の Web データベースから求める．

（3） **並列オペレータ** $paraOp(X_1 \Leftarrow q_1, \ldots, X_m \Leftarrow q_m)$：問合せ $q_1, \ldots, q_m$ を並列に実行し、それぞれの結果を $X_1, \ldots, X_m$ に入れる．

（4） **書き込みオペレータ** $insertOp(r, X)$：データ項目集合 X をリレーション r に挿入する．

まず、アルゴリズムは C の問合せ条件で選択問合せを作る (1–3 行目)．この選択問合せには、Web データベース検索のためのものも含まれている．そして、それらの問合せを渡しながら Exec 関数を呼び出す (5 行目)．

Exec 関数は与えられた問合せ $p_1, \ldots, p_k$ を並列に実行し、それぞれの問合せ結果を $X_1, \ldots, X_k$ に入れる（6 行目）．ある問合せの実行が完了するたびに、すぐに結果は次の半結合問合せを作るために使用される（7–19 行目）．半結合問合せは、C に問合せ条件を持たない Web データベース（プロキシリレーション）のために作られる．この「早いもの勝ち」(first-come-first-served）手続きは全体の応答時間を短縮でき、また検索条件を持たない Web データベースに対して、半結合問合せを作るための方法でもある．

各データ項目の集合 X_i において、X_i はある Web データベースからのデータであれば、該当するプロキシリレーションに挿入される（9–11 行目）．次に、Exec はリレーション r_i を R にある他のリレーションと結合するような全ての結合条件を JC から選択し、それらの結合条件を jcs に入れる（12 行目）．そして、jcs に含まれる結合条件とリレーションは以降のステップに処理されるので、それぞれは R と JC から削除される（14 と 15 行目）．

r_i は複数のリレーションと結合されることがあるので、jcs の結合条件はさらに r_i と結合されるリレーションに基づいてグループ化される（16 行目）．そして、Exec は各グループにおける各結合条件において、半結合問合せを作成し（17 と 18 行目)、最小の実行コストを持つ半結合問合せを選ぶ（19 行目）．

半結合問合せの実行コストはその問合せを実行するために必要とする時間の関数である．実行時間は結合されるカラムの中にある異なる値の数で推定される．

最後に、作成された半結合問合せは Exec 関数を再帰的に呼び出すことで並列に実行される（20 行目）．jcs が空であれば、アルゴリズムは停止する（13 行目）．

図 4 は4つのプロキシリレーション r_1, r_2, r_3, r_4 を結合する例である．結合条件 (jc) は二つのカラムを接続する線、あるカラムに対する条件はそのカラムに接続される円形に示される．c_1, c_2, c_3 はキーワードベース条件であるが、c_4 はそうではない例とする．

上のアルゴリズムを適応すると、まずアルゴリズムは二つの選択問合せ $p_1 = sq(r_2, c_1 \wedge c_2)$ と $p_2 = sq(r_4, c_3 \wedge c_4)$ を作る．これらの問合せは並行に Exec によって実行される：$paraOp(X_1 \Leftarrow p_1, X_2 \Leftarrow p_2)$．但し、$p_2$ を実行する時、c_4 はキーワードベース条件ではないので、無視される．仮に、p_1 の問合せの実行が先に完了したとすると、アルゴリズムは X_1 のデータ項目で検索条件を持たない Web データベース (r_1 と r_3 の Web データベース) のために半結合問合せを作る．まず、$jcs = \{jc_1, jc_2, jc_3\}$ を作成し、r_2 の結合相手のリレーションに基づいてグループ化する：$jcsG = \{\{jc_2, jc_3\}, \{jc_1\}\}$．$r_3$ の Web データベースの半結合問合せを最初の $jcsG$ の要素から作るが、どれの結合条件で作るのかを決めなければならない．ここで、jc_2 の結合条件から作る半結合問合せのコストが一番小さいとすると、アルゴリズムは問合せ $sjq(r_3, X_1, col_{3,2})$ を作る．同様に、$jcsG$ の二番目の要素から r_1 の Web データベースの半結合問合せを作る：$sjq(r_1, X_1, col_{1,1})$．最後に、これらの問合せを：$paraOp$ で並行に実行され、r_1 と r_3 のそれぞれの Web データベースが検索される．

4.3.2　第二パス問合せ実行

このパスはプロキシデータベースを管理する DBMS によって行われる．DBMS は SQL 問合せを GDS から受け取り、その問合せをローカルリレーションと既に Web データベースから第一パスの検索結果データが挿入されたプロキシリレーションに対して処理する．

