2004－OS－96　(8)
2004／6／17

Linux PostgreSQLの TPC-Cベンチマークツール開発及び性能向上方式の検討

中村 良太[†]　持永 浩太郎[†]　中村 武雄[†]
朴 紅淑[†]　梅野 英典[†]

PostgreSQL は豊富な機能を備えたオープンソースのデータベース管理システム（DBMS）である．これは現在，様々な所で利用されているが，大規模運用においては性能の面から商用 DBMS を用いることが多い．本論文ではこの PostgreSQL に焦点を当て，低コストで高性能・高信頼な DBMS の実現を目的とし，商用 DBMS に劣らない性能のための向上方式を検討する．そこで，TPC-C に準拠したベンチマークを作成し評価を行う．性能評価においては，メモリチューニングやテーブル分散について試作・実験し，これらの有効性を検証した．また TP モニタを作成し，DBMS への接続数を管理することでメモリ負荷の軽減による性能向上を確認した．

Development of TPC-C Benchmark Tools and Evaluation of Performance Improvement Methods for PostgreSQL in Linux

Ryota Nakamura,[†] Kotaro Mochinaga,[†] Takeo Nakamura,[†]
Honshuku Paku[†] and Hidenori Umeno[†]

The PostgreSQL is one of major open source database management systems (DBMS). It is widely used in many small or medium scale systems, but business DBMSs are usually used in large scale systems because they have better performance. This paper describes the methods for improving performance of the PostgreSQL. Our objective is to realize high performance and reliability DBMS with low costs by using it. We have developed transaction processing performance council benchmark c (TPC-C) tools and presented performance measurement of the PostgreSQL. In the benchmarking, we investigated and evaluated the memory tuning and the data distribution, implemented transaction processing monitor (TPM), and finally confirmed its efficiency due to its controlling the number of connections to the DB server.

1. はじめに

近年，幅広い分野でデータベース管理システム（DetaBase Mamagement System, DBMS）が利用されている．また取り扱うデータの大規模化・複雑化に伴い，DBMS にはより高い性能が要求されている．これまでに，商用・非商用を問わず様々な DBMS が開発されてきた．その中で，無償のオープンソースであり豊富な機能を備えた DB 管理ソフトとして，PostgreSQL が挙げられる．現状では，PostgreSQL は商用 DBMS にやや劣る部分がある．従って，大規模運用においては商用 DBMS を用いることが多い．しかし，商用 DBMS は非常に高価であり一般ユーザ向けではない．そこで我々は，低コストであるが商用と同等以上の高信頼・高性能 DBMS を実現しようと考えた．

本研究ではこの PostgreSQL に焦点を当て，専用のベンチマークツールを作成しその性能を評価する．それとともに，性能向上についての方式を検討・試験しそれらの有効性を検証する．ベンチマークには TPC-C というオンライントランザクション用の評価基準を採用し，その仕様にしたがって設計・実装を行った．性能向上方式については，メモリチューニングやデータ分散について検討を行い，作成したベンチマークを用いて性能を評価した．また測定によって判明した接続数問題について，TP モニタを作成し接続・通信管理を行うことで，性能の改善を確認した．

2. PostgreSQL

本章では，研究に取り上げた PostgreSQL について紹介する．

† 熊本大学大学院 自然科学研究科
Faculty of Science and Technology, Kumamoto University

2.1 概　　要

PostgreSQL とは，カリフォルニア大学バークレイ校で開発された POSTGRES, Version 4.2 から発展した伝統的な RDBMS で，特定の型の属性を含む名前付きリレーションの集合から成るデータモデルをサポートしている．また，商用データベースシステムに劣らない機能を備えたオープンソースであり，世界中のボランティアが現在も開発に参加している．

2.2 アーキテクチャ

PostgreSQL はクライアント/サーバモデルを用いている．PostgreSQL のセッションはサーバとクライアントアプリケーションの協調動作するプロセス (プログラム) から構成される．サーバプロセスはバックエンドと呼ばれ，データベースファイルを管理し，クライアントアプリケーションからのデータベースの接続を受け付け，クライアントに代わってデータベースに対する処理を行う．データベースサーバプログラムは postmaster と呼ばれる．

ユーザのクライアントアプリケーションはフロントエンドと呼ばれ，ユーザのデータベース操作を行う．その性質上非常に多様性があり，テキスト指向のツール，グラフィカルなアプリケーション，データベースにアクセスし web ページを表示する web サーバ，あるいはデータベースに特化した保守ツールなどがある．

