

2004 年度  
オープンソースソフトウェア活用基盤整備事業

---

「OSS 性能・信頼性評価 / 障害解析ツール開発」

DB 層の評価

---

独立行政法人 情報処理推進機構

## 商標表記

- MySQL は MySQLAB の登録商標です。
- Oracle は、Oracle Corporation の登録商標です
- Linux は、Linus Torvalds の米国およびその他の国における登録商標あるいは商標です。
- SUSE は米国 Novell, Inc.の事業部である SUSE LINUX AG.の登録商標です。
- Red Hat は、米国およびその他の国で Red Hat, Inc. の登録商標若しくは商標です。
- MIRACLE LINUX は、ミラクル・リナックス株式会社が使用許諾を受けている登録商標です。
- その他記載の会社名、製品名は、それぞれの会社の商号、商標もしくは登録商標です。

1	DB 層の評価	1-1
1.1	目的	1-1
1.2	プロジェクトのスコープ	1-1
1.3	概要	1-2
1.3.1	評価内容	1-2
1.3.2	評価環境	1-2
1.3.3	オープンソース RDBMS について	1-3
1.3.4	性能評価ツール DBT について	1-3
2	全体の考察	2-1
2.1	評価数値の取り扱い	2-1
2.2	DBT-1(TPC-W 相当の Web ベースシステム環境)を利用した共通手順の作成	2-2
2.2.1	スコープ拡大のための DBT-1 の改変	2-2
2.2.2	DBT-1 による PostgreSQL、MaxDB の性能評価	2-3
2.2.3	Linux カーネルの違いによる性能評価	2-5
2.2.4	DBT-1 の課題	2-6
2.3	DBT-3(TPC-H 相当の DSS 環境)を利用した共通手順の作成	2-7
2.3.1	DBT-3 による PostgreSQL の性能評価	2-7
2.3.2	DBT-3 の評価	2-10
2.4	大規模 DB 性能(運用性)の評価手順の作成および評価	2-11
2.4.1	実際の評価結果	2-12
2.4.2	大規模 DB 性能(運用性)の評価	2-14
2.5	DBT-1 のスコープ拡大のための MySQL 向け改変調査作業	2-15
2.5.1	DBT-1 の問題点	2-15
2.5.2	対応方針	2-15
2.5.3	実装方式	2-15
2.5.4	見積り	2-16
2.6	各ソース・パッチ / 修正事項の概要	2-16
2.6.1	DBT-1	2-16
2.6.2	DBT-3	2-16
2.6.3	大規模 DB 性能(運用性)の評価	2-16
3	DBT-1 による MaxDB の評価	3-1
3.1	概要	3-1
3.2	MaxDB へのポーティング	3-1
3.2.1	ポーティング方針	3-1
3.2.2	主な修正点	3-4
3.3	環境定義書	3-6
3.3.1	システム構成	3-6
3.3.2	Linux のインストール	3-8
3.3.3	MaxDB のインストール	3-9

3.3.4	DBT-1 用の事前環境設定 .....	3-11
3.3.5	DBT-1 のインストール .....	3-13
3.3.6	パラメータの設定 .....	3-14
3.4	評価手順書 .....	3-16
3.4.1	前提条件 .....	3-16
3.4.2	テストデータの作成 .....	3-16
3.4.3	データベースの作成 .....	3-17
3.4.4	パラメータの設定 .....	3-18
3.4.5	起動方法 .....	3-19
3.5	測定結果と考察 .....	3-22
3.5.1	測定結果 .....	3-22
3.5.2	チューニング前の結果の考察 .....	3-27
3.5.3	チューニング後の測定結果 .....	3-30
3.5.4	チューニング後の結果の考察 .....	3-33
3.5.5	インタラクションにおける性能の違い .....	3-39
3.5.6	計測データ .....	3-43
3.6	まとめ .....	3-47
4	DBT-1 による PostgreSQL の評価 .....	4-1
4.1	概要 .....	4-1
4.2	環境定義 .....	4-1
4.2.1	システム構成 .....	4-1
4.2.2	インストール .....	4-2
4.2.3	パラメータの設定 .....	4-6
4.3	評価手順 .....	4-8
4.3.1	環境変数の設定 .....	4-8
4.3.2	データベースの作成 .....	4-8
4.3.3	DBT-1 の実行 .....	4-11
4.4	実行及び考察 .....	4-11
4.4.1	実行結果の収集 .....	4-11
4.4.2	測定の観点 .....	4-12
4.4.3	測定結果と考察 .....	4-13
4.5	まとめ .....	4-28
5	DBT-1 の商用 RDBMS への適用 .....	5-1
5.1	概要 .....	5-1
5.2	商用 RDBMS へのポーティング .....	5-1
5.2.1	ポーティング方針 .....	5-1
5.2.2	修正概要 .....	5-2
5.2.3	修正範囲 .....	5-2
5.2.4	PL/PGSQL - Oracle PL/SQL 移植ガイドライン .....	5-5
5.3	環境定義 .....	5-11

