社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－DBS－144 (10)
2008－GN－ 66 (10)
2008／1／24

簡易型分散データベース処理システムの試験的開発

山本 泰之 †， 藤廣 健一 †， 畑中 雅彦 †， 清水目 和幸 ††

† 室蘭工業大学 †† 北海道 NS ソリューションズ株式会社

分散データベースシステム (分散 DB) を使用するユーザが、単一データベースシステム (単一 DB) と同じように利用できるようにするため、ユーザと分散 DB との処理を中継するシステム (中継サーバ) の開発を行った。本システムでは、ユーザからの SQL を分散 DB 向け SQL へと変換している。また、中継サーバとユーザ・分散 DB との通信内容を標準 SQL(SQL99) に限定していることにより、各分散 DB に使用される DBMS に依存することなく動作する。

本システムでは、機能を主な検索操作のみに限り、また用途を長期トレンド解析など長時間を要する処理に限ることで、単一 DB と同じか、あるいは数倍の処理性能を発揮することが確認できた。

キーワード：DBMS に依存しない、QueryBroker、クエリ変換アルゴリズム

A Simplified Distributed DataBase Processing System (α-version)

Yasuyuki YAMAMOTO † Kenichi FUJIHIRO † Masahiko HATANAKA † Kazuyuki SHIMIZUME ††

† Muroran Institute of Technology

†† Hokkaido NS Solutions Corporation

We have developed a "query-relay" server for Distributed DataBase (DDB). This server acts like that Single DataBase (SDB), because the server is located midway between user and DDB hosts, and will transform SQL from query for SDB to queries for DDB. At this time, the server uses standard SQL (SQL99), therefore it independent of DBMS on DDB hosts.

This system confines purposes to principal select function and heavily-loaded processing like prolonged trend analysis, thereby processing speed is fast from equal to severalfold more than SDB.

Keyword : independent of DBMS, QueryBroker, query transform algorithm

1 はじめに

ある特定のテーマに沿った大量のデータを効率的に管理・運用するためにデータベースシステム (以下、DBS) が用いられるが、今日では特に、負荷分散・リスク分散などの観点から分散データベースシステム (以下、分散 DB) を用いることが多くなってきている[1]。

ユーザが分散 DB を利用する際、特に重要なのは、データの分散を意識せずに使用することができる点である。一般に既存の DBMS は分散 DB 機能を備え、データの分散を意識せずに分散 DB を利用できるが[2]、製品固有の方法を用いて実装されているため、全ての DB ホストで同一の DBMS を用いらなければならない。また、商用の大規模 DBMS 製品を用いて構築した分散 DB 環境では、固有のハードウェアを用いる場合が多いため柔軟に拡張を行えず、一般的に高価である。

本研究室では、多くの DBMS 製品が抱える分散 DB 環境構築時の柔軟性欠如の問題点を解決するために、DB ホストに用いられる DBMS に依存することなく動作し、かつ安価に構成できる分散 DB 処理システムの開発を行っている[3][4]。

本論文では、処理できる機能を限定した簡易版として構築した、分散 DB 処理システムの試作版について報告する。

2 システム導入環境とシステム要件

時々刻々と変動する株価や為替レートなど大量で途切れることのないストリームデータを扱う株式市場や金融市場用の DBS は、大規模・高性能・高信頼性が要求される。また一般的に、この種の DBS は、データの確実な保存と複数のユーザから

の短期トレンド解析などをリアルタイムで処理することが優先される[5]。

本システム開発は、上述したメインのシステム(以下、メインシステム)が存在することを前提に、これと並列し用途を限った安価で簡易型のサブシステムをターゲットとした。このターゲットに対する要求定義として、

(A) 想定する主要な処理要求は、長期トレンド解析など長時間を要すること、
(B) システムコスト低減のため、汎用の構成要素を多く用いること、
(C) ネットワーク上の情報資源の活用を考慮すること、
(D) 分散DBホストに使用されるDBMSに依存されないこと、

などを採用した。本システムの導入を想定した環境の概念図をFig.1に示す。大量のデータを参照し、

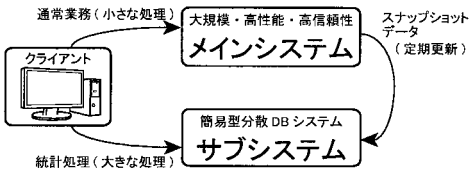

Fig. 1　システム導入環境の概念図

長時間の処理を必要とする長期トレンド解析などをサブシステム側で行うことで、即時応答性が要求されるメインシステムの負荷を低減させることができる。また、サブシステム側でメインシステムとは独立してデータを保持するので、データ消失のリスクを低減させることもできる。サブシステム側のデータはメインシステムからのスナップショットデータを定期的に取得することを想定している。