2.3 トランザクション

データベースを更新する際，連続・一括して更新しなければならない作業単位が重要となってくる．データを複数処理したい場合は，この要求をまとめて処理する必要がある．このデータベースの処理単位を**トランザクション（Transaction）**という．トランザクション処理の流れを図 1 に示す．

一般のアプリケーションでは，1 度の処理で 1 命令というものだけではなく，複数のデータ処理を要するものが多い．このトランザクションの概念を用いると，まとまった処理を一括して処理でき，信頼性の高い処理を実現できる．しかし，これを実現するにはさまざまな制約を満たさなければならない．例えば，データを正しく更新し保持するためには，まとまった処理をすべて実行しトランザクションを完了させるか，もし途中で失敗してもトランザクション処理の前まで状態を戻し，途中までの処理を取り消す必要がある．特に，トランザクション処理の完了を**コミット（COMMIT）**，処理の取り消しを**ロールバック（ROLLBACK）**と言う．トランザクション処理では，以上のようなトランザクション単位でのデータの正確性や並列処理されるトランザクション同士の競合など，複数の問題を解決しなければならない．これらの満たすべき種々の性質を **ACID 特性**という．

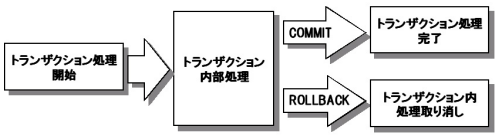

図 1　トランザクション処理の流れ

3. TPC-C ベンチマーク測定プログラム

ベンチマークとは，コンピュータの処理速度や処理能力といった性能を評価・比較するための指標である．また，TPC とは Transaction Processing Performance Council(オンライン・トランザクション処理性能評議会) という団体の名称であり，コンピュータ・システムのトランザクション処理性能を測定するための仕様の策定，および測定結果の真偽や公平性の審査を行う目的で，主にコンピュータ関連の有力企業が集まり設立された非営利団体である．

TPC-C はこの団体が定義したベンチマークである．

3.1 概　　要

図 2 は TPC-C のビジネス環境での倉庫，地区，顧客の階層を示している．TPC-C ベンチマークは業務処理性能の評価に用いられ，オンライントランザクション処理を想定している．ここで**オンライントランザクション処理（OnLine Transaction Processing, OLTP)** とは，複数の端末がサーバにメッセージを送りホストコンピュータがそのメッセージに従いデータベース処理を行い，処理結果を即座に端末に返すような処理である．

TPC-C が対象とするエミュレート環境は卸売会社のトランザクション処理システムである．一つの会社があり，その会社は商品がストックされたいくつかの倉庫を持っている．そして各倉庫ごとに 10 の担当地区があり，各地区には 3000 人の顧客が存在する，つ

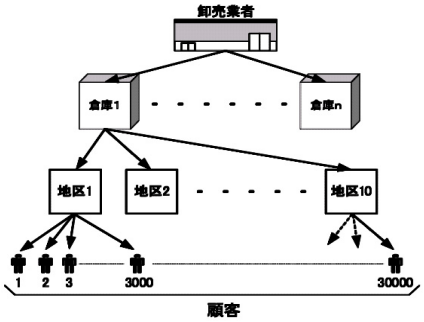

図 2　TPC-C ベンチマーク・エミュレートモデル

まり顧客が 1 倉庫に 30000 人いる環境をエミュレートする．会社には 100000 の商品（アイテム）があり，顧客は会社に新しい注文をし，あるいは現在の注文の状態を問い合わせる．会社のシステムとしては，顧客から支払いを受け入れ，配達の注文処理をし，在庫レベルを調査する．

3.2 データベース構成

TPC-C ベンチマークのデータベースは表 1 における 9 つのテーブルから成る．ただし，テーブルサイズにおいてはおおよそのもので，実際は DBMS の格納方式によって異なる．

表 1 TPC-C のデータベース構成

テーブル	サイズ [KB]	詳細
WAREHOUSE	0.089×n	倉庫に関するデータ
DISTRICT	0.950×n	地区に関するデータ
CUSTOMER	19650×n	顧客に関するデータ
HISTORY	1380×n	支払いの履歴データ
NEW-ORDER	720×n	新規注文データ
ORDER	72×n	注文データ
ORDER-LINE	16200×n	注文の詳細情報
STOCK	30600×n	在庫情報
ITEM	8200	商品データ