このパスは `WHERE` 節の条件に基づいてカラム選択、射影、結合オペレーションを行って、第一パス結果に含まれる冗長なデータを除く．また、このパスは集約関数、`GROUP BY`、`ORDER BY` 節のような他の SQL ステートメント要素も実行する．ここで注意すべきは、第一パスで無視されたプロキシリレーションに対するキーワードベースでない条件は、このパスで実行される．

5.　システムの実装

OGSA-WebDB は Globus Toolkit 3 (GT3) [4] と OGSA-DAI 3.1 [6] の上に Java で実装されている．現在、本システムは科学や薬、特許、文献などの Web データベースへのアクセスが可能である．Web データベースの GDSF は Tomcat servlet コンテナー [1] に展開され、プロキシデータベースは MySQL DBMS [5] に管理される．

ラッパは Compaq の Web 言語 (WEBL) [8] と Republica の XFetch Wrapper [9] で実装された．図 5 にグリッドクライアントのスナップショットを示す．この例では、クライアントは2つの Web データベース（Drugs@FDA [3] と ChemFinder [2]）

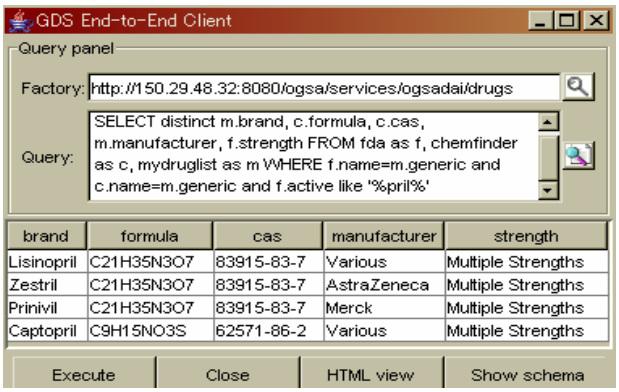

図 5　グリッドクライアント

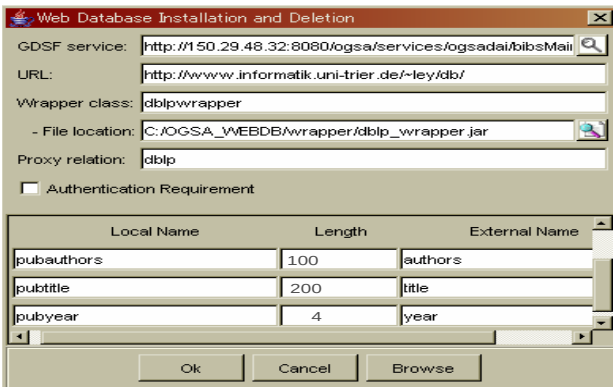

図 6　プロキシデータベース管理ツール

と 1 つのローカルリレーション mydruglist を結合する SQL 問合せを送る．問合せは活性成分を含む薬の名前とその量を Drugs@FDA から求め、それらの薬のブランド名と製造業者を mydruglist から、 化学式と CAS 番号を ChemFinder から求める．この時、活性成分の名前は “pril”の文字列を含むものとする．ChemFinder は認証を必要とする Web データベースであるが、OGSA-WebDB はその認証プロセスを自動的に処理する．

図 6 は新たな Web データベースとして DBLP を本システムに追加しているプロキシデータベース管理ツールのスナップショットである．プロキシリレーションの名前は dblp、ラッパの名前は dblpwrapper に設定している．また、DBLP のフィールド検索に使われるフィールドの名前 authors、title、year はそれぞれある適当なカラム名にマッピングされる．

6. システムの拡張：意味的キャッシングの利用

現在の OGSA-WebDB は、プロキシリレーションは GDS にアクセスされた後、プロキシリレーションのデータを直ぐに削除する．しかし、Web データベースからの結果データは次の問合せに答えるために使用することができる．これはシステムの応答時間と効率を一層向上できると思われる．

意味的キャッシング（semantic caching）[15], [23] は *data shipping technique* に基づくクライアント・サーバー DBMS の性能を向上するキャッシングのメカニズムである [24]．Data shipping では、問合せはクライアン側で処理されるために、データのコピーはサーバーからクライアントに転送される．意味的キャッシングはデータのコピーを保持すると共にそのデータの意味的記述（semantic description）を管理する．

意味的キャッシングのメカニズムは、メディエータと Web データベースをそれぞれクライアントとサーバーとして見なせば、OGSA-WebDB に適用できる．しかし、Semantic caching メカニズムはクライアントとサーバの問合せ処理能力が高いと仮定するので、OGSA-WebDB に適用するために工夫必要がある．次の節では OGSA-WebDB の意味的キャッシングについて説明する．