3　簡易型分散データベース処理システム

本システムの概念図を Fig.2 に示す。本システムは、クライアント・クエリ中継サーバ(QueryBroker)・分散DBの3層構造から成る。QueryBrokerはユーザが単一DBと同じように分散DBを利用できるように、ユーザからの単一DB向けのSQLを分散DB向けのSQLへ変換したり、分散DBからの結果をユーザ向けの結果へ変換するなどの処理を行う。

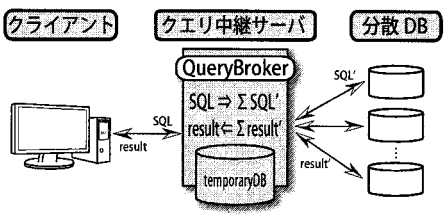

Fig. 2　簡易型分散DB処理システムの概念図

3.1　分散データベース部

分散DB環境を構築するとき、データの配置方法は、完全重複型, 部分重複型, 完全重複型の3つに分けられる。

完全重複型は、全てのDBホストで同一の内容を保持し、各クライアントからのトランザクションを各DBホストで並行に処理する形態をとる(一般には、レプリケーションデータベースシステムと呼ばれる)。一般的に、全てのクライアントに同一のコンテンツを提供するwebシステムで多く用いられる。完全重複型の概念図をFig.3に示す。

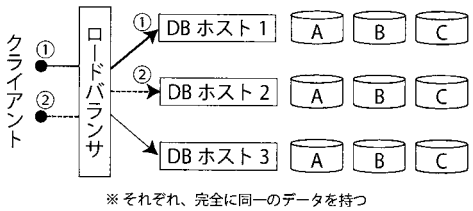

Fig. 3　完全重複型分散DBの概念図

部分重複型は、一部のデータについては複数のDBホストで重複して保持するが、あるデータについては1箇所のDBホストにしか保持していない形態をとる。大企業など多くの地域に拠点を持つ場合、各支店で独自にDBSを構築し、それらを統合し企業全体で分散DBを構築するなどが例として挙げられる。自拠点からは参照があまり無いデータについては、他拠点のデータ利用することでデータ容量を小さくすることが可能となる。部分重複型の概念図をFig.4に示す。　完全分割型は、データを単純に分割して重複無く各DBホストに分散配置する形態をとる。主にDBホスト間の負荷分散を目的としたシステム構成で、分割した全てのデータに対して均一な負荷が発生する処理に有効である。完全分割型の概念図をFig.5に示す。

本システムは、多くのデータを参照する長期トレンド解析を主な用途としていることや、DBホ

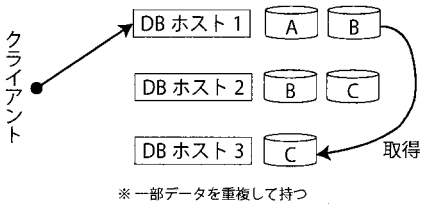

Fig. 4　部分重複型分散 DB の概念図

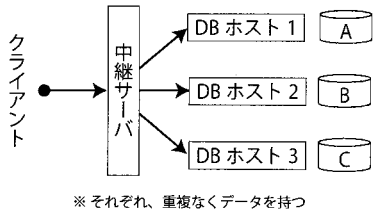

Fig. 5　完全分割型分散 DB の概念図

ストに一般ユーザが使用する PC や数世代前の PC など汎用の構成要素を用いるので、各ホスト間で負荷が分散される完全分割型でシステムを構築した。また完全分割型の分散 DB は、構成が単純であるため障害復旧に対する労力が少なくてすむというメリットもある。

分散 DB 層は、オープンソースを主体とした DBMS が動作している複数台の PC から成り、異なる DBMS が混在した環境にも対応できる。また DBMS に依存しないため、既に稼動しているシステムを接続してシステムを構築することができる。このように基本的に分散 DB に使用されるマシンには特に手を入れることなく、システムを構築できる。