※ n は倉庫数を表す．

3.3 規定トランザクションと要求仕様

TPC-C ベンチマークで処理されるトランザクションは 5 種類ある．各トランザクションの動作内容と発生頻度は表 2 のようになる．表 2 を見ると，New-Order トランザクションには発生頻度が設定されていないが，他のトランザクションは最低値が設定されている．これらの設定値より，New-Order 以外のトランザクションにおいては，ベンチマークプログラムの実行中に処理されるトランザクションの割合が表の最低値以上でなければならない．

表 2 トランザクションの内容と発生頻度

トランザクション	内容	発生頻度
New-Order	商品の新規注文	
Payment	注文商品の代金処理	43.0%以上
Order-Status	注文の確認作業	4.0%以上
Delivery	未配達の注文を配達	4.0%以上
Stock-Level	在庫情報の確認	4.0%以上

3.4 応答時間制約

各トランザクションは，各端末上でそれぞれをオペレータが操作することによって発生する．端末では次のような動作がエミュレートされる．

(1) 各端末での初期メニュー画面の表示
(2) 実行したい操作 (トランザクション) の決定
(3) 選択されたメニューに応じた初期画面の表示
(4) 初期値の入力（必要な場合）
(5) 操作 (トランザクション) の実行
(6) 実行結果の表示
(7) 表示の終了・確認

以上のような動作はエミュレートされる各端末すべてで独立して行われる．

またそれぞれの時間には以下のような制約がある．

- 少なくとも全てのメニュー選択のうち 90%は 2 秒より短いメニュ応答時間を持たねばならない．
- 各トランザクションタイプに対し入力時間は一定であり，最低でも New-Order は 18 秒，Payment は 3 秒，Order-Status，Delivery，Stock-Level はそれぞれ 2 秒ないといけない．
- 各トランザクションタイプに対し，すべてのトランザクションのうち少なくとも 90%において New-Order，Payment，Order-Status，Delivery はそれぞれ 5 秒，Stock-Level は 20 秒よりもそれぞれ短い応答時間を持たねばならない．

3.5 測 定 基 準

TPC-C では，1 倉庫につき 10 端末がサーバへ要求を送る仕様となっている．また，測定した MQTh（Maximum Qualified Throughput）を報告するために使われる基準として，1 分ごとに処理される新規注文数（tpmC）が使用されている．これは，ロールバック（取り消し）されたトランザクションを含む New-Order トランザクションの一分間あたりの処理量を意味している．また仕様として，1 倉庫あたりの tpmC 値は 9〜12.86 内でなければならない．この数値は，OLTP での運用レベルから限界値までを表している．

3.6 実　　装

今回は，Linux 上で動作する PostgreSQL 用 TPC-C ベンチマークツールを作成した．また本研究チームでは，某商用 DBMS についても同様のベンチマークツールを開発しており，現在では商用との性能比較・課題検討なども行っている．

以降の章では，このベンチマークを用いて検討する性能向上方式について評価を行う．

4. パフォーマンス・チューニングと性能評価

データベースシステムのチューニングとは，システムの持つ性能を最大限に発揮できるように利用可能なリソースの中でユーザが求めるシステム機能を実現することである．また，ユーザからの要求とパフォーマンスの変化を観察し劣化を予防することも重要なチュー

ニング作業である．つまり，良好なパフォーマンスの維持とトラブルの予防，リソース (人材・機器・費用・時間) の有効利用を図ることが主な目的である．

今回は，作成した TPC-C ベンチマークツールでの性能測定に際して，以下のチューニングを行った．

4.1 カーネルチューニング

カーネル側では，以下の項目について調整を行った．

- **/proc/sys/vm/bdflush**
  bdflush プロセスのディスク書き込み間隔
- **/proc/sys/kernel/shmmax**
- **/proc/sys/kernel/shmall**
  DBMS が利用できる共有メモリの最大値

PostgreSQL は OS 上における 1 つのソフトウエアであり，その性能は OS 自身の管理機構に大きく影響する．ここでは OS 上の余計なサービスの停止や，ユーザプロセスへのリソースの提供などに注目した．

