2005－DBS－137（Ⅱ）（58）
2005／7／14

複数 RDB 上の分散データの XML ビューによる統合
－データグリッドエンジン「FADGE（フェッジ）」－

金政 泰彦† 門岡 良昌† 石原 康秀‡ 瓦井 健二‡ 宮澤 君夫†

†株式会社富士通研究所 〒211-8588 神奈川県川崎市中原区上小田中 4-1-1
‡富士通株式会社 〒211-8588 神奈川県川崎市中原区上小田中 4-1-1
E-mail: †kanemasa@jp.fujitsu.com, kadooka@labs.fujitsu.com, kmiyazawa@fujitsu.com,
‡{ishi, kawarai}@ssd.ssg.fujitsu.com

あらまし 近年、企業合併・吸収や企業内再編などビジネス変化が活発であることから、複数拠点の DB に分散したデータを統合して利用したいというニーズが高まっている。しかも、ビジネス変化の周期が短いことから、統合形態を柔軟に素早く変更できることが求められている。データグリッドはそのようなニーズに答える技術として注目されている。しかし、従来技術はデータアクセス手段の統合に留まり、データビューに関しては統合できていなかった。本論文では、複数 RDB 上の分散データをデータグリッド技術で統合する場合において、仮想的な XML 形式のデータビューを提供し、それに対して XQuery による問い合わせを実現することによって、データビューを統合する手法について紹介する。

キーワード データグリッド，DB 統合，ビュー，XML，XQuery

Integration of Distributed Data on Multiple RDBs Using XML View
－Data Grid Engine “FADGE”－

Yasuhiko KANEMASA† Yoshimasa KADOOKA†

Yasuhide ISHIHARA‡ Kenji KAWARAI‡ and Kimio MIYAZAWA‡

†Fujitsu Laboratories Ltd. 4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8588, Japan
‡Fujitsu Limited 4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8588, Japan
E-mail: †kanemasa@jp.fujitsu.com, kadooka@labs.fujitsu.com, kmiyazawa@fujitsu.com,
‡{ishi, kawarai}@ssd.ssg.fujitsu.com

Abstract Recently, businesses change rapidly and frequently in the world, so it is desired to make data distributed among DBs on multiple sites be available together. And further, because businesses change in short term, a flexible and speedy way to integrate data is desired. “Data Grid” is well-known as one of the solutions deal with such requests. However, traditional ways can handle only how to access distributed data but can’t handle how to give a integrated view of the data. In this paper, we mention how to give a integrated data view providing a XML-style logical view and a query interface using XQuery.

Keyword Data Grid, DB Integration, View, XML, XQuery

1. はじめに

企業データ／科学データが年々増加していく中で、複数の RDB に分散しているデータを統合して利用したいというニーズが高まっている。特に、ビジネス分野では企業合併・吸収や企業内再編成などビジネス変化が活発であることから、企業内データの利用方法が頻繁に変わり、情報系システムの変更が頻繁に求められている。しかし、ビジネスの変更周期が短い為に、多大な費用と開発時間を掛けてシステムを再構築しても、すぐにまた再変更の必要が生じ、コスト的にも時間的にもビジネスの変化に追従できな

いという事態が発生している。

そのような状況の中で、近年、グリッドの一形態であるデータグリッドが注目を集めている。データグリッドとは、分散した異種・異環境のデータリソースを仮想的に統合して、あたかも 1 つのデータリソースであるかのように扱うことを可能とする技術である。データを物理的に分散したまま仮想的に統合することによって、データを物理的に 1 箇所に集めるよりも迅速に統合検索システムを構築することができるし、統合方法の頻繁な変更に対しても柔軟に対応することが出来る。

我々は、複数 RDB 上の分散データをデータグリッド技術で仮想的に統合する場合の問題点について研究している。そして、RDB 統合を迅速・柔軟に行うためのデータグリッドエンジンとして「FADGE」（Fujitsu Advanced Data Grid Engine）を開発した。以下では、「FADGE」で実現したデータビューの統合機能について紹介する。まず最初に従来のデータグリッドの取り組みとその問題点について節2で説明した後に、節3で我々の開発した「FADGE」について紹介し、節4でその主要技術について詳細を説明する。