3.2　クエリ中継サーバ部とクライアント部

QueryBroker はサーバ機能を提供する本体と再集約用の一時 DB(temporaryDB) から構成される[6]。QueryBroker はクライアントから単一 DB 向けのクエリ (SQL) を送信される。クエリを受け取った QueryBroker はクエリ変換アルゴリズムを用いて分散 DB 用のクエリ (SQL ') に変換し、全ての分散 DB ホストに送信、結果 (result') を取得する。さらに、取得した各 result' を temporaryDB に統合・格納し、再集約処理によりデータベース全体の結果 (result) を取得し、クライアントに返信して処理が終了する。

クライアント-QueryBroker 間、QueryBroker-分散 DB 間の処理要求などの通信は、DB で標準的に使用される SQL(SQL99)[7] によって行われている。このため、各分散 DB には現在普及している様々な DBMS を使用することができる。例えば、商用の DBMS である Oracle やオープンソースソフトウェアの PostgreSQL、MySQL などの製品を混在して使用することができる。

クライアントと QueryBroker 間では、SQL 通信の他に QueryBroker に使用しているマシン名や待ち受けポート番号など特別な情報のやり取りを行っているため、クライアントは専用のものを使用する必要がある。

3.3　システムの動作

開発した QueryBroker の機能構成を Fig.6 に示す。図中の①から④は、次節で説明するクエリ変

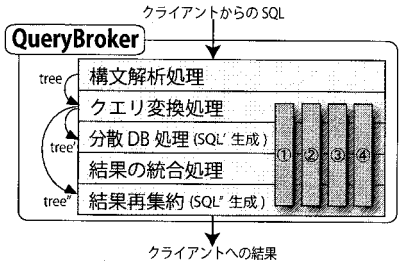

Fig. 6　QueryBroker の機能構成図

換アルゴリズムが扱うクエリの種類を示す[8]。

① 変換の必要のない SQL 文
② データ取得命令を変換する SQL 文
③ 必要データを追加取得する SQL 文
④ その他の SQL 文

QueryBroker は、構文解析処理・クエリ変換処理・分散 DB 処理・結果の統合処理・結果再集約処理の 5 つの段階で動作する。

- **構文解析処理**

  構文解析処理では、クライアントから送信された SQL 文の内容を解析するため、字句解析処理・構文解析処理を通じて構文の木構造表現である構文木 (tree) を生成する。

  字句解析処理とは、SQL 文全体を単語 (トークン) 毎に分解し、それぞれのトークンがどういう種類の単語であるかを決定する処理である。

  構文解析処理とは、字句解析で得られたトークン列の順番を見て、どのようにトークンが連続しているかを判断する処理である。本システムの構文解析処理では、判断の結果とし

て構文に応じた構文木オブジェクト (tree) を生成する。

- **クエリ変換処理**
  クエリ変換処理では、構文解析処理で生成された tree を受け取り、それをもとに分散 DB 用クエリの構文木 (tree') と結果再集約用の構文木 (tree") を生成する。tree'、tree " の生成にはクエリ変換アルゴリズムを用いる。
- **分散 DB 処理**
  分散 DB 処理では、クエリ変換処理で生成された tree' から分散 DB 用の SQL 文 (SQL') を生成して各分散 DB に投入する。このとき、分散 DB 台数分のスレッドを生成し、分散 DB 処理を並列動作させている。各スレッドが分散 DB に接続し、SQL 文を送信、結果 (result') の取得を行っている。
- **結果の統合処理**
  分散 DB 処理から渡された result' は途中結果であり、データベース全体の結果、つまりクライアントへの最終結果を得るためには再集約する必要がある。再集約処理は QueryBroker が持つ temporaryDB 上で実行される。そのため、再集約処理に先だって temporaryDB に result' の内容を格納する必要がある。
  結果の統合処理では、同時に複数のクライアントからの処理を実行できるようにするため、クライアントごとに一意の名前をもつ一時テーブルを作成する。その後、result' の内容を 1 行ずつ分解する。次に temporaryDB へ格納するための INSERT 文を構成し、temporaryDB へ投入することで一時テーブルへデータを格納していく。
- **結果再集約処理**
  結果再集約処理では、クエリ変換処理で生成された再集約用クエリの構文木 tree " から再集約用の SQL 文 (SQL ") を生成し、temporaryDB へ投入する。そして、結果 (result) を取得し、クライアントへ送信する。

