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

2003－DBS－131 (16)
2003／7／16

並列分散 DBMS の開発
－プロトタイプのスケーラビリティ評価－

谷越 桂太† 宇山 公隆† 藤塚 勤也† 雲田 厚司† 近藤公久†

あらまし 大規模クラスタ環境においてスケーラブルな性能を有する並列分散 DBMS の実現を目指して開発を行っている。今回は、独立した DBMS が稼動する複数のノード(S-Node)を統括するノード(C-Node)を構築することで、シングルシステムイメージの並列分散 DBMS を実現するプロトタイプを開発した。本プロトタイプでは、配置関数によって各ノードにデータを分散配置する方式とすべてのノードにデータのコピーを冗長配置する方式をテーブル毎に設定可能とした。各テーブルのデータ配置方式をグローバルスキーマとして C-Node で管理することにより、C-Node は要求されたクエリの処理対象となるテーブルの配置方式に応じて各ノードに適正に処理を分配することが可能である。性能評価の結果、ノード数に応じたスケーラビリティが確認され、並列分散化の有意な効果が認められた。

キーワード データベース、並列分散処理、負荷分散、スケーラビリティ、性能評価

Development of a Parallel and Distributed DBMS
－An Evaluation of the Scalability of a Prototype System－

Keita TANIKOSHI† Kimitaka UYAMA† Kinya FUJIZUKA†
Atushi KUMOTA† and Tadahisa KONDO†

Abstract The main purpose of this study is to develop the parallel and distributed DBMS which exerts a scalable performance on a large scale cluster system. This paper introduces a prototype system of the DBMS which consists of one coordinator node (C-Node) and several subordinated nodes (S-Nodes). The prototype provides two data distribution methods: table data distributed by hash function, table data copied to all S-Nodes. The C-Node delivers queries from clients to appropriate S-Nodes by referencing the global schema which shows the target table's distribution method in the queries. Results from several performance evaluation tests on the prototype system showed scalable performance

Keyword database, parallel and distributed processing, load balancing, scalability, performance evaluation

1. はじめに

IT 化の進む現在、大規模なデータを安全かつ高信頼に保管・管理し、高速に処理する DBMS の役割が重要になっている。そして、データの大規模化は今後益々進むと予想される。また、大量データを高速に処理するだけでなく、サービスを継続的に提供する高可用性(high availability)も DBMS の必須の要件となっており、究極的には 24 時間 365 日連続サービスが求められることも少なくない。これまで、このような要件を満たすシステムはメインフレーム上で専用に設計されたシステムやハイエンドサーバと厳密に設計された周辺ハード構成上で稼動する DBMS によってのみ提供されてきた。

† 株式会社 NTT データ
NTT DATA Corporation

近年、PC やネットワーク機器の性能の向上とともに、PC をネットワーク接続して、高性能、高可用性を低価格で実現する PC クラスタの技術が盛んに研究されている。PC クラスタ技術は、科学計算分野では既にスーパーコンピュータ並みの性能を持つものも現れ、冗長化による高可用化と合わせて実用レベルになってきている。そこで、我々は大規模クラスタ環境で稼動する並列分散 DBMS を開発することとした。大規模クラスタ環境でスケーラブルな性能を発揮することで大量データを高速処理する機能およびデータの大規模化に柔軟に対応可能である。また、冗長化構成により高可用なシステムを実現することが可能である。

本報告では、小規模な Linux PC クラスタ上で

稼動する並列分散 DBMS プロトタイプを開発して、性能評価を行った結果を示す。

2. 並列分散 DBMS 設計

2.1. システム構成

我々が目指す並列分散 DBMS(以下 PDMS: Parallel Data Management System)の構成を図1に示す。PDMS はユーザからのクエリを受け付ける C-Node と、データを保持する S-Node から構成される。C-Node と S-Node によって並列処理と冗長処理を行うことで、高性能と高可用性を確保する。