2. データグリッドによる RDB 統合

2.1. 従来技術とその問題点

データグリッドによる RDB 統合の一番基本的な形は図 1のようになる。この方式では、データグリッドは異種 RDB への統一されたアクセス手段を提供し、RDB 製品毎の差異やアカウント管理体系の違いを吸収してくれる。このような機能を提供するソフトウェアとして Globus Toolkit[1]＋OGSA-DAI[2]がある。Globus Toolkit（以下、Globus）は基本的なグリッド機能を提供するツール群で、グリッドの標準仕様の参照実装となっている。また。OGSA-DAI は Globus と一緒に使用するデータグリッドのツール群で、データグリッドの標準仕様の参照実装となっている。この方式では、上位アプリの開発者は各 RDB に対して統一された方法で SQL を発行することが出来るが、どのデータがどの RDB に格納されているか、また各データの格納構造はどうなっているか等を意識して問い合わせを発行する必要があり、開発作業は決して容易にはならない。

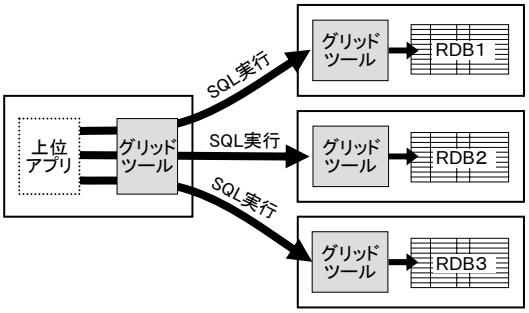

図 1：グリッドによるアクセス手段の統一

そこで、図 2のように上位アプリとの間に分散クエリーエンジンを配置し、分散クエリーシステムを構築する方式がある[3][4]。これらの分散クエリーシステムでは、上位アプリの開発者は単一のクエリーによって複数 RDB に分散したデータを一括して取得することができる。しかし、この方式では見えているビューはリレーショナルモデルそのものなので、上位アプリの開発者はデータの分散を意識し、それらの結合を考慮して問い合わせを発行しなければならない。この問題は単一の RDB でも生じる問題であるが、開発者も開発時期も異なる複数 RDB 上のデータを結合する場合は、単一 RDB の場合よりもその困難さが増すのは明らかである。ゆえに、この方式では上位アプリの開発作業は必ずしも容易にはならない。

2.2. 我々のアプローチ

我々は、複数 RDB を結合した結果としてデータをリレーショナルモデルそのままで見せるのではなく、XML 形式のビューでアプリ開発者に見せる方式を提案する。複数 RDB に分散しているデータを１つの巨大な XML データというビューで見せ、それに対して XQuery で問い合わせを受け付け、問い合わせを受けた時点で問い合わせ結果の生成に必要なデータとその取得方法を解析し、各 RDB から SQL 問い合わせでデータを取得し、その取得データを XML 形式に合成して問い合わせ元に返す。XML 形式のビューは、予め作成する DB 統合用メタデータで任意に定義可能である。

こうすることにより、上位アプリの開発者は複数 RDB にデータが分散していることを一切意識することなく、単に XML データに対して XQuery を発行するかのような手軽さでアプリ開発が行えるようになる。