4 クエリ変換アルゴリズム

完全分割型分散 DB では、クライアントが入力する単一 DB 向けのクエリ (SQL) をそのまま分散 DB に投入しても正しい結果が得られない。正しく結果を得られるようにするには、各分散 DB から取得対象のデータに加えて冗長なデータを取得・補完し、それらのデータを統合した後に再度集約を行う必要がある。

本システムでは、クライアントから入力される SQL 文を自動的に各処理で利用する SQL 文に変換するしくみを確立した。これをクエリ変換アルゴリズムと呼ぶ。クエリ変換アルゴリズムは、クライアントから入力されたクエリの種類の応じて異なる方法で変換を行う。以降で、この種類ごとにクエリ変換処理の動作を説明する。

4.1 変換の必要のない SQL 文 (Fig.6 ①)

クライアントからのクエリの種類によっては、変換の必要の無いものもある。これらに該当するのは、条件に合致する SQL 文、総和を求める SQL 文、最大値・最小値を求める SQL 文がある。

- **入力クエリ (単一 DB 向け SQL)**
  SELECT target FROM Table WHERE attribute = 20;
- **分散 DB 用クエリ**
  SELECT target FROM Table WHERE attribute = 20;
- **再集約用クエリ**
  SELECT target FROM Table WHERE attribute = 20;

4.2 データ取得命令を変換する SQL 文 (Fig.6 ②)

上記以外の SQL 文では、入力クエリをそのまま分散 DB や再集約処理に用いても正しい結果を得られない。データ件数を取得する SQL 文と平均値を求める SQL 文で正しい結果を取得するためには、データ取得命令である集約関数を変更する必要がある。

4.2.1 データ件数を求める SQL 文

データ件数を求める SQL 文は、SELECT 句に集約関数 COUNT を指定したクエリである。この SQL 文は各分散 DB で取得したデータ件数の合計を求めることで、DB 全体のデータ件数を求めることができる。

- **入力クエリ (単一 DB 用クエリ)**
  SELECT COUNT(target) FROM Table;
- **分散 DB 用クエリ**
  SELECT COUNT(target) FROM Table;
- **再集約用クエリ**
  SELECT SUM(COUNT) FROM TempTable;

4.2.2 平均値を求める SQL 文

平均値を求める SQL 文は、SELECT 句に集約関数 AVG を指定したクエリである。各分散 DB に格納されている該当データの件数が異なる場合、入

力クエリをそのまま分散DBへ投入しても正しい結果を得られない。そこで、平均値は総和/件数で求められるため、平均値を求めるプロセスを総和と求めるプロセスと件数を求めるプロセスとに分解する。

- **入力クエリ (単一DB用クエリ)**
  SELECT AVG(target) FROM Table;
- **分散DB用クエリ**
  SELECT SUM(target), COUNT(target) FROM Table;
- **再集約用クエリ**
  SELECT SUM(SUM)/SUM(COUNT) FROM TempTable;

4.3 必要データを追加取得するSQL文 (Fig.6 ③)

グループに対して条件を適用するSQL文や整列を行うSQL文では、条件適用・整列を再集約処理で行う。入力クエリをそのまま分散DBに投入した場合、条件適用・整列に必要な情報が再集約処理のtemporaryDBに取り込まれないため、処理が行えない問題が発生する。この問題を解決するため、本アルゴリズムでは、条件適用・整列に必要なデータを追加取得するようにSQL文を変換する。

4.3.1 グループに対して条件を適用するSQL文

グループに対して条件を適用するSQL文は、GROUP BY句とHAVING句を指定したクエリである。分散DB環境では、グループに該当するデータが分散DB間に跨って配置されている可能性があるので、再集約処理時にグループ化する必要がある。このとき、条件に使用するデータとグループ化対象のデータが異なるならば、条件適用に使用するデータも追加取得する必要がある。

- **入力クエリ (単一DB用クエリ)**
  SELECT target FROM Table GROUP BY target HAVING SUM(attribute) < 20;
- **分散DB用クエリ**
  SELECT target, SUM(attribute) FROM table GROUP BY target;
- **再集約用クエリ**
  SELECT target FROM TempTable GROUP BY target HAVING SUM(attribute) < 20;

4.3.2 整列を行うSQL文

整列を行うSQL文は、ORDER BY句とHAVIGN句を指定したクエリである。整列には全てのデータが揃っている必要があるため、整列処理は再集約処理時に行う必要がある。このとき、結果として取得するデータとソートキーのデータが異なる場合がある。この場合は、各分散DBからソートキーとなるデータを追加取得する必要がある。

