DBMS の並列分散化とその有効性

宇山 公隆　谷越 桂太　藤塚 勤也　澤井 健　近藤 公久

(株)NTT データ　〒104-0033 東京都中央区新川 1-21-2
E-mail: {uyamak, tanikoshik, fujidukak, sawait, kondoutdh}@nttdata.co.jp

あらまし　大規模クラスタ環境でスケーラブルな性能を有する並列分散 DBMS の開発を行っている。これまでに、入力 SQL を複数の DBMS に分配することによって並列分散 DB 処理を行うプロトタイプを構築し、並列分散化の有効性を確認した。しかし、このような SQL を分配する方式では、処理に限界が存在する。本論文では、DB 処理を統括するノード(C-Node)とデータを蓄積/処理するノード(S-Node)に DBMS の機能を分割して並列分散処理を行う DBMS の処理方式について報告する。特に結合演算に着目し、S-Node 同士でデータの授受を行うためのプリプランと呼ぶ処理をクエリプランに追加した。性能評価の結果、ノード数に応じたスケーラビリティとプリプラン方式による処理の有効性を確認した。

キーワード　並列 DB、問合せ処理、性能評価、最適化、並列プラン

Development of a Parallel and Distributed DBMS, and Evaluation of the Scalability

Kimitaka UYAMA　Keita TANIKOSHI　Kinya FUJIZUKA　Takeshi SAWAI
and　Tadahisa KONDO

NTT DATA Corporation　1-21-2 Shinkawa, Chuo-ku, Tokyo, 104-0033 Japan
E-mail: { uyamak, tanikoshik, fujidukak, kondoutdh}@nttdata.co.jp

Abstract　A parallel and distributed DBMS which has a scalable performance on a large scale cluster system has been developed. The scalability of a prototype system was confirmed. In this prototype system, a query from a client was re-written appropriately on a coordinator node (C-Node) and it was delivered to several subordinated nodes (S-Nodes). As the SQL was delivered from C-Node to S-Nodes on which an independent DBMS was running, it was extremely difficult to join the tables on different S-Nodes. This paper introduces the 'Plan' distribution method for the query distribution from C-Node to S-Nodes. To implement this method, DBMS functions are separated into C-Node and S-Nodes. The C-Node analyzes a query and generates a plan, and the S-Nodes execute the plan. The plan includes a 'Pre-plan' which assigns the way of data transfer between S-Nodes for the join operation between the nodes. The results of the evaluation of the prototype system using Plan distribution method show a scalable performance as good as the prototype system using SQL distribution method.

Keyword　Parallel DB, query execution, performance evaluation, optimization, parallel query plan

1. はじめに

IT 化の進む現在、大量の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 の役割が益々重要になっている。また、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。

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実現している。また、分散環境に対する DBMS への適用という点においては、過去から様々な研究がなされてきた[1]。

一方、Linux を始めとするオープンソースソフトウェアの機能・性能が向上し、基幹業務システムへの適用が推進されている。しかし、基幹業務システムにおいて必須である DBMS においては、オープンソースソフトウェアは商用の DBMS に比較して明らかに機能および性能面で劣っており、特に、並列分散化に関して

は開発が遅れている。

そこで我々は、大規模 Linux クラスタ環境でスケーラブルな性能を発揮する DBMS の開発を開始した。本開発の目的は、基幹業務システムに適用可能とするために、大量データを高速に処理でき、データの大規模化に柔軟に対応可能な DBMS を開発することである。今回はこの目的達成のための第一ステップとして、クラスタ技術や並列分散処理技術を DBMS にとりこむための基盤システムを開発した。本開発においては、最も機能・性能面で優れているオープンソースソフトウェアの DBMS の１つである PostgreSQL をベースにすることで、開発期間の短縮を図った。

2. 谷越ら[2]のシステムの問題点

谷越ら[2]の並列分散 DBMS では、PostgreSQL が稼動する複数のデータベースサーバ(S-Node)に対して入力クエリ(SQL)を分割配布するサーバ（C-Node）を置くことで並列分散処理を実現している。このシステムでは、独立に稼動する DBMS に対し SQL を分配することによって処理を実行させるため、以下の問題点が存在した。

(1) C-Node と S-Node 双方において、SQL の構文解析が必要
(2) 結合演算はノード間のデータ通信が不要なもののみ実行可能
(3) 並列処理を考慮したプランの最適化が困難
(4) トランザクション保証が困難