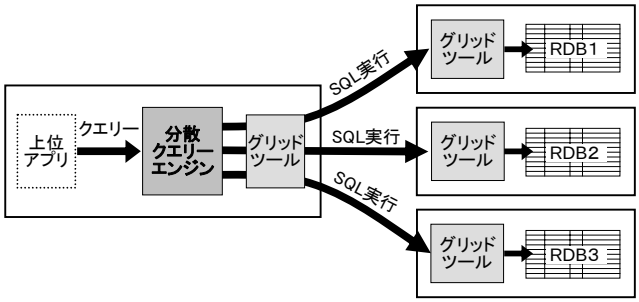

図 2：分散クエリーシステム

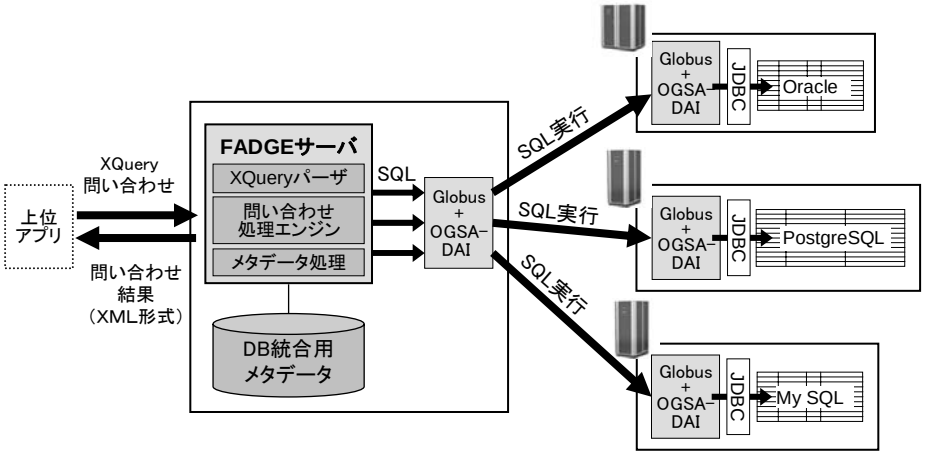

図 3：FADGE の全体構成

2.3. XML ビューに関する従来研究との関係

RDB に格納されているデータを XML 形式で取り出す研究は既に色々と行われている。代表的なものに XPERANTO[6]（XML for Tables[7]）や SilkRoute[8]等がある。しかし、これらの研究はデータが単一の RDB に格納されていることを前提としており、データグリッド環境のような複数の RDB 上に分散していることは想定していない。我々の研究は複数 RDB 上に分散しているデータを統合することが主目的であり、その際に各 RDB から如何にしてデータを取得して XML ビューを実現するかが研究課題である。

また、XML ビューのアクセス権制御の問題に付いても取り扱う。単独の RDB から XML ビューを作成する場合のアクセス権制御の研究は既に行われている[9]が、本研究では管理体系が異なる複数 RDB のデータから XML ビューを作成する場合のアクセス権制御の問題について言及する。

3. データグリッドエンジン「FADGE」

本節では、我々の開発したデータグリッドエンジン「FADGE」について紹介する。

3.1. システム構成

FADGE システムの全体構成を図 3に示す。FADGE システムでは各 RDB へのアクセスに前述の Globus＋OGSA-DAI を利用している。これらのグリッドツールを用いることによって、統一された I/F での SQL 問い合わせを実現している。その上位に、XML 形式のビューを生成する FADGE サーバを配置している。FADGE サーバは、XQuery の問い合わせを受け付けると、XML ビューの生成ルールである DB 統合用メタデータの中から該当するものを参照し、その定義内容と XQuery の問い合わせ内容を解釈して問い合わせプランを作成する。その問い合わせプランに基づいてグリッドツール経由で各 RDB へ SQL を送信し、その結果を受け取るとそれから問い合わせ結果となる XML データを合成し、それを問い合わせ元へ返す。

FADGE は問い合わせを処理する為に最低限必要な中間データを保持するだけで、データをキャッシュしたりはしない。各 RDB へなるべく検索条件の多く含まれた SQL を発行して、問い合わせ処理を可能な限り RDBMS に任せる戦略である。ただし、データは複数の RDB に分散しているので、それらの間に跨る最適化や JOIN 処理は FADGE が行う必要がある。

3.2. サンプル RDB データ

以下の説明に使用するサンプル RDB データを図 4に示す。このサンプルでは、受注処理に関連するデータが 4 台のサーバの 4 つの RDB に分散して格納されている。サーバ A の orderDB には、注文伝票の情報が正規化されて 2 つのテーブルに格納されている。ここでは注文商品は商品コードで記録されており、商品名は記録されていない。サーバ B の itemDB には商品コードと商品名の対応表が格納されている。サーバ C の stockDB1 とサーバ D の stockDB2 には商品の在庫数の情報が格納されており、どちらの DB に格納されるかは商品に応じてどちらか一方に決まっている。以下では、この 4 つの RDB に分散したデータを統合する例を考える。