4.2 メモリチューニング

メモリチューニングは，性能測定において非常に影響力が大きい項目であった．これは PostgreSQL の処理のほとんどがメモリに確保されたバッファを利用して処理を行うためと考えられる．PostgreSQL はデータをメモリバッファに確保し，バッファにダーティビットがつくまでキャッシュをディスクに書き込まない．よって，メモリを効率的に確保できればディスクアクセスが減少し，パフォーマンスの向上が期待できる．今回のメモリチューニングでは，PostgreSQL サーバの設定として表 3 の項目を調整した．

表 3 メモリチューニング項目

項目	説明
MAX_FSM_RELATIONS	空き領域管理テーブル数
MAX_FSM_PAGES	空き領域管理ページ数
SHARED_BUFFERS	共有バッファ数
SORT_MEM	ソートヒープ数
VACUUM_MEM	VACUUM 用バッファ数
CHECKPOINT_SEGMENTS	自動チェックポイント間隔（セグメント指定）
CHECKPOINT_TIMEOUT	自動チェックポイント間隔（時間指定）
WAL_BUFFERS	ログ用バッファ数

4.3 I/O チューニング

DBMS においては，データはすべて物理媒体に格納される．よってデータへの新規アクセス時は，必ず処理速度の遅い装置への I/O が必要になってくる．これはデータベース規模が大きくなるにつれてさらに増加する．また，データベースへのアクセスはランダムアクセスになることが多く，ディスクが本来持つシーケンシャルアクセス時の性能よりも大きく低下する．PostgreSQL においては，データは 1 ディレクトリ内に集められており，別々のディスクに格納されたテーブル同士で，互いの情報にアクセスする方法はない．

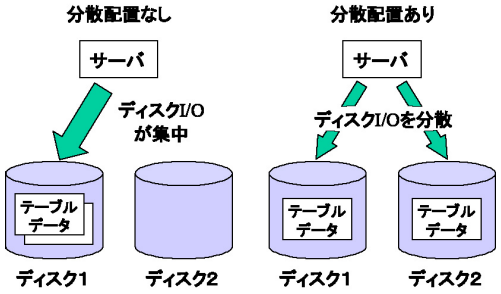

図 3 テーブルデータの分散配置

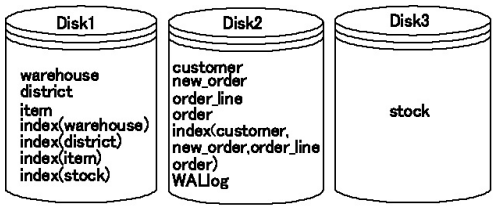

図 4 テーブル配置レイアウト

そこで本研究では，シンボリックリンクを用いたデータ分散を行った．これは DBMS のシステムカタログに，予めデータの path 情報を保存しておくことにより，テーブル作成時のファイル格納位置を指定するものである．これにより，本来 1 つのデータベースを複数のディスクへ分散して格納することで，システムが本来一ヵ所で受け取るべき I/O 負荷を軽減できる．

今回のチューニングでは，3 台の RAID ディスクを用いてテーブルデータとログデータの分散配置を行った．提案方式での実装に関し，ログの分散は問題なかったものの，テーブルの分散では一部手動で設定する必要があった．今後はリンク方式や事前処理についての検討も必要だが，場合によってはテーブル生成命令自体の拡張も検討対象になると思われる．

レイアウトを図 4 に示す．チューニングの方針としては，シーケンシャルアクセスが多いデータとアクセス頻度が高いデータを分散し，それぞれのアクセスがおおよそ同じ頻度になるように調整する．本来，ログのようなシーケンシャルアクセスがメインであるデータは，ランダムアクセスが多い他のデータとは別のディスクへ配置するのが最適である．しかし，予備テストではログの書き込み頻度がそれほど多くなかったため，比較的アクセスの少ないテーブルなどと同じディスクへ配置した．

表 4　測定環境

OS	RedHatLinux 8.0（kernel: 2.4.20-8）
DBMS	PostgreSQL 7.3.0
Server	HA8000(CPU/Pentium　xeon 500MHz*4, Memory/512MB)
Raid Disk	NTC SendbackRaid 6HDD level5
Client	CPU/K-6　400MHz, Memory/384MB)
Network	100BASE-TX LAN