3. 並列分散 DBMS 設計

2 章で示した問題(1)を解決するために、構文解析から最適化の処理までは C-Node のみで実施し、プランレベルで処理を分割する。本システムでは、PostgreSQL を改造して DBMS の処理機能を DB 処理を統括するノード(C-Node)とデータ蓄積/処理ノード(S-Node)に分割する。容量及び処理能力のスケーラビリティを確保するために、S-Node の数は増減可能とする。本システムに於いては、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する。なお、本方式については、既に谷越ら[3]が報告している。さらに、将来的には C-Node も増減可能とし、負荷分散を図る予定である。

また、今回開発したシステムでは、2 章で示した問題(2)を解決するために、プリプラン方式を導入する。

なお、(3),(4) の問題点については、最適並列プラン生成アルゴリズムの導入、および、トランザクション管理ノードの追加や二相コミットの導入により解決する予定であるが、今回の実装では対象外とした。したがって、今回のシステムではトランザクション管理関係の処理は行わない実装となっている。

3.1. システム構成

図 1 に今回開発した並列分散 DBMS のシステム構成図を示す。

本システムは、単一の C-Node と複数の S-Node で構成されている。C-Node は 複数の S-Node に分散したデータの格納情報などのメタ情報を保持するグローバルシステムカタログを管理する。

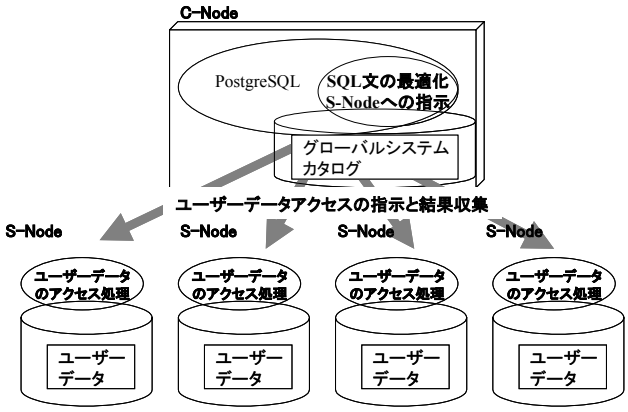


図１． システム構成図

C-Node ではクライアントからのクエリを受付け、構文解析、最適化を行う。ここまでは PostgtreSQL の機能そのままである。本システムでは、最適化処理後に生成される物理プランを並列処理用に変更して並列プランを生成する。この並列プラン生成に、グローバルシステムカタログ情報を用いる。この並列プランを処理を実行させる S-Node に送信する。また C-Node は S-Node で実行した処理結果を集約してクライアントに回答する。

S-Node は実際にデータを格納しているノードであり、物理的にデータへアクセスし、処理を実行する。S-Node においては、C-Node から送信されたプランに従って処理を実行するのみで、クライアントからのクエリや分散環境について再解析する必要はない。

3.2. データ配置方式

本システムでは宇山ら[4] のデータ配置方式を採用し、テーブル毎に以下のデータ配置方式を設定可能である。データの性質に応じてデータ配置方式を適切に設定することで、並列分散データベース環境における性能の向上を図ることが可能である。

(1) Partitioned Table

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。

(2) Specific Node Table

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

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。

3.3. 並列プラン実行方式

本システムでは、3.1 節で述べたとおり、C-Node がグローバルシステムカタログを参照して生成する並列プランを S-Node に対して送信することで並列処理を行う。

ここでは、以下の基本的な SQL が実行される場合を例に、本システムにおける並列プランと並列処理方式について詳細に説明する。

SELECT max(int_attr) FROM table_a;
但し、int_attr に対するインデックスは存在しない。

まず、この SQL が実行される時、オリジナルの PostgreSQL が生成する物理プランツリーを図２に示す。

物理プランは図２のようなツリー構造で表現されており、各ノード（箱）は処理、矢印の先の箱は上位の処理より先に実行する必要のある処理を示している。したがって、ツリーの枝の末端から処理が実行される。この例では、まず T_SeqScan（順走査）を table_a に対して実行し、その結果に対して、T_Agg（集約関数：最大値）を実行することを表している。