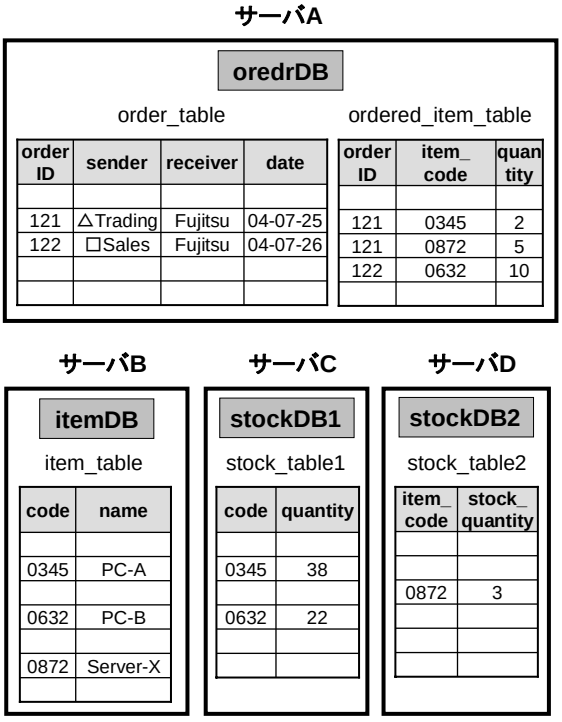

図 4：統合する RDB データの例

3.3. XML 形式のデータビュー

図 4のサンプル RDB データに対して生成される XML ビューの例を図 5に示す。注文伝票リストという 1 つの大きな XML の中に、エレメントとして各注文の情報が繰り返されるというビューである。在庫の情報は、各注文の中の注文商品の情報の中に current_stock として埋め込まれて出現している。

```
<?xml version="1.0" encoding="EUC-JP" ?>
<order-list>
     :
  <order>
     :
  </order>
  <order>
    <order_number>121</order_number>
    <sender>△Trading</sender>
    <receiver>Fujitsu</receiver>
    <date>04-07-25</date>
    <oreder_items>
      <item>
       <code>0345</code>
       <item_name>PC-A</item_name>
       <quantity>2</quantity>
       <current_stock>38</current_stock>
      </item>
      <item>
        <code>0872</code>
        <item_name>Server-X</item_name>
        <quantity>5</quantity>
        <current_stock>3</current_stock>
      </item>
    </order_items>
  </order>
  <order>
    <order_number>122</order_number>
    <sender>□Sales</sender>
    <receiver>Fujitsu</receiver>
    <date>04-07-26</date>
    <order_items>
      <item>
       <code>0632</code>
       <item_name>PC-B</item_name>
       <quantity>10</quantity>
       <current_stock>22</current_stock>
      </item>
    </order_items>
  </order>
  <order>
     :
  </order>
  <order>
     :
  </order>
     :
</order-list>
```