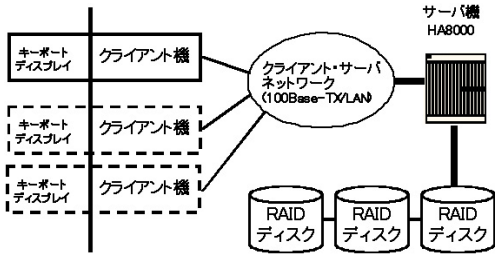

図 5　実験環境図

4.4 評　　価

4.4.1 環境と評価方法

本測定では，倉庫数の増加に伴う TPC-C 性能基準値（tpmC）の変化を調べる．実験環境を表 4，図 5 に示す．TPC-C ベンチマークでは 1 倉庫あたり約 210MB の領域を必要とする．また，倉庫ごとに決められた数の端末が要求を送る仕様になっている．よって，倉庫数を増加させることにより，データベース規模の拡大とともに I/O 負荷が増加する．また性能の限界については，応答時間制約を満足できなくなった時点で測定終了とする．

ここでは，I/O チューニングによるパフォーマンスの変化に注目するため，他のチューニング項目は全て同じ条件とした．以上の評価により，I/O におけるデータ分散の有効性を検証する．

4.4.2 測 定 結 果

表 5　New-Order トランザクションの 90%応答時間 [sec]

倉庫数	4	6	8	15	16
分散なし	1.440	4.983	8.250	—-	—-
分散あり	0.735	0.783	1.133	3.793	5.910

測定結果について New-Order トランザクションの 90%応答時間を表 5 に，MQTh 値を図 6 に示す．これらから，データ分散を行った方が高い性能を発揮していることがわかる．

応答時間を見ると，仕様（5sec 以内）を満たす最大の倉庫数は分散なしが 6 倉庫なのに対し，分散を行った場合は 15 倉庫まで増加した．また，同倉庫数でも

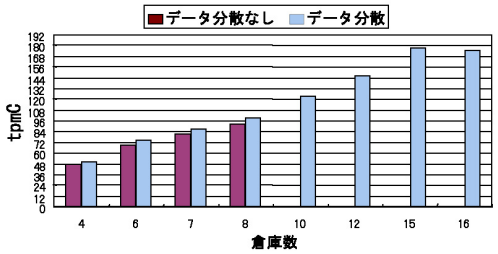

図 6　MQTh 値 測定結果

分散の有り無しによって，性能に差があることが読み取れる．原因として分散なしの場合には，ランダムアクセスの集中によって I/O 処理に関した待ち時間が増加してしまったものと考えられる．

データ分散を行ったことで，応答時間・性能基準の両面で向上が確認できたことから，問題の I/O 負荷における分散処理の有効性は非常に大きいと言える．

表 6　各デバイスに対する平均 I/O セクタ数（8 倉庫時）

デバイス	分散なし [sect/s]	分散あり [sect/s]
system	171.44	112.62
disc1	—-	1013.74
disc2	—-	1848.08
disc3	3992.47	1385.90

また，今回のチューニングでどれ程の I/O 分散がなされたのかを調査した．結果を表 6 に示す．表 6 を見ると，8 倉庫時には分散なしで 1 つのディスクに 4000 セクタが集中している．これを分散することで 1 ディスクへの負荷を 1600 セクタ前後に抑えることができた．しかし，各ディスクを見るとやや disc2 の負荷が大きい．これは，分散時にログの配置によって予想できていたことではあるが，理想的には平均して分散を行うことが望ましい．これを改善するため，より詳細なテーブル内部の分割も考慮すべき点になると言える．

5. トランザクション処理モニタの導入

これまでの実験で用いたシステムは，図 7 のような二層構造であった．この構造では，ネットワーク上のクライアントアプリケーションはデータベースサーバと直接通信ソケットを確立し，データアクセスを行う．すると，DB サーバは多数のクライアントからの接続要求に対し，それだけ接続を確立していかなければならない．TPC-C ベンチマークでは，倉庫数が増えるのに比例してサーバ・クライアント間の接続数が増える．PostgreSQL では，接続要求がある度に fork() で子プロセスを作成し処理を行っているため，プロセス数が

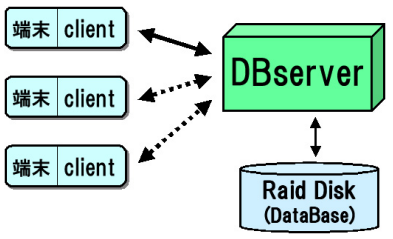

図 7　既存構造

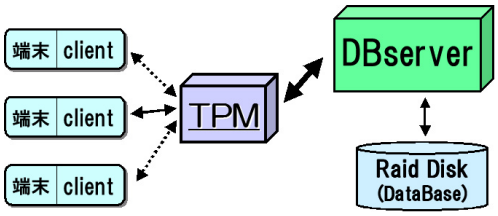

図 8　設計三層構造

過剰になり，データベースサーバに余計な負荷がかかる．このため，既存構造ではサーバが処理できる要求数に至る以前に，過剰なプロセス増加により処理速度が低下しているものと考えられる．本章では，以上の問題を解決するため図 8 のように**トランザクション処理モニタ（Transaction Processing Monitor, TPM）**を用いた新たなシステムを導入する．