6.1 諸 定 義

［定義 1］　意味的リージョン（semantic region）はキーワードベース条件の連言である．また、意味的キャッシュ（semantic cache）は意味的リージョンの集合からなる．

［定義 2］　$s_1 = s_2$ である時、意味的リージョン s_1 と s_2 は等価（つまり $s_1 \equiv s_2$）であると言う．

［定義 3］　$\langle s_1(r_1)\rangle \supset \langle s_2(r_2)\rangle$ である時、リレーション r_1 の意味的リージョン s_1 はリレーション r_2 の意味的リージョン s_2 を包含する（つまり $s_1 \supset s_2$）と言う．但し、$s_i(r)$ は条件 s_i を r に適用することを意味し、$\langle s_i(r)\rangle$ は適用した結果のデータ項目の集合を意味する．

［定義 4］　$s_1 \not\supset s_2$、 $s_2 \not\supset s_1$ かつ $\langle s_1(r_1)\rangle \cap \langle s_2(r_2)\rangle \neq \emptyset$ である時、リレーション r_1 の意味的リージョン s_1 とリレーション r_2 の意味的リージョン s_2 は互いに重なり合う（つまり $s_1 \oslash s_2$）と言う．

［定義 5］　$\langle s_1(r_1)\rangle \cap \langle s_2(r_2)\rangle = \emptyset$ である時、リレーション r_1 の意味的リージョン s_1 とリレーション r_2 の意味的リージョン s_2 は互いに交わらない（つまり $s_1 \otimes s_2$）と言う．

各プロキシリレーション r は一つの意味的キャッシュを持ち、r のデータ項目は一つ以上の意味的リージョンに属することができる．また、意味的リージョンはキーワードベース条件の連言と定義される理由は、プロキシリレーションのデータは代理される Web データベースからキーワードベース条件の連言で処理求められた結果だからである．

6.2 意味的キャッシングの問合せ処理

r はプロキシリレーション、また S と w はそれぞれ r の意味的キャッシュと Web データベースとする．w に対する問合せ q が与えられる時に、q は次のように処理される(注5)．

1. IF $s_i \equiv q$ or $s_i \supset q$ となるような意味的リージョン $s_i \in S$ が存在すれば、$\langle q(r)\rangle$ を答として返す．
2. IF $s_i \oslash q$ $(1 \leqq i \leqq k)$ となるような意味的リージョンの集合、$S' = s_1, \ldots, s_k$ $(S' \subseteqq S)$ が存在すれば、
   2.1. IF w は OR と否定オペレータをサポートすれば、
      2.1.1. $res_1 \cup res_2$ を答として返す．但し、$res_1 = \langle q(r)\rangle$ かつ $res_2 = \langle (q \wedge \neg(s_1 \vee \ldots \vee s_k))(w)\rangle$ である．
      2.1.2. q と res_2 をそれぞれ S と r に挿入する．

(注5)：ここで、q はラッパ／ Web データベースに送られる問合せである．したがって、q はキーワードベース条件あるいはその連言である．

2.2. Otherwise $res = \langle q(webdb) \rangle$ を答として返し、q と $res - \{\langle s_1(r) \rangle \cup \ldots \cup \langle s_k(r) \rangle\}$ をそれぞれ S と r に挿入する．

3. Otherwise $res = \langle q(webdb) \rangle$ を答として返し、q と res をそれぞれ S と r に挿入する．

一番目の IF 条件は、q はキャッシュにあるデータで完全に処理可能な時のものである．この時に、q は S にある意味的リージョンに完全にカバーされるので、q は S に挿入されない．

二番目の IF 条件は、q はある意味的リージョンの集合と互いに重なり合う時のものである．この時に、もし Web データベース w が OR と否定オペレータをサポートすれば、意味的リージョンと重なっていない部分が w から求められて、r に挿入される．さもなければ、q はそのまま w に送られ、重なっていないデータ項目だけが r に挿入される．どちらのケースでも、q はキャッシュのデータで完全に処理できないので、q は S に挿入される．

最後の条件は、q と S は互いに交わらない時のものである．この時に、q はそのまま w に送られ、マッチしたデータ項目は全て r に挿入され、q は S に保存される．

本手法と既存の意味的キャッシングの方法との大きな違いは、本手法では q がキャッシュのデータで完全に処理できる時に q を S に挿入しないことと、S にはキーワードベース条件だけが含まれる（つまり範囲問合せなどが含まれない）ことである．これにより意味的キャッシュのサイズを大きく減らすことができ、条件の照合時間を短くできる．また、S はキーワードマッチングの条件しか含まないので、意味的キャッシュの実装は容易である(注6)．