図 5：XML ビューの例

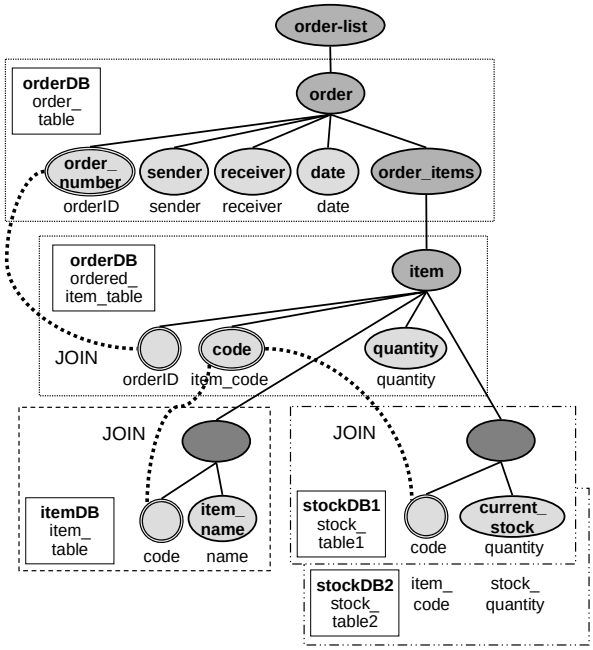

図 6：FADGE のビュー定義

この XML ビューはあくまでも一例に過ぎず、FADGE の生成する XML ビューはビューの生成ルールである DB 統合用メタデータで自由に定義できる。FADGE のビュー定義は、[6]や[8]のようにビュー自体を XQuery で定義するのではなく、「ビューの XML スキーマ」「各 XML エレメントに対応する RDB カラムの指定」「RDB テーブル間の JOIN 要素の指定」の３つの情報を XML 形式で定義する方式である（図 6参照）。

4. 分散 RDB データ上での XML ビューの実現

本節では、FADGE において分散 RDB データに対してどのようにして XML ビューを実現しているのかを説明する。

4.1. XQuery 問い合わせ処理の基本戦略

複数 RDB に跨る XML ビューに対する問い合わせ処理で難しいのは、問い合わせが RDB 間に跨るので問い合わせ処理の最適化を RDBMS に全て委ねるという方法が取れないことである。また、問い合わせ順序を失敗するとネットワーク経由で多量の中間データが移動することになり、失敗ペナルティが非常に大きいということもある。問い合わせ順序を失敗しなくても、中間データの取り扱いコストが非常に大きいので、そのことを考慮して問い合わせ順序を考える必要がある。

そこで、FADGE では次のような基本戦略で XQuery 問い合わせ処理を実現している。

①XML ビューの生成ルール(DB 統合用メタデータ)や XQuery 内の条件式から、１つの RDB に関するデータ絞込み条件を可能な限り集め、絞り込み条件の多く含まれた SQL を生成し、後はそれを RDB へ投げて RDBMS の処理能力と最適化に任せる。これを実施するためには、XQuery 内の条件式を如何にして単一の SQL に纏めるかがポイントとなる。

②発生する中間データの量がなるべく少なくなるような問い合わせ開始 RDB を選択する。

③問い合わせ回数がなるべく少なくなるように、XQuery 中の問い合わせ条件の一部については条件判定を最終結果生成時に後回しにする。

4.2. XQuery 問い合わせの実行手順

問い合わせ処理の詳細を具体例を使って説明する。図 5の XML ビューに対して、次のようなサンプル XQuery を実行する場合を考える。

```
<result> {
    for     $i in
            fn:doc("order-list.xml")//order
    let     $j := $i/order_items
    let     $k1 := $j/item
    let     $k2 := $j/item
    where   $k1/item_name = "PC-A"
      and   $k1/quantity = 2
      and   $k2/item_name = "Server-X"
      and   $k2/quantity >= 3
      and   $i/date <=
            xs:date("2004-07-26")
    return  $i
} </result>
```

問い合わせ処理は以下の手順で行われる。

1. XQuery の where 節内の and 結合された各条件式を、対応する RDB 毎に纏める。サンプルでは次のようになる。
   - orderDB：
     $k1/quantity = 2
     $k2/quantity >= 3
     $i/date <= xs:date("2004-07-26")
   - itemDB：
     $k1/item_name = "PC-A"
     $k2/item_name = "Server-X"