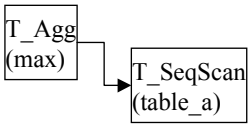


図２． オリジナルの物理プランツリー

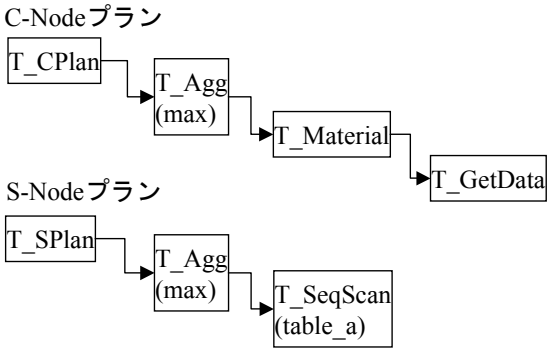


図３． Partitioned Table 時の物理プランツリー

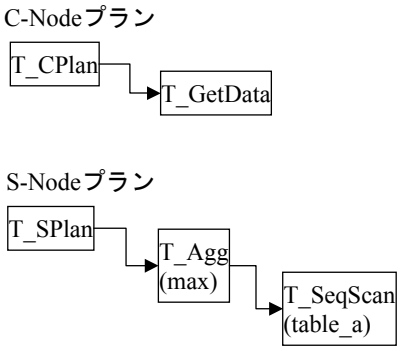


図４． All Node Table または Specific Node Table 時の物理プランツリー

本システムでは、図２の物理プランから C-Node 用（C-Node プラン）と S-Node 用（S-Node プラン）の二つの並列プランを生成し、S-Node プランを複数の S-Node に並列実行させる。

並列プランは、同じ SQL に対するものであっても処理対象テーブルのデータ配置方式によって異なる。table_a が Partitioned Table である場合には図３、table_a が All Node Table または Specific Node Table である場合には図４の様になる。図中のツリーの先頭の T_CPlan, T_SPlan はそれぞれ、C-Node プラン、S-Node プランを示すタグである。

図３と図４を比較して分かるように、S-Node プランはどちらも同じプランとなっている。しかし実際の実行時には、Partitioned Table に対する処理では、S-Node プランはすべての S-Node に送付される。一方、All Node Table または Specific Node Table の場合は、特定の１つの S-Node にのみ送付される。

また、Partitioned Table に対する処理では、 T_Agg (max)処理が S-Node だけでなく C-Node に対するプランにも含まれている。これは処理対象が Partitioned Table の場合、集約演算系の処理は、各 S-Node で実行した処理結果を C-Node において集めた上で、最終的

な結果を得るための演算処理が必要であるためである。

これに対して All Node Table または Specific Node Table の場合は、1 つの S-Node 内に閉じた演算で最終結果が得られるため、C-Node において処理を行う必要はない。このため、図 4 の C-Node プランには T_Agg(max)は存在しない。

3.4. プリプランによる S-Node 間データ転送方式

分割方式の異なるテーブル同士の結合演算を実行する場合には、ノード間でデータの授受が必要となる。例えば 1 つの S-Node のみに存在する Specific Node Table と全ての S-Node にデータが分割配置されている Partitioned Table の結合演算を実行しようとすると、どのノードで結合演算を実行するにせよ、S-Node 間でデータ通信が必要となる。すなわち、結合演算を実行するにあたって、処理対象テーブルをもつ S-Node から、処理を実行する S-Node に対して処理に必要なデータを転送する必要がある。

ノード間通信を行ってデータを授受し、結合演算を実施する方式として、合田ら[5]による報告がある。合田らの方式におけるプランとしては一般的に知られている論理プランを使用している。このプランを元に、分散したノードで結合演算をするために、結合キーでハッシュをかけ、ハッシュ値に対応するノードに一度データを再配置することで、各ノードでハッシュ結合を実施し、その後最終的に結果をまとめるというハッシュ結合を前提とした方式をとっている。

本システムにおいては、S-Node プランツリーにノード間のデータの授受を指示するプリプラン（PrePlan）を埋め込む方式を採用した。

プリプランによる S-Node 間のデータ転送方式では、C-Node から S-Node への指示をプランとして一括送信可能である。また、S-Node はクエリの意味内容やテーブルの配置情報を一切考慮せずに、受け取ったプランを実行するだけでよい。

プリプランの生成過程、S-Node プランへの埋め込み方法、および、プリプランの実行方式を、結合演算を含む以下の SQL を例に説明する。ここで、table_p は Partitioned Table、table_s は Specific Node Table である。

```
SELECT * FROM table_p a, table_s b
        WHERE a.int_attr = b.int_attr;
```