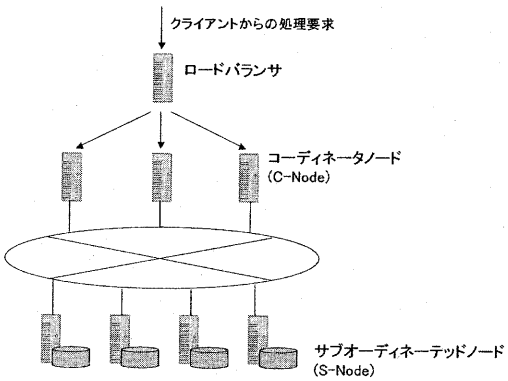


図1．システム構成概要

2.2. データの配置方式

PDMS では、並列分散処理によって性能の向上を図る。従って、データは異なるノードに分散されて配置される。しかし、分散配置されたデータ間の演算が必要な場合にはノード間でデータを交換する必要が生じるため、ネットワークトラフィックの増加がボトルネックとなることが懸念される[1]。また、複数のノードにデータが冗長化されて配置された場合には、更新処理における冗長データ間の整合性を保証する必要が生じるため処理が複雑になる。このような問題を軽減するため、S-Node へのデータの配置方式に以下の3方式を用意し、データベース設計時にデータの性質に応じてテーブル毎に選択することを可能とする[2]。

1．*Partitioned Table*
この方式のテーブル内のデータは配置関数によって複数の S-Node に分散配置される。配置関数を置き換えることでハッシュ分割、範囲分割などを実現できる。データを分散配置することにより、並列分散処理が可能となるため、大規模なテーブルに有効である。しかし、配置関数の引数に用いるキー属性（配置属性）以外をキーとするクエリの処理は全 S-Node を対象とした処理が必要になるため、通信コストがかかるという問題がある。

2．*Specific Node Table*
この方式のテーブルは分割されずに特定の S-Node に配置される。更新頻度が高く、比較的小規模なテーブルに有効である。複数のノードに配置して冗長化することも可能である。

3．*All Node Table*
この方式のテーブルは分割されずにすべての S-Node に配置される。*Specific Node Table* の冗長度を最大にしたものであり、すべてのノードにおいて通信を介さずにこのテーブルのすべてのデータにアクセスできることが保証される。このため、参照頻度は高いが更新頻度は低く、比較的小規模なテーブルに有効である。

2.3. クエリの分配分式

データを分散配置した場合、C-Node はクライアントからのクエリを適切にデータの配置に合わせて S-Node に分配する必要がある。クエリの分配においては、SQL から論理クエリツリー、物理クエリツリーへと変換される過程[3]に従い3通りの方式が考えられ、それぞれの方式によって並列実行ツリーを作成するレベルと S-Node への処理命令方式が異なる[4]。

表1．クエリ分配方式

1	SQLレベルでの分配	入力されたSQLを構文解析し、SQL構文木を作成する段階でクエリの分配を行う。
2	論理クエリツリーレベルでの分配	SQL構文木を意味解釈し、論理演算子によって表現する論理クエリツリーを作成する段階でクエリの分配を行う。
3	物理クエリツリーレベルでの分配	論理クエリツリーの格納されているデータの状況に合わせた演算の実行アルゴリズム、実行順序、データアクセス方式などを指定する物理クエリツリーを作成する段階でクエリの分配を行う。

表中の方式1から方式3になるにつれて並列実行ツリーをよりきめ細かく設定することが可能であるが、実際の開発は容易でなくなる。実現機能と実装量のトレードオフからどの方式を適用するかを適切に選択する必要がある。

2.4. プロトタイプ開発

並列分散処理による性能のスケーラビリティ評価を行う目的で、プロトタイプを開発した。本プロトタイプは前節までに示した PDMS の一部のみを実現したものである。

2.4.1 システム仕様と処理方式

単一の C-Node と最大 4 台の S-Node で構成される本プロトタイプは、ユーザからは S-Node の存在を意識する必要はなく、C-Node が単一の DBMS のように見えるシングルシステムイメージを実現している。

本プロトタイプのデータ配置方式は上述した *Partitioned Table* と *All Node Table* の 2 つのみを実装した。また、*Partitioned Table* の配置に用いる配置関数はハッシュ関数 1 種類のみを用意した。