5.3.1	システム構成.....	5-11
5.3.2	インストール.....	5-12
5.3.3	パラメータの設定.....	5-14
5.4	評価手順.....	5-16
5.4.1	データベースの作成.....	5-16
5.4.2	DBT-1 パラメータの設定.....	5-17
5.4.3	DBT-1 の実行.....	5-19
5.5	実行.....	5-20
5.5.1	実行結果の収集.....	5-20
5.6	まとめ.....	5-20
6	DBT-3 による PostgreSQL の評価.....	6-1
6.1	概要.....	6-1
6.1.1	OSDL-DBT-3 の概要.....	6-1
6.1.2	テーブルの概要.....	6-4
6.1.3	クエリーの概要.....	6-8
6.2	環境定義.....	6-10
6.2.1	システム構成.....	6-10
6.2.2	インストール.....	6-11
6.2.3	パラメータの設定.....	6-18
6.3	評価手順.....	6-19
6.3.1	前提条件.....	6-19
6.3.2	評価環境.....	6-19
6.3.3	レポートの作成.....	6-24
6.3.4	レポートの内容.....	6-27
6.4	測定結果と考察.....	6-36
6.4.1	性能に影響を与えうるファクター.....	6-36
6.4.2	測定結果と考察.....	6-39
6.4.3	まとめ.....	6-78
7	大規模 DB 性能 (運用性) の評価.....	7-1
7.1	概要.....	7-1
7.1.1	テーブル構成.....	7-2
7.2	環境定義.....	7-4
7.2.1	システム構成.....	7-4
7.2.2	インストール.....	7-5
7.2.3	パラメータの設定.....	7-9
7.3	評価手順.....	7-10
7.3.1	前提条件.....	7-10
7.3.2	評価環境.....	7-10
7.3.3	大量データのロード.....	7-11
7.3.4	バックアップとリカバリ.....	7-12

7.3.5	インデックス再構築	7-13
7.3.6	クラッシュリカバリ (PostgreSQL8.0のみ)	7-14
7.4	測定結果と考察	7-18
7.4.1	測定結果と考察	7-19
7.4.2	まとめ	7-39
8	DBT-1のMySQL用改変基本設計	8-1
8.1	DBT-1 Architecture 概要	8-1
8.1.1	OSDL-DBT 1 (Open Source Development Labs Database Test1)	8-1
8.1.2	OSDL-DBT-1の概要	8-1
8.2	基本的な構造(現状)	8-3
8.2.1	dbdriverの構造	8-4
8.2.2	appServerの構造	8-5
8.3	DBT-1のトランザクション概要	8-6
8.4	DBT-1動作特性	8-7
8.4.1	負荷パターン	8-7
8.4.2	DBT-1計測タイミングの調整について	8-8
8.5	現状の問題点	8-10
8.6	設計方針	8-11
8.7	設計概要(基本設計)	8-11
8.7.1	概要	8-11
8.7.2	構造	8-14
8.7.3	改変見積り	8-17
9	付録資料一覧	9-1
9.1	DBT-3によるPostgreSQLの評価	9-1

# 1 DB 層の評価

## 1.1 目的

データベースは企業システムの中でもデータを統括的に管理するための要になる部分であり、多くの企業向けシステムでもデータベースが一般的に利用されている。

オープンソース製品においても OS レベルからミドルウェアレベルに注目が移りつつあり、特にオープンソース RDBMS への関心が高まっている。

こうした中でシステムの開発局面において、RDBMS を選択する場合の判断基準としての性能評価手法のニーズが高まっており、オープンソース RDBMS の適正な利用範囲を見極める上で、標準的な評価手順を持つ意義は大きいと考えられる。