まず始めに、オリジナルの PostgreSQL が上の SQL に対して生成するプランツリーを図 5 に示す。図 5 のプランは、table_p および table_s それぞれに対して、T_SeqScan(順次走査)を実行し、その結果を T_Sort(ソート)し、ソート済みの結果どうしを T_MergeJoin(マージジョイン)するプランとなっている。

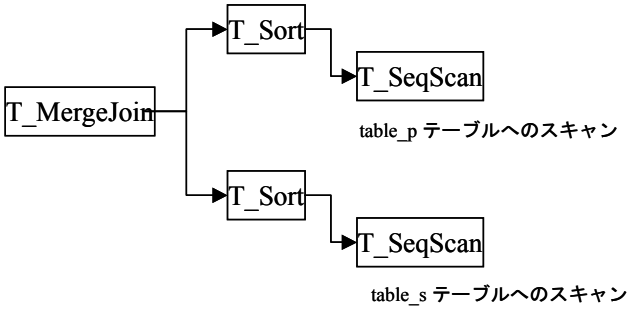


図 5． 分割前の結合演算を含むプランツリー

しかし、このプランはテーブルが異なるノードに分散して存在することを考慮していない。

本システムではまず、SQL 中に存在するテーブルが Specific Node Table か Partitioned Table かをグローバルシステムカタログ情報から C-Node が判断する。C-Node は、処理対象の各テーブルの配置方式から、処理を実行するノードと処理に必要なデータを所持するノードを決定する。この例では、Partitioned Table を所持する S-Node で処理を実行し、Specific Node Table である table_s を所持する S-Node から table_s のデータを転送する。

以上の処理をプリプランとして S-Node に埋め込む。この図 6 の S-Node プランの例では、マージジョインノード後の table_s 側に T_PrePlan(プリプランタグ)を挿入することで、それ以下のプランツリーがプリプランであることを示している。

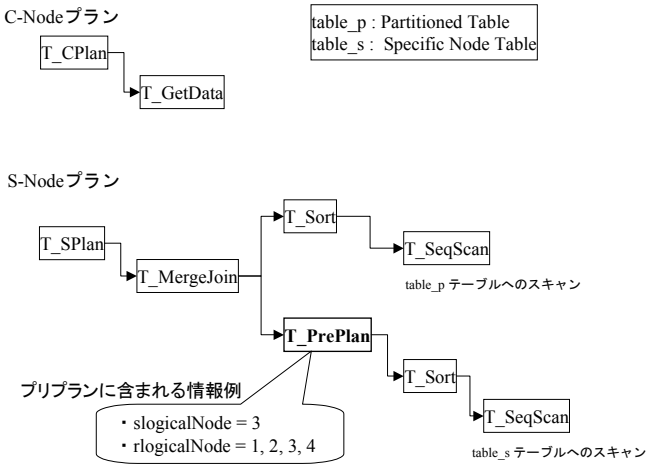


図 6． 分割後の C-Node, S-Node 用プランツリー

このプリプランが埋め込まれた S-Node プランは C-Node から全ての S-Node に対して送信される。S-Node プランを受け取った各 S-Node は、送信され

た S-Node プランを実行する。S-Node プラン中、プリプラン実行時点においてプリプランタグノードが持つ二つの変数、slogicalNode と rlogicalNode を参照して S-Node 間のデータ転送が行われる。変数 slogicalNode は T_PrePlan タグ以下のプリプランを実行するノードのノード番号で、rlogicalNode が slogicalNode のプラン実行結果の転送先ノード群のノード番号リストを示す。

ここでの例(図 6 の吹き出し参照)では、slogicalNode＝3 であり、ノード番号が 3 の S-Node が T_PrePlan 以下のプリプランを実行する。また、rlogicalNode＝{1, 2, 3, 4} であるため、ノード番号が 1, 2, 3, 4 の S-Node は、ノード番号が 3 の S-Node からのプリプラン実行結果を受け取る。実際にはデータを受け取る側の S-Node はデータが転送待ち状態となる。

各 S-Node は、プリプランによるデータ転送終了後、それ以降のプランを実行する。すなわち、それぞれの S_Node に存在する table_p をソートし、送信されたプリプラン実行結果とマージジョインを実行する。

一方、C-Node プランは、以上で示した各 S-Node の処理結果をすべて受け取ることを示している。

4. 性能評価

本性能評価では、まず、谷越ら[2]の評価結果との比較を行うことにより、DBMS(PostgreSQL) を機能的に分割する方式による並列分散化によるスケーラビリティの検証を行う。