C-Node は S-Node に分散配置されたテーブルの属性、分割方式などを”グローバルスキーマ”として保持する。ユーザからのクエリ (SQL) を受け取った C-Node はこのグローバルスキーマを参照してクエリを上述の方式 1 で S-Node に SQL の形で送信する。

S-Node では各々オープンソースソフトウェア (OSS) である PostgreSQL バージョン 7.2.1 が稼動している。また、ノード内に存在するテーブルの属性情報をローカルスキーマとして持つ。S-Node は C-Node から割り振られた SQL を受け付け、ローカルスキーマを参照してクエリ処理を実行し、結果を C-Node に返す。ローカルスキーマは PostgreSQL の管理情報=システムカタログである。本プロトタイプでは、ローカルスキーマに合わせてグローバルスキーマを手動で設定した。

参照系クエリの場合、C-Node は各 S-Node の実行結果を取得・集約する。

2.4.2 機能仕様と制約

SQL レベルの分配方式の場合、C-Node ではクエリの構文解析のみが行われ、任意のクエリに含まれるテーブルやその結合の種類、結合属性などは判断できない。このため一部のクエリに対しては適切な機能制約を設ける必要がある。以下、本プロトタイプにおける機能仕様と制約を示す。

1．共通項目
- *All Node Table* の更新は *2 phase commit* (2PC) などの全 S-Node が一貫した動作を保証する機構を設ける必要があるため許可しない。
- 副問い合わせ文はすべて許可しない。
- 操作対象となるテーブルが *Partitioned Table* であり、かつクエリのキー属性が配置属性である場合には、グローバルスキーマを参照して対象データを持つ S-Node にだけ実行命令 (SQL 文) を送信する。それ以外はすべての S-Node に実行命令 (SQL 文) を送信する。

2．SELECT 文
- FROM 句に *Partitioned Table* が複数ある場合、S-Node 間でのデータ交換が必須となるため、FROM 句に含まれる *Partitioned Table* は高々1 つとする。すなわち、他のテーブルはすべての S-Node に存在するものとする。
- FROM 句に含まれるテーブルが *All Node Table* のみの場合、任意の S-Node に SQL を送信する。

3．UPDATE 文
- *Partitioned Table* の配置属性を対象とした更新は許可しない。これはデータの再配置を伴うためである。

4．ユーザインタフェース
- 簡略化したコマンドライン入力方式のみで、C ライブラリ、ODBC、JDBC などは実装しない。

3. PostgreSQL の単体性能評価

3.1. 概要

PostgreSQL は OSS の中でも豊富な機能と実績を持った DBMS である。プロトタイプの性能評価を行う前に、まず様々なマシン上で稼動する PostgreSQL 単体の性能評価を行った。

試験は単一クエリのレスポンス性能評価 (Wisconsin benchmark ベース) とトランザクションスループット性能評価 (TPC-B Benchmark ベース) の 2 種類であった。測定に用いたマシンの仕様を表 2 に示す。なお、本評価試験においては一部 PostgreSQL バージョン 7.2.3 を用いているが、バージョン 7.2.1 との違いは小規模のバグフィクス程度であるので、性能に大きな違いは無いと考えられる。

表 2．PostgreSQL 単体評価試験マシン仕様

	HW1	HW2	HW3	HW4
OS	RHL AS 2.1	RHL AS 2.1	RHL 7.3	RHL 7.3
CPU	Xeon 2.4GHz × 2	PentiumⅢ 1.4GHz × 2	Pentium Ⅲ 1.26GHz	Pentium Ⅲ 500MHz
メモリ	SDR SDRAM 4GB	SDR-SDRAM 1.5GB	256 MB	512 MB
HDD	SCSI 15,000 rpm	IDE 5,400 rpm	IDE 7,200 rpm	IDE 5,400 rpm
RAID	RAID 1	なし	なし	なし