7. まとめと今後の課題

インターネット上に存在する情報源の数は膨大で、かつ急速に増えつつある．グリッド仕様を満足する OGSA-WebDB の実現により、これらの情報源に対してグリッドアプリケーションからの SQL での統一的なアクセスを可能となった．現在は、システムの性能を向上すべく 6 節で述べた意味的キャッシュメカニズムを実装中である．

今後の課題としては、情報源の記述 (source description) の SQL での利用が考えられる。この記述は検索インターフェイスの問い合わせ能力に関するメタデータである．一般のパターンマッチング以外にも SQL のオペレータの処理をサポートする情報源に対しては、より多くの `WHERE` 節の条件を変換できるので、問い合わせの効率が向上できる．また、これから増加するであろう Web サービスによる情報源は、アクセスの仕様とそのメタデータ定義が与えられている．このメタデータを同様に活用することで、本システムのアーキテクチャはそのままに、Web サービスによる情報源へのより効果的なアクセスの実現が可能と考えられる．

(注6)：例えば、プロキシリレーション r と同じスキーマのリレーション r_c を作り、キーワードベース条件を挿入する時に、該当するキーワードを挿入するとキャッシュとして使える．

文　献

[1] Apache Tomcat. http://jakarta.apache.org/tomcat.
[2] ChemFinder. http://chemfinder.cambridgesoft.com.
[3] Drugs@FDA. http://www.accessdata.fda.gov.
[4] Globus Toolkit. http://www-unix.globus.org/toolkit.
[5] MySQL. http://www.mysql.com.
[6] Open Grid Services Architecture Data Access and Integration (OGSA-DAI). http://www.ogsa-dai.org.
[7] PubMed. http://www.ncbi.nlm.nih.gov/entrez.
[8] WEBL. http://research.compaq.com/SRC/WebL.
[9] Xfetch Wrapper. http://www.x-fetch.com/index.html.
[10] P. Buneman et al. A data transformation system for biological data sources. In *Proc. VLDB*, 1995.
[11] N. P. Chue et al. Grid data service specification. The GGF9, 2003.
[12] M. P. Said and I. Kojima. OGSA-WebDB: An OGSA-Based System for Bringing Web Databases into the Grid. In *Proc. IEEE Int. Conf. on Inf. Technology*, 2004. (to appear).
[13] N. P. Chue et al. Grid data service specification: The relational realisation. The GGF9, 2003.
[14] N. P. Chue et al. Grid data service specification: The XML realisation. GGF9, 2003.
[15] S. Dar, M. J. Franklin, B. T. Jónsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proc. VLDB*, pages 330–341, 1996.
[16] DOLPHIN. http://www.cp-dolphin.com.
[17] D. Florescu et al. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.
[18] I. Foster. The anatomy of the Grid: Enabling scalable virtual organizations. *Int. Journal of High Performance Computing Applications*, 15:200–222, 2001.
[19] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. OGSI-WG, GGF, June 2002.
[20] H. Garcia-Molina et al. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.
[21] M. Halevy and A. Tarczy-Hornoch. A model for data integration systems of biomedical data applied to online genetic databases. In *Proceedings of AMIA Annual Symposium*, pages 473–477, 2001.
[22] L. M. Hass, E. T. Lin, and M. A. Roth. Data integration through database federation. *IBM System Journal*, 41(4):578–596, 2002.
[23] A. M. Keller and J. Basu. A predicate-based caching scheme for client-server database architectures. *VLDB Journal: Very Large Data Bases*, 5(1):35–47, 1996.
[24] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Survey*, 32(4):422–469, 2000.
[25] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. VLDB*, pages 251–262, 1996.
[26] OGSA-DQP. http://www.ogsadai.org.uk/docs/dqp.
[27] D. Pearson. Grid database requirements. The GGF4, 2002.
[28] J. Smith et al. Distributed query processing on the grid. The GGF5, 2002.
[29] G. Wiederhold. Mediators in the architecture of future information systems. In *Readings in Agents*, pages 185–196. 1997.
[30] L. Wong. Kleisli, a functional query system. *Journal of Functional Programming*, 10(1):19–56, 2000.
[31] G. Zhou, R. Hull, and R. King. Generating data integration mediators that use materialization. *Journal of Intelligent Information Systems*, 6(2/3):199–221, 1996.
[32] Y. Zhuge, H. García-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *Proc. ACM SIGMOD*, pages 316–327, 1995.