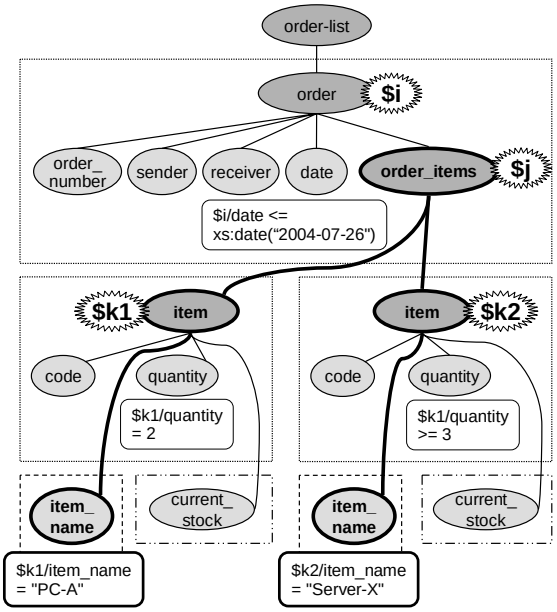

図 7：条件式グループの判定

2. RDB 毎に纏めた条件式群内の任意の２つの条件式について、条件付けされている XML エレメント同士を結ぶパスが同一 RDB の範囲内に全て納まっているか判定する。このときに、パス中で使われている変数が異なると別個のインスタンスになるので、そのことを考慮してパスを考える必要がある。パスが全て同一 RDB に収まっていれば、両条件式は同一の SQL で使用することが出来るので一つのグループに纏めておく。これをここでは「条件式グループ」と呼ぶ。サンプルでは、条件式グループは次のような３つになる。itemDB に対する二つの条件式は、変数が異なることによって別々の条件式グループになる（図 7参照）。
   - 条件式グループ１（oredrDB）
     $k1/quantity = 2
     $k2/quantity >= 3
     $i/date <= xs:date("2004-07-26")
   - 条件式グループ２（itemDB）
     $k1/item_name = "PC-A"
   - 条件式グループ３（itemDB）
     $k2/item_name = "Server-X"
3. 各条件式グループに関して、それぞれの絞込み能力のスコアを計算する。スコアは、含まれている条件式の数と、各条件式の絞込み能力から算出される。絞り込み能力は、比較対象が文字列の場合は数字よりも高く、比較記号が等号の方が不等号よりも高いとしている。スコアの値と算出式はヒューリスティックに決めている。サンプルの場合は条件式グループ２と３が一番高いスコアになる。
4. 一番スコアの高かった条件式グループの RDB に対して、一番最初に問い合わせを実行する。条件式グループ内の全条件式を使用して、XML ツリー上で上位に隣接する RDB との JOIN 項目を求める SQL を作成し、それをグリッドツール経由で RDB へ送り実行する。サンプルの場合は次のような SQL が itemDB で実行される。

```
SELECT   DISTINCT t0.code
FROM     item_table t0
WHERE    t0.name = 'PC-A'
```

5. 一つの RDB で問い合わせが完了すると、次にその結果を使用して XML ツリー上で上位に隣接する RDB で問い合わせを行う。この際に、変数によるインスタンスの違いを考慮した上で、使用できる条件式グループがあれば前回の結果と一緒に SQL の問い合わせ条件に加える。サンプルの場合は以下のような SQL が orderDB で実行される。

```
SELECT   DISTINCT t0.orderID,
                  t0.sender,
                  t0.receiver,
                  t0.date
FROM     order_table t0,
         ordered_item_table t1,
         ordered_item_table t2
WHERE    t1.item_code = '0345'
  AND    t1.quantity = 2
  AND    t2.quantity >= 3
  AND    t1.orderID = t0.orderID
  AND    t2.orderID = t0.orderID
```

6. 手順5を繰り返して XML ツリー上で一番上（サンプルの場合は order エレメント）まで求め終わったら、そこから今度は XML ツリーを下位に向けて辿る為の SQL を順次発行し、取りこぼしなくサブツリー全体を取得する。
7. RDB から取得した全データから、XQuery 問い合わせの結果となる XML データを合成する。このとき、RDB に問い合わせした際に反映出来ていない条件式が残っている可能性があるので、変数間の関係を考慮しながら再度条件判定を行う。サンプルでは、条件式グループ３の条件式が反映できずに最後まで残っている。
8. XML データを問い合わせ元へ返して、XQuery 問い合わせ完了。