次に本システムで実装したプリプラン方式の有効性の検証を行う。

4.1. スケーラビリティ検証

4.1.1. 試験環境

評価環境は谷越ら[2]の報告と同じ環境を用いた。

測定には C-Node、S-Node 共に以下の表 1 で示すマシン環境を使用した。

また、測定で用いたスキーマ構成を図 7 に示す。

表 1．測定マシン環境

OS	RedHat Linux 7.3
CPU	Pentium Ⅲ 1.26GHz
メモリ	256 MB
HDD	IDE 7,200 rpm

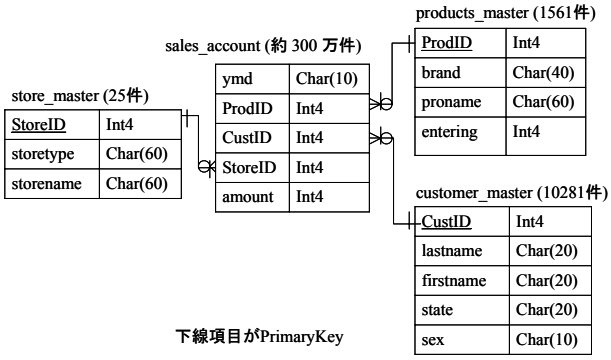


図 7．評価試験用スキーマ構成

評価試験で用いたテーブルの分割方式は以下のとおりであった。

All Node Table:

'store_master', 'customer_master', 'products_master'

Partitioned Table:

'sales_account'

(CustID を配置属性としてハッシュ分割)

4.1.2. 評価試験項目

上述の環境においてレスポンス時間を測定する評価試験を実施した。評価試験で用いた SQL を以下の表 2 に示す。

表 2．評価試験で使用した SQL 文一覧

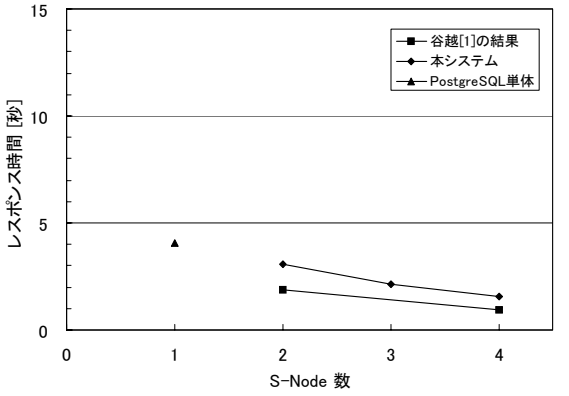
ID	単一/並列	SQL文	結果件数
SQL1	単一	DELETE FROM sales_account WHERE CustID = 1051;	90
SQL2	並列	DELETE FROM sales_account WHERE StoreID = 11;	144396
SQL3	単一	INSERT INTO sales_account VALUES ('2004/04/04', 2000, 5000, 30, 100);	1
SQL4	並列	SELECT sales_account.YMD,products_master.proname,sales_account.amount FROM sales_account, products_master WHERE sales_account.ProdID = products_master.ProdID AND products_master.ProdID = 1559;	774

本評価では、C-Node は 1 台固定、S-Node を 2, 3, 4 台と変化させて測定を行った。また、PostgreSQL 単体に対する測定も行った。PostgreSQL 単体に対する測定では、すべてのテーブルを非分割で一台のサーバに配置した。それ以外の条件は、本論文でのシステム評価環境と同一であった。