RHL AS 2.1: RedHat Linux Advanced Server 2.1
RHL 7.3 : RedHat Linux 7.3

表３．テーブルを構成する属性

属性	値域	順序
unique1	0-999999	ランダム
unique2	0-999999	昇順
two	0-1	ランダム
four	0-3	ランダム
ten	0-9	ランダム
twenty	0-19	ランダム
onePercent	0-99	ランダム
tenPercent	0-9	ランダム
twentyPercent	0-4	ランダム
fiftyPercent	0-1	ランダム
unique3	0-999999	ランダム
evenOnePercent	0,2,4,・・・,1	ランダム
oddOnePercent	1,3,5,・・・,1	ランダム
stringu1		ランダム
stringu2		昇順
string4		循環的

表４．クエリリスト

No	SQL
query101	SELECT * FROM tenk1 WHERE unique2 BETWEEN 0 AND 99
query102	SELECT * FROM tenk1 WHERE unique2 BETWEEN 0 AND 999
query107	SELECT * FROM tenk1 WHERE unique2 = 2001
queyr109	SELECT * FROM tenk1, tenk2 WHERE (tenk1.unique2 = tenk2.unique2) AND (tenk2.unique2 < 1000)
query110	SELECT * FROM tenk1, bprime WHERE (tenk1.unique2 = bprime.unique2)
query111	SELECT * FROM onek, tenk1 WHERE (onek.unique2 = tenk1.unique2) AND (tenk1.unique2 = tenk2.unique2) AND (tenk1.unique2 < 1000)
query118	SELECT DISTINCT two, four, ten, twenty, onePercent, string4 FROM tenk1
query120	SELECT MIN(tenk1.unique2) FROM tenk1
query121	SELECT MIN (tenk1.unique2) FROM tenk1 GROUP BY tenk1.onePercent
query126	INSERT INTO TENK1 VALUES (全カラムを指定)
query127	DELETE FROM tenk1 WHERE unique2 = 10001
query128	UPDATE tenk1 SET unique2 = 10001 WHERE unique2 = 1491
query132	UPDATE tenk1 SET unique1 = 10001 WHERE unique2 = 1491

表５．異なるHW環境における
PostgreSQL レスポンス性能

	No Index				B-Tree Index			
query	HW1	HW2	HW3	HW4	HW1	HW2	HW3	HW4
101	3748	4043	33299	26390	1133	1638	1483	4255
102	13612	15220	42659	62022	11119	13091	13967	42851
107	2546	2631	32303	24727	1	0	14	2
109	47438	43054	286465	624045	21891	26472	38578	82762
110	45408	40854	237700	500363	45244	40408	230007	397093
111	63121	76846	367061	787370	28074	34020	37885	106326
118	47321	52362	98793	276013	47762	52382	99648	290688
120	2828	3167	32300	22050	2813	3181	32386	23385
121	17094	18341	66801	165181	17026	18084	65795	159498
126	1	3	1	7	1	1	1	7
127	2548	2642	32467	22388	1	4	1	7
128	2550	2638	32507	22327	8	17	14	8
132	2547	2641	32678	23730	7	16	11	1

単位:[msec]

3.2. レスポンス性能評価

Wisconsin Benchmark はシングルユーザ環境のもとで人工的なデータに対する単一のクエリの実行時間を測定するベンチマークである[5]。測定に用いたテーブルを構成する属性及びクエリリストを表３、４に、測定結果を表５に示す。なお、テーブルのタプル数は 100 万、クラスタインデクスとして B-Tree インデクスを”unique2”属性に付与した。また、Wisconsin Benchmark は共有バッファサイズを指定するが、今回はすべての環境で同一サイズ=３２[Mbyte]としている。

表５の結果から、全般的には搭載メモリ容量、CPU クロック数が大きいほどレスポンス性能が高い傾向が見られる。これは、大規模な結合演算を伴うため、計算性能に依存した性能が見られたものと考えられる。

また、HW３と HW４の比較においては、CPU クロックは HW３が、搭載メモリ容量は HW４がそれぞれ高性能なものとなっているため、CPU 性能への依存度が高い処理の場合には HW３が、メモリへの依存度の高い処理の場合は HW４の性能が高くなったものと考えられる。

3.3. スループット性能評価

TPC-B は銀行業務をモデルとしたトランザクション性能を測定するベンチマークである[5]。なお、PostgreSQL は追記型の DBMS であり、更新クエリに伴いデータ内に不要領域が増加し、性能が劣化する。これを防ぐため、一定時間=５[min]ごとに VACUUM コマンドにより不要領域を削除した。測定に用いたテーブル構成を図２、トランザクションの内容を表６、測定結果を図３に示す。