- **入力クエリ (単一DB用クエリ)**
  SELECT target FROM Table ORDER BY attribute;
- **分散DB用クエリ**
  SELECT target , attribute FROM table;
- **再集約用クエリ**
  SELECT target FROM TempTable ORDER BY attribute;

4.4 その他のSQL文 (Fig.6 ④)

上記のSQL文の分類は、プリミティブな基本機能を変換方法の違いにより分類したものであるが実際には、それらの機能が複合的に指定され、より複雑な処理を実行する場合が多い。その際にも、本アルゴリズムによって順にSQL文を変換することで、分散DBから正しい結果を得ることができる。

また、上記以外にもSQL文には、さらに複雑な処理を実現する機能が実装されているが、本アルゴリズムには実装されいない。実装されていないSQL文で主要な処理には、join処理やメインシステムからスナップショットデータを受け取り分散DBに挿入する処理などがある。これらに対処するため、joinに関しては、独立性の高いオプショナル・モジュール[9]として、データ挿入に関しては、データ検索とは全く別の処理方法を用いる特別な処理機構[10]として実装する試みを行っている。

5 動作実験

クエリ変換アルゴリズムを用いることによって、データが分散配置されている分散DB環境から正しく結果が得られるかを検証するため、及び、構築したシステムが単一DBと比較してどの程度の処理性能を発揮するかを検証するために簡易型分散DB処理システムの動作実験を行った。実験は、本研究で考案したクエリ変換アルゴリズムを検証する予備実験と、システムの処理速度を検証する本実験を実施した。

5.1 実験環境

実験には小規模のLAN環境を利用した。Fig.7に実験に使用したネットワーク環境の構成図を示す。簡易型分散DB処理システムはGigabit Ethernetで接続されている。また、結果送信時間短縮のため、クライアントはQueryBrokerと同一マシンを利用した。

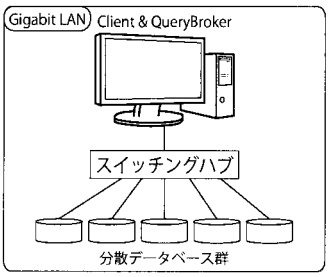

Fig. 7　実験環境のネットワーク図

実験に使用したマシンの諸性能を Table 1 に示す。QueryBroker には処理負荷が集中するため、分散 DB と比較して能力の高いマシンを利用した。また、OS には UNIX ライクでフリーライセンスの FreeBSD を使用した。

Table 1　実験に使用したマシンのスペック

マシン	CPU	メモリ
QueryBroker	Xeon Dual/3.4GHz	4.5GB
分散 DB1	Celeron/800MHz	384MB
分散 DB2	Celeron/800MHz	384MB
分散 DB3	Celeron/800MHz	384MB
分散 DB4	Pentium-Ⅲ/700MHz	384MB
分散 DB5	Celeron/800MHz	384MB

予備実験には、1,478,576 件の南極気象データを使用した。また、本実験には、156,410,393 件で予備実験に使用したデータとは別の南極気象データを使用した。

5.2　予備実験:クエリ変換アルゴリズムの動作検証

クエリ変換アルゴリズムが正しく動作することを確認するため予備実験を行った。実験に使用するデータを単一 DB として用意した環境と、分散 DB として用意した環境に登録し、それぞれの処理結果の比較を行った。

実験には 3 つの SQL 文を使用した。

(1) SELECT MIN(target) FROM Table;
(2) SELECT target FROM Table GROUP BY target HAVING AVG(attribute) < -24.0;
(3) SELECT target FROM table WHERE target is NOT NULL ORDER BY target;

3 つの SQL 文のうち、(1) の SQL 文は変換の必要のない SQL 文である。(2) の SQL 文はデータ取得命令を変換する SQL 文とデータを追加取得する SQL 文が含まれている。(3) の SQL 文にはデータを追加取得する SQL 文と変換の必要がない SQL 文が含まれる。