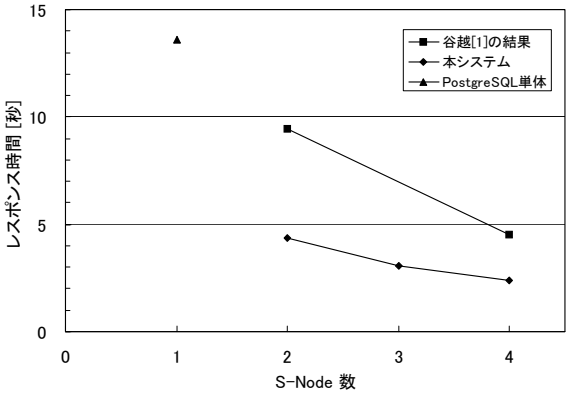
4.1.3. 評価試験結果

図 8 にレスポンス時間測定結果を示す。図 8 (a)～(d) 全体を通して、谷越ら[2]の結果と今回の測定結果は傾向が似ていることがわかる。

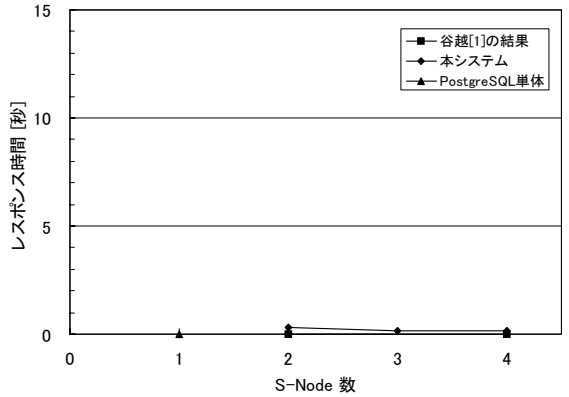
また、図 8 (c)を除けば、S-Node 4 台まではほぼ直線的に S-Node の台数に対してレスポンス時間が短くなっていることがわかる。すなわち、この台数の範囲ではスケーラブルな性能が得られているといえる。



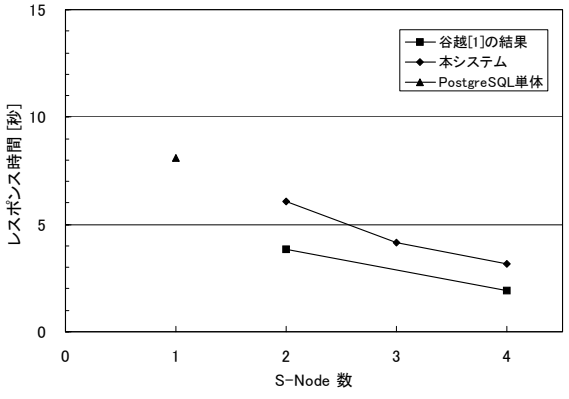
(a) SQL1のレスポンス時間とノード数の関係



(b) SQL2のレスポンス時間とノード数の関係



(c) SQL3のレスポンス時間とノード数の関係



(d) SQL4のレスポンス時間とノード数の関係

図８． 各 SQL に対するレスポンス時間とノード数の関係

この効果は、クエリが対象とするテーブルが Partitioned Table であるので、S-Node が増えるに従い、S-Node 一台あたりの処理件数が減少するためと説明できる。

図８(c)では S-Node の台数の違いによってレスポンス時間の差が見られない。これは、SQL3 の処理が INSERT であって、かつ、このテーブルにはインデックスも張っていないので、処理性能は対象テーブルの件数に左右されないためである。

図８(a),(c),(d)においては、本システムの方が谷越ら[2]の結果よりもレスポンス時間が長い傾向がある。プロファイリングの結果、C-Node と S-Node 間で構造体データを通信するために必要となるデータの構造変換処理の負荷が大きいことが分かった。

しかし、図８(b)においては、上述した結果と異なり、本システムの方が谷越ら[2]の結果よりもレスポンス時間が短い。これは、本システムにおいてはログ関連の処理が未実装であるために、ログ出力を行っていない事が要因として考えられる。つまり、図８(b)の SQL2 では、処理対象とする件数が 14 万件強と、他の クエリと比較して大量にあるために、ログやチェックポイントの処理といったオーバーヘッドがレスポンスタイムに大きく影響していると考えられる。

4.2. プリプラン方式の有効性検証

プリプランの実用性を検証するための評価を行った。ここでは、プリプランが生成され、S-Node 間でのデータ転送が起こる場合と、プリプランを必要とせずデータ転送なしにクエリが実行される場合とを比較する。

4.2.1. 試験環境

ここでは、4.1.1 のスケーラビリティ検証試験と同一の環境を用いた。

4.1.2 で実施した SQL4 の測定においては products_master テーブルが All Node Table であったためプリプランは生成されなかった。ここでは、All Node Table であった products_master テーブルを Specific Node Table に変更した条件で、表２の SQL4 を実行した場合のレスポンス性能測定を実施した。

本試験環境では、products_master テーブルを Specific Node Table としているために、表２中の SQL4 の検索処理を実行すると、Partitioned Table と Specific Node Table の結合演算となり、products_master テーブルに対する処理がプリプランとなる。

4.2.2. 試験結果

性能試験結果を図９に示す。

図９から、プリプランを含む場合においても、S-Node の台数が増加するに伴い、レスポンス時間が短くなっており、S-Node の台数増加に伴うスケーラビリティが確認できる。また、プリプランを含む場合と、含まない場合においてレスポンス時間に明確な差がついていない事も確認できる。