5.1　三層データベースシステム

導入するデータベースのシステム構造は三層データベース構造と呼ばれる．これは，既存のサーバ・クライアント構造に中間層としてミドルウエア層を加えた構造となっている．この構造の中間層での一般的な役割は以下のようなものがある．

- **接続数の調節**
  接続数の調節は，DB サーバのパフォーマンスを最大限に活用できるように最適な接続環境を実現する．これにより，サーバは接続数増加に伴う負荷の増大を抑え，十分な DB 処理能力を発揮できる．
- **複数サーバシステムでの処理の分散**
  サーバの処理の分散は，クライアントからの要求を処理するサーバに均等に割り振ることで複数サーバの分散処理を実現する．これにより，処理速度の向上が期待できる．
- **トランザクション管理**
  トランザクション管理は，クライアントから送られてきた要求を管理し，トランザクション単位でのデータベース処理を ACID 特性の保持とともに実現する．これにより，信頼性の高い DBMS を構築することができる．

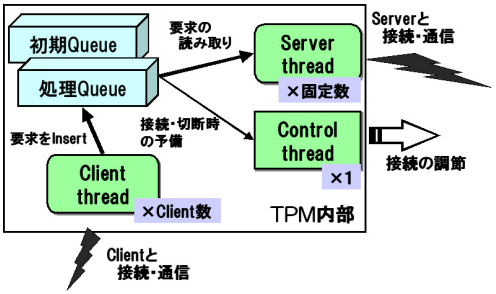

図 9　TPM 設計

5.2　TPM 設計

本研究で設計する TPM の主要な役割は，クライアント・サーバ間の過剰通信による負荷の軽減にある．そこで，今回はクライアントとサーバの接続数を別々に設定できるようにした．これで DB サーバ側のプロセス数を軽減し，効率良い DB 処理を行える接続環境を構築する．また，スレッドをベースにした設計にすることで，多接続環境でのメモリ負荷を考慮している．

設計の概要を図 9 に示す．TPM 内部は大きく分けてスレッドとキューの 2 つの部分で構成される．スレッドはサーバとクライアントとの通信管理を行い，問題となっていた接続数の軽減を実現する．キューはクライアントからの要求のキューイングに使用し，全スレッドの共有データとして位置づけた．各部については以降，通信処理と接続数問題の解決，キュー利用の三部に分けて説明する．

5.2.1　スレッドを用いた通信処理

スレッドはその処理から考慮して大きく 3 つに分かれている．それは，クライアントからのリクエストの受付を行うクライアント（Clt）スレッド，DB サーバとの通信を行うサーバ（Srv）スレッド，接続を管理するコントロール（Ctrl）スレッドである．それぞれのスレッドはプログラムのメインルーチンから生成され，並列に実行される．これにより 1 プロセスで，多数のクライアントと DB サーバとの接続・通信管理を行える．

基本的な通信の流れは，クライアントからの要求を Clt スレッドが受け付け，Srv スレッドがその要求をサーバへ送信し，応答を受けるようになっている．しかし，クライアントからの要求と DB サーバからの応答の送信は，必ずしも 1 対 1 ではない．よって単にクライアントから受信した内容をサーバに送信し，サーバからの応答を受信してクライアントへ送信するという一連のループとはできない．この問題を解決するため，受信ごとにシグナルマスクのチェックを行う．通信時にはそれぞれのエンドポイントを監視し，読み込

み可能なシグナルを判断した上で，メッセージの送受信を行っている．これにより，変則的な通信制御を可能にした．

5.2.2 接続数問題の解決

サーバへの接続数軽減を実現するにあたって，最大の問題はサーバとクライアントの１対１通信である．つまり，通常の通信ではクライアント数がサーバへの設定接続数を超えてしまうと，それ以上の接続は行われない．よって，残るクライアントは通信中のクライアントが処理を終了し接続を切断するまで待機しなければならない．