図３から、スループット性能は、メモリ容量・CPU クロック数に必ずしも依存しないことがわかる。特に、HW１と HW２の比較から、マルチ CPU による優位性は全くないと言える。

図２．スループット試験テーブル構成

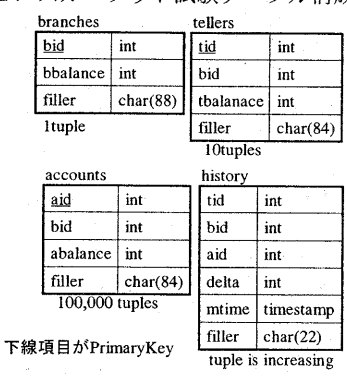


表６．トランザクションの内容

1	update accounts set abalance=abalance+DELTA where aid=AID;
2	select abalance from accounts where aid=AID;
3	update tellers set tbalance=tbalance+DELTA where tid=TID;
4	update branches set bbalance=bbalance+DELTA where bid=BID;
5	insert into history(tid, bid, aid, delta) values(TID, BID, AID, DELTA)

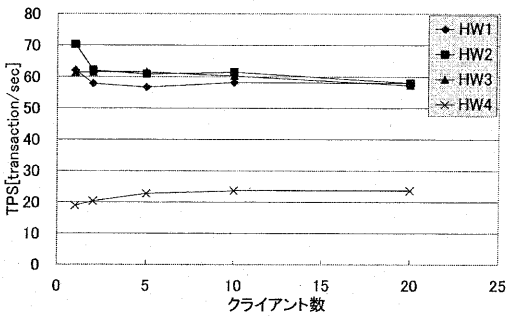


図３．異なるHW環境における
PostgreSQLスループット性能

4. プロトタイプのスケーラビリティ評価

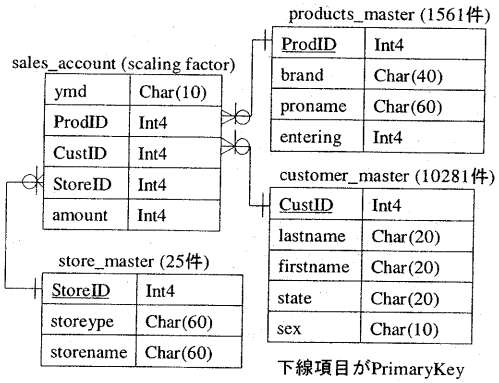
ここでの評価は、PDMSが備えるインタフェースはコマンドライン入力方式のみであること、使用できるSQL文に制約があることから、前章で行ったPostgreSQLの単体性能評価試験とは異なったものとなっている。

4.1. 評価環境

スループット試験とレスポンス試験の２つの評価試験を、PostgreSQL単体、PDMS２ノード(C-Node１台, S-Node１台)、PDMS３ノード(C-Node１台, S-Node２台)、PDMS５ノード(C-Node１台, S-Node４台)という４つの構成に対して行った。測定にはC-Node、S-Node共に表２で示したHW３を用いた。

測定で用いたテーブルの構成を図４に示す。”store_master”、”customer_master”、”products_master”はAll Node Table、”sales_account”はCustIDを配置属性としたPartitioned Tableとした。

図４．PDMS評価試験におけるテーブル構成



4.2. レスポンス性能

“sales_account”テーブルに300万件のデータを挿入した状態で、表７に示す８種類の各SQL文を１つのトランザクションとしてレスポンス時間を測定した。なお、Partitioned Tableにはインデクスは張っていない。

測定結果を図５に示す。レスポンス性能について、今回の試験ではクエリが単一のノードに処理要求される単一クエリと、複数のノードに処理要求される並列クエリの２種類があるが、図５の通り単一クエリ、並列クエリともにノード数の増加に伴ってレスポンスが向上している事がわかる。PostgreSQL単体とPDMS２ノードを比較すると、論理的にはC-Nodeアプリケーション及びネットワークのオーバヘッドがある分だけ、PDMSの方が余計な処理時間がかかるはずである。しかしながら、そのような顕著な傾向は見られなかった。これは、DBの構築条件などによって測定値のばらつきが生じているためと考えられる。