実験の結果、全てのクエリで結果が一致したため、クエリ変換アルゴリズムが正しく動作していることが確認できた。

5.3　実験:単一 DB との処理性能比較の検証

構築したシステムの性能を評価するために、単一 DB 環境との処理性能比較を行った。

QueryBroker には、単一 DB との性能評価を比較するためにスタンドアロンモードを用意している。スタンドアロンモードは、QueryBroker が接続する DB のうち 1 台のみを使用し、クライアントからの SQL 文を直接 DB へ投入するモードである。このスタンドアロンモードで実行した処理時間と開発した簡易型分散 DB 処理システムの処理時間を比較することで性能評価を行う。

実験には 2 つの SQL 文を使用した。

(1) SELECT MIN(target) FROM Table;
(2) SELECT AVG(target) FROM Table;

処理時間の比較を Fig.8、Fig.9 に示す。

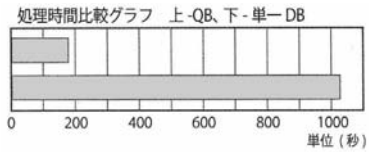

Fig. 8　実験 (1) の処理時間比較

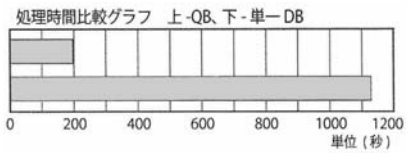

Fig. 9　実験 (2) の処理時間比較

6　まとめと考察

本システムでは、分散 DB から得られた結果を一度 temporaryDB へ格納する必要がある。この処理は、格納する件数が多い場合、膨大な時間がかかることが分かっている。しかし、このような処理の場合、ユーザが受け取るデータ件数も膨大なものとなる場合が多く、DBS の利用例としては、あまり現実的ではない。

本システムでは、実験の結果からtemporaryDBに格納される件数が少ない処理の場合((1)は5件、(2)も5件)は、単一DBと比べて優れた処理性能を発揮することが確認できた。これは、単一DBでは単独で行う検索処理などが、分散DBの場合は各DBホストで分割して行われるため、処理時間が短縮できるが大きいからである。またQueryBrokerのみ、処理性能が高いマシンを利用しているが、これにより集約処理を短時間で行うことが可能となり、全体の処理性能向上にも効果が大きい。

また予備実験から、ユーザがデータの分散を意識することなく、単一DBと同じSQL文をQueryBrokerに送信することで、正しい結果を得るシステムの構築ができたことも確認できた。

参考文献

1) M.Tamer Özsu, Patrick Valduriez: *Principales of Distributed DataBase Systems (2nd ed.)*, Prentice Hall, pp.1-24 (1986).

2) C.J.Date (藤原 譲 監訳): **データベースシステム概論** *(原書第6版)*, 丸善, pp.646 (1997).

3) 畑中 雅彦, 藤廣 健一, 岡田 あゆこ, 生田 正弘: **簡易型分散データベース処理システムの開発** *(1)* **-システムの概念設計-**, 開発技術研究会 平成17年度研究発表会, pp.9-10 (2005).

4) 山本 泰之, 藤廣 健一, 畑中 雅彦, 岡田 あゆこ, 生田 正弘: **簡易型分散データベース処理システム**, 電気・情報関係学会北海道支部連合大会, pp.148 (2006).

5) R.Kimball (藤本 康秀 監訳): **データウェアハウス・ツールキット**, 日経BP社, pp.27-130 (1998).

6) 藤廣 健一, 畑中 雅彦, 岡田 あゆこ, 生田 正弘: **簡易型分散データベース処理システムの開発** *(2)* **-システム実装と性能評価-**, 開発技術研究会 平成17年度研究発表会, pp.11-12 (2005).

7) Kevin Kline, Daniel Kline (石井 達夫, 宮原 徹 監訳): *SQL* **クイックリファレンス**, オライリー・ジャパン, pp.1-10 (2002).

8) 藤廣 健一, 畑中 雅彦, 生田 正弘: **分散データベースを対象とした** *SQL* **文の変換について**, 開発技術研究会 平成16年度研究発表会, pp.17-18 (2004).

9) 岡田 あゆこ, 生田 正弘, 藤廣 健一, 畑中 雅彦: **簡易型分散データベース処理システムの開発** *(3)* **-オプショナル・モジュールの開発-**, 開発技術研究会 平成17年度研究発表会, pp.13-14 (2005).

10) 山本 泰之, 畑中 雅彦, 清水目 和幸: **分散データベース処理システム** *QueryBroker* **のデータ挿入機能の実装**, 開発技術研究会 平成19年度研究発表会, pp.16-17 (2007).