そこで，Ctrl スレッドを作成し余ったクライアントの接続処理を実現する．Ctrl スレッド処理の手順を以下に示す．

(1) クライアント数が設定接続数を超えるまで待機．
(2) 余った接続要求に対し 1 つの Clt スレッドが内容を受け取り，内容を Ctrl スレッドへ渡す．
(3) Ctrl スレッドは過去の接続処理に利用したサーバ応答を利用し，クライアントとの接続を完了する．

このようにして，すべてのクライアントが接続を完了し，処理状態へ移行することができる．

5.2.3 トランザクションを考慮したリクエスト管理

今回は，リクエストのキューイングを用いてクライアント処理の切り替えを行った．作成したキューの構造は一般的な FIFO 方式であり，両端に TOP と BOTTOM という領域を確保し，データをこの間で挿入・削除していく設計である．また，キューは状態の種類よって初期状態キューと処理状態キューの 2 つに分けた．これは，要求を受けたクライアントの状態が接続段階であるか，DB 処理段階であるかを区別するためである．データの中には，そのデータの直前と直後のデータへのポインタ，クライアントからの要求内容とクライアントの接続シグナル番号，その他スレッド番号などが入るようになっている．

クライアントからリクエストがあると，Clt スレッドは該当する状態キューをロックし必要な内容をキューの末尾に挿入する．Srv スレッドはキューの先頭データを取り出し，その内容を DB サーバへ送信して DB 処理を開始させる．この時，設計においてどの時点で要求を送るクライアントを切り替えるかという問題は，TPM 全体の性能に大きく影響する．１クライアントの処理を数トランザクション連続して行うと残ったクライアントのレスポンスは非常に長くなり，並列処理という目的に大きく弊害をもたらす．そこで考慮すべき概念がトランザクションという単位である．トランザクションは複数の要求で構成されており，これらは一連の処理として行われなければならない．これらの考えから，Srv スレッドとサーバとの交信は，トランザクションごとの切り替えに設計した．Srv スレッドはトランザクションの終了を検知した時点で，クライアントからのリクエスト受付を Clt スレッドに戻し，つぎのリクエストをキューから取り出す．これにより，効率的な処理の切り替えを実現できる．

5.3 評　　価

5.3.1 環境と評価方法

表 7 測定環境

OS	RedHatLinux 9.0（kernel: 2.4.20-8）
DBMS	PostgreSQL 7.3.4
Server	HA8000(CPU/Pentium xeon 500MHz*4, Memory/1GB)
Raid Disk	NTC SendbackRaid(63.6GB) 6HDD level5
Client	FLORA370(CPU/Pentium 400MHz, Memory/256MB)
TPM	CPU/Seleron 1.7GHz, Memory/512MB
Network	100BASE-TX LAN

測定環境を表 7 に示す．本測定では，接続数の増加に伴う TPC-C ベンチマーク性能基準値（tpmC）の変化を調べる．TPC-C では DB サーバとの交信で，1 倉庫ごとに 10 接続を行う．実験では測定ごとに倉庫数を増やし，データベース規模の拡大とともに接続数を増加させることで DB サーバへの負荷を上げていく．また，接続数を増加させるためにクライアント数を増やす方法も用いる．TPM を用いた構造では，2 倉庫以上でのサーバへの接続数を 20 に固定し，サーバのプロセス数を適当数に保つ．これにより，サーバへの過剰負荷を防ぎ効率よい DB 処理を実現する．

これらの測定方法で，接続数の増加に伴う tpmC 値の変化について，二層構造と TPM を用いた構造をそれぞれ測定する．そして 2 つの構造について両者を比較・評価し，TPM の有効性を検証する．

5.3.2 測 定 結 果

結果を図 10，11 に示す．これらより，既存の二層構造に比べ TPM を用いた構造の方が高い性能を発揮しているのが分かる．

図 10 の倉庫あたりの tpmC 値では，10 倉庫辺りまでは変わらないものの，それ以後は倉庫数が増えるにしたがって両構造の測定結果に差が出ている．二層構造では，約 15 倉庫程度で DB サーバのプロセス数が過剰になりすぎてしまうため，過負荷の状態になっている．すると，1 トランザクションの処理にかかる時