なお、手順5では XML ツリーを単純に上に向かって辿っているが、これでは問い合わせの内容によっては中間データが最少にならない。サンプルの場合でも、条件式グループ３の条件式を使って itemDB に問い合わせを行い、その結果を加えて orderDB に問い合わせを行った方が、その後の問い合わせで無駄な中間データを生成せずに済む。しかし、そうすることには二つのデメリットがある。一つは問い合わせ回数が増えてレスポンスが遅くなること、もう一つはその問い合わせでは絞り込み条件が少なくなるので結果データの量が莫大になる可能性があるという点である。この問題に関しては効果的な最適化手法を現在検討中である。

4.3. アクセス権制御

RDB 統合におけるアクセス権制御を考える上で難しいのが、それぞれの RDB の管理体系が異なることである。各 RDB でユーザアカウントは異なるし、アクセス権制御のポリシーも異なる。また、非技術的な問題も発生する。統合される各 RDB の管理者の立場からすると、アクセス権は RDB 管理者が責任を持って管理すべきことで、アクセス権を他人が設定することなどは到底受け入れ難い。

そこで FADGE では、RDB アクセス時のアクセス権制御は RDB 側に任せて、FADGE のユーザグループと各 RDB のユーザアカウントとのマッピングを取ることによってアクセス権制御を実現している。マッピングは二段階で行い、一段階目で FADGE ユーザグループから Globus ユーザ（globus のユーザ証明書）へのマッピングを行い、二段階目で Globus ユーザから RDB アカウントへのマッピングを行う。一段階目は FADGE の機能であるが、二段階目は Globus＋OGSA-DAI の機能である。グリッドツールの機能をそのまま利用することによって、セキュアな RDB アクセスを実現している。また、一段階目のマッピングは FADGE の管理者が設定するが、二段階目のマッピングは RDB 管理者が設定することになるので、データ公開の最終的な権限を RDB 管理者が握ることになり、RDB 管理者の安心感を得られやすい。

XML ビュー（を定義している DB 統合用メタデータ）は FADGE の特定のユーザグループと対応付けて管理されており、XML ビューを参照できる（XQuery 問い合わせを実行できる）のはそのユーザグループだけに限られている。

5. おわりに

本論文では、データグリッドによる RDB 統合で問題となるデータビューの問題を取り上げ、それに対する一つの解として、複数 RDB に分散したデータに対して XML ビューを提供し、XQuery での問い合わせを実現する手法について紹介した。単一の RDB に格納されているデータに対して XML ビューを提供する研究は広く行われているが、本論文では複数 RDB に分散したデータから XML ビューを作成する場合の問題点を挙げ、それを如何に解決しているのかを詳細に説明した。我々は、この XML ビュー機構をデータグリッドエンジン「FADGE」として既に実装しており、現在は評価とさらなる改良を行っている。

今後の課題としては、データが複数 RDB に分散していることによって発生する可能性がある、ネットワーク経由での多量の中間データの移動に対する対策が必要と考えている。現在、問い合わせ計画のより高度な最適化と、適当なポリシーによる問い合わせ打ち切りという二通りの方向で検討を進めている。

文　献

[1] Globus Toolkit,http://www.globus.org/toolkit/

[2] OGSA-DAI, http://www.ogsadai.org/releases/ogsadai.php

[3] IBM, WebSphere Information Integrator, http://www-6.ibm.com/jp/software/data/ii/

[4] OGSA-DQP, http://www.ogsadai.org/dqp/

[5] XQuery, http://www.w3.org/XML/Query

[6] M. J. Carey, J. Kiernan, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian, “XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents,” Proc. 26th International Conf. on VLDB, pp.646-648, Cairo, Egypt, Sept. 2000.

[7] IBM, XML for Tables, http://www.alphaworks.ibm.com/tech/xtable

[8] M. F. Fernandez, Y. Kadiyska, D. Suciu, A. Morishima, and W. C. Tan, “SilkRoute: A framework for publishing relational data in XML,” TODS, Vol.27, no.4, pp.438-493, Dec. 2002.

[9] 品川徳秀, 北川博之, “セキュリティを考慮した RDB の XML ビュー機構の検討,” 電子情報通信学会第 16 回データ工学ワークショップ論文集(DEWS2005), 5A-o4, Feb. 2005.