以上の結果から、今回の試験で測定した環境では、プリプランで実行される S-Node 間の通信は処理性能のボトルネックにはならないことが示された。但し、今回の試験で使用したクエリの SQL4 では、WHERE 句において、"products_master.ProdID = 1559" としているため、プリプランを実行する S-Node から返ってくるタプル数は 1 つだけであり通信量が非常に少なく、

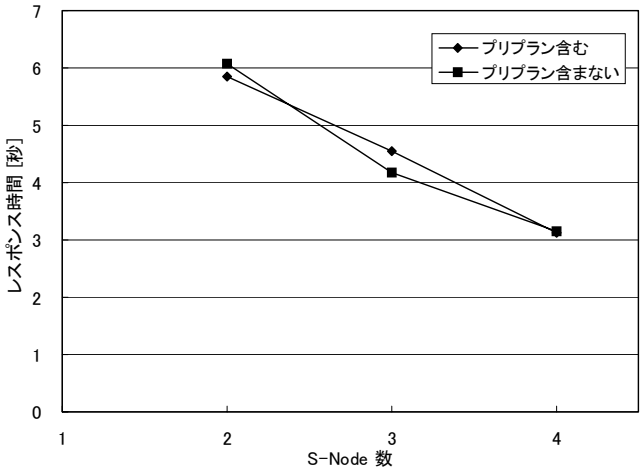


図９． プリプランの有無によるレスポンス時間とノード数の関係

大量のタプルを S-Node 間で授受する必要が発生する場合では、S-Node 間のデータ転送による通信がボトルネックとなる可能性が大きい。

しかしながら、プリプランの生成を必要とする Specific Node Table は、更新頻度が高く、比較的小規模なテーブルに使用することを想定している。S-Node 間の通信がボトルネックとなるような状況を避けるようにデータベース設計を行い、テーブルの配置方式の選択を慎重に行えば、プリプラン方式は、実際の使用においても十分使用できるものであると考えられる。

5. 考察

本システムでは、プリプランを埋め込んだ並列プラン方式の導入によって谷越ら[2]のシステムにおいて指摘された第２章で示した問題の(1)と(2)を解決する方式を検討し、実際に PostgreSQL を改造して実装したシステムを開発した。また、並列分散 DBMS 環境におけるレスポンス性能のスケーラビリティを確認した。

プリプラン方式の導入により、分割方式が異なるテーブルに対する結合演算も複数の S-Node において分散実行することが可能となった。また、4.2.2 で示したように、プリプランを含むクエリにおいても、S-Node の増加に伴いレスポンス性能が向上しており、プリプランを含まない場合の性能と比較しても性能の劣化は見られず、この方式の実用性が確認された。しかし、プリプランによるデータ転送量が大きくなると通信ボトルネックが予想される。どの程度のデータ転送量が発生する場合まで高性能が維持されるかなどの検証実験を行うことによって、適切にテーブルの配置方式を設定する指標としたいと考えている。

並列分散データベース環境において、テーブルが別々のノードに配置されている場合にテーブルの結合演算を実行するためには、どこか一つのノードにデータを集め、一つのノードで一括して結合演算を実施する方式が考えられる。この場合は、結合演算を実施する特定のノードへの処理の集中がボトルネックとなる問題が生じる事となる。本システムでは、Partitioned Table と Specific Node Table の結合演算の例で示したように、Specific Node Table のデータを Partitioned Table を所持する複数の S-Node に転送することで、結合演算を並列処理可能にしている。

プリプラン方式を導入せずに、本システムと同様に上述の例の結合演算を分散処理させる場合には、結合演算に必要なデータを送信する側のノード（ここでは Specific Node Table を所持する S-Node）に、まず、送信すべきデータと送信相手ノード情報とともに転送命令を送信する必要がある。一方、結合演算を実行するノード(Partitioned Table を所持する複数の S-Node）に対しては、受信すべきデータと相手ノード情報とともに受信待ち命令を送信する必要がある。さらに、これらの処理終了を待って C-Node は実際のプランを全ノードに送信する必要がある。もし、これらのデータ転送命令をプランとともに送信する場合であっても、S-Node 毎に異なるプランを配布する事となる。そのため、C-Node において、複数のプランを生成するというオーバーヘッドが生じる事になる。