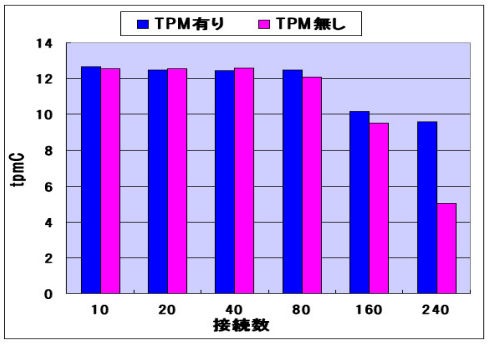

図 10 倉庫あたり tpmC の変化

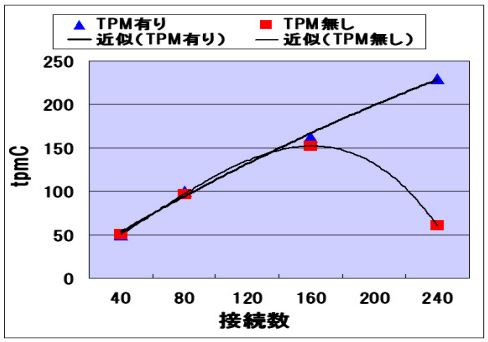

図 11 MQTh 値の変化

間が今までよりも長くなってしまうため，全体として tpmC 値が下がる．対して TPM を用いた場合は，DB サーバへの接続数は少倉庫数時と変わらず一定であるため，DB サーバは 1 トランザクションを効率よく処理できる．しかし倉庫数の増加に伴い，DB サーバの一定時間あたりの処理数が限界に達すると，TPM の内部キューに要求が溜まりだし，倉庫あたりの tpmC 値は少なからず減少傾向となる．制約条件の点から見ると，TPM を使用しない既存の二層構造においては，仕様 tpmC 値（9〜12.86）を満たすのは 16 倉庫までであった．TPM を用いた構造は実質 240 端末（24 倉庫分に相当）での測定において仕様を満している．

また，MQTh 値において図 11 の結果をみると，測定範囲内で既存構造が対数的な増加なのに対し，TPM を用いた構造はまだ直線的に増加している．これは，TPM 構造が測定範囲の倉庫数で十分な処理速度を維持していることを示している．既存構造では，約 16 倉庫で直線的な増加傾向を維持できなくなった．それに比べ，TPM を利用した構造は測定範囲の接続数・倉庫数で仕様 tpmC 値に準拠し，さらに高負荷の測定で結果が期待できる水準にある．

また，二層構造での過剰プロセス状態のメモリ負荷を調査するため，測定中のスワップ発生状況を調査した．結果を表 8 に示す．表 8 を見ると，それほどの接続がなされていない 8 倉庫では，両構造ともスワップの使用は少ない．しかし，tpmC 値に差が出始めた 16 倉庫になると，三層構造ではある程度の数値に抑えているものの，二層構造では非常に大きな値となっていた．スワップ処理によって発生したディスクへの余計なアクセスは当然ボトルネックとなるため，二層構造では性能低下は避けられない．対して三層構造でのスワップ縮小は，TPM の接続数の調節によってメモリの利用効率が大きく改善されたことを示している．

表 8 測定中の swap 領域使用状況

	8 倉庫時 [MB]	16 倉庫時 [MB]
TPM 無し	25.132	181.504
TPM 有り	4.040	20.137

これらの議論から，TPM を用いた構造は既存構造に比べ，DB サーバの性能をより発揮させていることが実証された．

6. おわりに

本研究では，オープンソースの DBMS である PostgreSQL に焦点を当て，性能評価のためのベンチマークツールを作成した．そして性能向上についてメモリチューニングや I/O チューニングを検討し，実際にベンチマークを用いてそれらの有効性を確認できた．また，問題となっていた接続数の問題について TPM を導入することで大きな改善を得ることができた．

今後は DBMS 独自のキャッシュ管理や，データ分散のより詳細な配置方式の実装・評価を行っていく考えである．また TPM については，キューイング方式の改良や汎用性への設計・実装を予定している．これらによって，DBMS の効率性・信頼性がより向上していくことを期待する．

参考文献

1) http://www.tpc.org
2) http://www.postgresql.org
3) http://www.postgresql.jp
4) Bruce Momjian, PostgreSQL performance tuning, Linux Journal Volume 2001, Issue 88 (August 2001) table of contents Page: 3, 2001
5) Rolf Herzog, PostgreSQL–The Linux of Databases, Linux Journal Volume 1998, Issue 46es (February 1998) table of contents Article No.1, 1998