4.3. スループット性能

１つのトランザクションは”sales_account”テーブルに対する単純なINSERT文である。１クライアント当たりのトランザクションは５万件とし、クライアント数を５、１０と変化させて測定した。図６に示す測定結果の通り、台数の増加に応じて処理能力が向上している事が分かる。但し、１０クライアント条件における３ノードから５ノードへの性能向上率が鈍化している。これは、C-Nodeの処理能力がボトルネックとなっているためと考えられる。

表７．PDMS レスポンス性能試験で用いた SQL

ID	単一/並列	SQL文	結果件数
SQL1	単一	delete from sales_account where CustID=1051	90
SQL2	並列	delete from sales_account where StoreID = 11	144396
SQL3	単一	insert into sales_account values ('2004/04/04', 2000, 5000, 30, 100)	1
SQL4	並列	select count(*) from sales_account	296100
SQL5	単一	select count(*) from sales_account where CustID=1001	18
SQL6	並列	select sales_account.YMD, products_master.proname,sales_account.amount from sales_account, products_master where sales_account.ProdID = products_master.ProdID and products_master.ProdID = 1559	774
SQL7	並列	update sales_account set amount = 101 where storeid = 21	197712
SQL8	単一	update sales_account set amount = 500 where CustID = 2570	1800

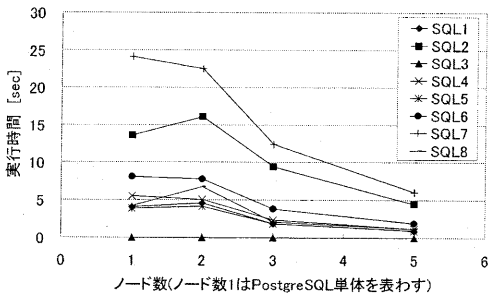


図５．PDMS レスポンス性能

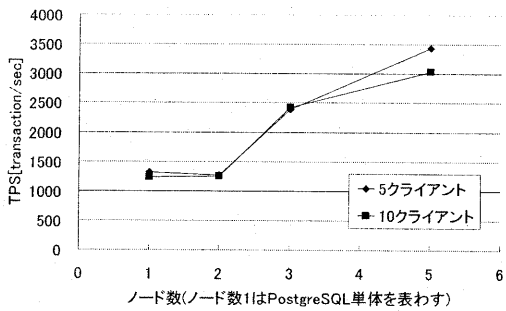


図６．PDMS スループット性能

5. 全体の考察

本試験結果から、データ蓄積/処理ノード(S-Node)の並列化を行うことによるレスポンス時間およびスループット性能に関する高スケーラビリティが確認できた。また、PostgreSQL 単体と比較した時の有意な性能向上が確認された。

４章で行った本プロトタイプに対する性能測定試験は３章で PostgreSQL 単体に行ったレスポンス性能試験と異なる試験であるため単純な比較はできない。しかし、Wisconsin Benchmark におけるインデクス無しもしくはインデクスの効果が無い場合と比較することで、並列分散化による性能向上とマシン性能を上げることによる性能向上との比較がある程度可能である。

例えば、レスポンス性能において、query１１８は JOIN を伴わない１テーブルの順スキャンであるが、HW１と HW３の間には 2 倍程度のレスポンス時間差がある。これに対して本プロトタイプのレスポンス性能において、SQL２，４などで、２ノードに対して３ノードで約２倍の性能を発揮することが示されている。したがって、HW３を３ノード以上の構成にした場合に、HW１に匹敵する性能を得ることができると推測される。しかし、HW１と HW３の間でレスポンス性能が１０倍以上の差を示しているクエリも多く存在しており、単純に並列分散化しただけでは埋めることのできない性能差があることも示されている。

スループット性能に関しては、ハードウェア性能による差は HW４を除くとほとんど見られなかった。一方、本プロトタイプにおいて、スループット性能は S-Node の台数によるスケーラビリティが確認されており、並列化による性能向上が可能であることが示された。