一方、本システムで採用したプリプラン方式では、プリプランタグノードにデータを送信する S-Node とデータを受信する S-Node を変数として埋め込み、プランを受け取った S-Node 側で送信処理を実行するノードであるか受信処理が必要なノードであるかを判断可能としている。これにより、転送命令をプランに埋め込んで一括で並列プランとしてデータ転送の指示を行えるだけでなく、すべての S-Node に対してまったく同一のプランを配布することが可能である。これに

より、C-Node が生成するプランは一つだけで済むため、C-Node の処理の負荷を軽減することが可能となっている。

さらに、プリプランを導入した事で、分割情報等のシステムカタログは C-Node のみで保持すれば良く、S-Node にこれらのシステムカタログを保持する必要が無い。現在 S-Node を動的に追加/削除する機能を検討しているが、この方式によってシステムカタログ情報の更新は C-Node についてのみ考慮すれば良いことになり、この点についても有効な方式であると考えられる。

本システムはまだ開発途中であり、今後以下の課題について継続して開発を行う。

まずは、今回実装から除外したトランザクション関連の処理を実装する。トランザクション管理ノードの追加や二相コミットの導入を行い、並列分散 DBMS として実運用に耐えうるシステムとしていく。

次に、並列プランについては、今後 S-Node のデータの分散状況などをシステムカタログとして追加し、分散状況に応じた最適化を行うことを予定している。

また、本システムではオリジナルの PostgreSQL が生成するプランを分割することで並列プランを生成した。既存の物理プランを分割する事で、並列プランを生成する本方式は、既存のソースコードに対する修正を必要とせず、並列プラン生成モジュールを追加する形式であるため、改造が楽であるとともに、オリジナルの PostgreSQL のバージョンが上がった場合においても、対応が比較的簡単であるというメリットがある。

しかし、オリジナルの PostgreSQL が生成するプランはテーブルの分散環境を考慮せずに最適化された結果である。したがって、生成されたプランは分散環境においては必ずしも最適とは限らない場合があると予想される。データの分散環境を考慮した並列プラン生成および最適化を PostgreSQL の最適化段階から考慮に入れる方式も合わせて検討する予定である。

本システムにおいては、4.1.3 で述べたように、C-Node と S-Node 間のデータ通信に伴う内部データ構造の変換処理に伴うボトルネックが確認された。
現在、C-Node と S-Node 間の通信で使用しているコンテナとして、基本的には PostgreSQL の内部で使用している構造体をそのまま使用している。
この構造体は通信を考慮した構造とはなっていないため、非常に複雑な構造となっている。そのため、構造体の内部をスキャンし、必要な情報を取り出し、通信できる形式に型変換を行う処理に多くのリソースを必要とする事が判明した。今後この C-Node と S-Node 間の通信方式について再検討する予定である。

今回は C-Node 1 台、S-Node が最大で 4 台の環境において性能試験を行い、そのスケーラビリティを確認した。しかし、S-Node の台数が増加し、S-Node 全体としての処理能力が向上すると、現在 1 台である C-Node の処理が限界に達してしまい、それがボトルネックとなることが予想される。そこで、C-Node も並列化できる様に現在検討を進めている。C-Node を並列化するとトランザクション管理が問題となるため、系全体でユニークなトランザクション ID を払い出す機構の開発を現在行っている。

更に、高可用性の確保、系全体を一貫性を持ってバックアップ可能な機構の開発、そして、クラスタシステムの運用性を向上するためのツール類の開発なども行う予定である。

6. まとめ

本報告では、DBMS の機能を、DB 処理を統括するノード(C-Node)とデータを蓄積/処理するノード(S-Node)に分割し、C-Node で生成した並列実行プランを複数の S-Node に配布することによって動作するシステムのスケーラブルな性能を示した。またこの時、複数の S-Node を配置することにより、単体の PostgreSQL のレスポンス性能を上回る事を確認した。

さらに、異なるノードに存在するテーブル間の結合演算において必要となる S-Node 間のデータ授受に対するプリプラン配布方式の有効性を示した。

文　献

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

[2] 谷越, 宇山他. "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-", 信学技報 DE2003-23, pp.91-96, Jul. 2003.

[3] 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.

[4] 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.

[5] 合田, 田村, 喜連川."並列データベースカーネル DBKernel を用いた高速情報検索処理:TREC データによる実験", 電子情報通信学会 第 12 回データ工学ワークショップ(DEWS2001), 3A-3, 2001.3