今回開発したプロトタイプは、使用できる SQL に制限はあるものの、適切な DB 設計を行うことで単体の DBMS では実現が困難な性能を得ることが可能であることを本結果は示している。実際に同様の方式を用いて大規模実用システムを構築した事例も報告されている[6]。今後、本プロトタイプに C ライブラリ、ODBC 及び JDBC といった汎用的なインタフェースを実装することで実用範囲を広げていく予定である。また複数ノードにまたがる処理に関するトランザクション保証については PostgreSQL に対応した 2 phase commit 方式 [7]や、可用性の確保については PostgreSQL を冗長化する”PG Replicate” [8]の組み込みや、HA クラスタ技術との連携を行っていく予定にしている。

本プロトタイプに用いた SQL レベルでの分配方式では、使用できる SQL の制限をなくすことは困難である。大量のデータを分析処理する DWH(Data Ware House)や、複雑なトランザクション処理への適用を考えた場合、このような機能の制限を撤廃することが重要な課題となる。また、本プロトタイプのように SQL レベルで分配し、S-Node それぞれに DBMS を稼動させる方式では、様々なオーバヘッドが存在するため非効率であり、また並列処理を考慮に入れたプラン

生成が不可能であるという問題がある。我々が最終的に目指している PDMS では、物理クエリツリーレベルでの分配を行う方式を採用し、クエリの意味解釈とデータの分散状態、S-Node のロード状態などの時事刻々と変化する統計情報など、使用可能なあらゆる情報を加味した上で並列処理の最適化を行う機構を実装する予定である。さらに、複数の C-Node を配置することで、C-Node の負荷を分散する構成を検討している。この場合、各 C-Node が保持しているメタ情報、トランザクション管理情報、キャッシュメモリなどを C-Node 間で同期をとるメカニズムが必要であり、複雑な管理機構を必要とする。トランザクション管理についてはトランザクション管理専用ノード(T-Node)の導入も現在検討しているところである。

また、並列分散化と合わせて PostgreSQL 単体についてもエンタープライズ化を進め、処理性能の向上、高可用化を進める予定である。

6. まとめ

本稿では、S-Node に Partitioned Table、All Node Table 方式を用いてデータを分散配置し、C-Node が SQL レベルでクエリを複数の S-Node で稼動している DBMS に分配し並列に処理を実行することで、スケーラブルな性能向上を図れることを示した。一方で、本プロトタイプの並列分散方式の限界について指摘し、物理クエリツリーを分配する方式の重要性を示した。

謝　辞

PostgreSQL 及び PDMS の性能測定に当たり、NTT データ先端技術株式会社の鈴木幸市氏、井久保寛明氏、柏木雄一郎氏に多くのご助言を頂いた。

文　献

[1] M. T. Ozsu, and P. Valduriez. "Principles of Distributed Database Systems Second Edition", Prentice Hall, Upper Saddle River, NJ, 1999.

[2] 宇山, 谷越他. "並列分散データベース管理システム PDMS の構成とデータ分割法", 2003 信学総大論文集, D, pp. 49, mar. 2003.

[3] Hector Garcia-Molina, Jefrey D. Ullman, Jennifer Widom. "DataBase System Implementation", Prentice Hall, Upper Saddle River, NJ 2000.

[4] 谷越, 宇山他. "PDMS における分散質問方式", 2003 信学総大論文集, D, pp. 49, mar. 2003.

[5] Jim Gray (編), 喜連川他 (訳). "データベース・ベンチマーキング", 日経 BP 社, 東京, 1992.

[6] 渡部. "大規模会員管理システムへの PostgreSQL の運用事例", 日本 PostgreSQL ユーザ会, PostgreSQL Conference 2003, may. 2003.

[7] 永安. "PostgreSQL における二相コミットとその応用", 日本 PostgreSQL ユーザ会, PostgreSQL Conference 2003, may. 2003.

[8] 三谷. "レプリケーションシステム、PGReplicate の現状と最新動向", 日本 PostgreSQL ユーザ会, PostgreSQL Conference 2003, may. 2003.