これに応えるために本評価では、システム開発において必要となる RDBMS の性能測定のための環境構築・評価手順を策定し、RDBMS のサイジングやチューニングにおける標準的な手順として利用できることを検証する。

また、この手順を利用して主要なオープンソース RDBMS に関する性能測定を行い、参考値として結果を示す。

なお、本評価では性能評価手順を作成し、この手順を利用する上での留意点を示す事を目的としており、個々のハードウェア、ソフトウェアの個別製品の性能比較を行う事を目的としていない。

## 1.2 プロジェクトのスコープ

本評価では、オープンソース RDBMS として重要と思われる製品と、その性能評価ツールとして有用と思われるものを取り上げた。

- ・ 手順策定の対象 RDBMS として、オープンソース RDBMS の PostgreSQL 及び MySQL(MaxDB)と、対比として商用 RDBMS の Oracle9i を選択した。
- ・ 評価するアプリケーションモデルとしては、Web トランザクション系のアプリケーションと、意思決定支援システム系のアプリケーションを対象とした。

## 1.3 概要

### 1.3.1 評価内容

本評価では以下の4点を評価項目として考えた。

- (1) Webシステムのためのデータベース性能評価の共通手順作成および評価
  - ・ DBT-1(TPC-W を模した Web ベースシステム環境)を利用した共通手順の作成
  - ・ DBT-1 + PostgreSQL、MySQL(MaxDB)での性能評価
  - ・ Linux カーネルの違いによる性能評価
  
- (2) DSS (意志決定支援システム) 環境のためのデータベース性能評価の共通手順作成および評価
  - ・ DBT-3(TPC-H を模した DSS 環境)を利用した共通手順の作成
  - ・ DBT-3 + PostgreSQL での性能評価
  
- (3) 大規模データベース環境における性能検証
  - ・ バックアップとリカバリに要する時間の性能評価
  - ・ 大容量データのロードに要する時間の性能評価
  - ・ インデックスの再構築に要する時間の性能評価
  - ・ PostgreSQL 8.0 の Point In Time Recovery (PITR)を使用し、クラッシュリカバリに要する時間の性能評価
  
- (4) DBT-1 のスコープ拡大のための Oracle 向け改変作業及び MySQL 向け改変調査作業  
現状 SAP DB 及び PostgreSQL のストアードプロシージャで実装されている DBT-1 を、  
広範囲の評価モデルとして利用できるようにするために、Oracle9i への移植と MySQL5.0  
の改変のための調査作業を行う。
  - ・ 既存ストアードプロシージャの事前評価
  - ・ DBT-1 改変調査作業(Oracle9i、MySQL 5.0)及び 改変作業(Oracle9i)

### 1.3.2 評価環境

本評価には、以下のソフトウェアを利用した。

- (1) オペレーティングシステム
  - ・ SUSE LINUX Enterprise Server 9
  - ・ MIRACLE LINUX V3.0 – Asianux Inside
  - ・ Red Hat Enterprise Linux AS 3  
カーネルバージョンは各章を参照

- (2) データベース
  - ・MaxDB 7.5.0.16
  - ・PostgreSQL 7.4.6、PostgreSQL 8.0.0beta5
  - ・Oracle9i
- (3) 性能評価ツール：OSDL-DBT-1 及び DBT-3  
ハードウェア環境は、各評価環境によって異なる。

### 1.3.3 オープンソース RDBMS について

本評価ではオープンソース RDBMS として次の 2 つを採択した。一般的に国内で入手可能で、利用実績が多く、普及していると判断できるものを採択した。

- (1) PostgreSQL <http://www.postgresql.jp/>  
日本で最も普及しているオープンソース RDBMS であり、商用版もリリースされている。国内の主要ベンダがサポートしており、日本国内の利用実績も多い。
- (2) MySQL(MaxDB) <http://www.mysql.com/>  
世界で最も普及しているオープンソース RDBMS。MySQL には数種類のデータベースエンジンがあるが MaxDB もそのひとつ。2003 年に MySQL 社と SAP 社がクロスセンシングを行い MaxDB(旧名 SAP DB)としてリリースしたもの。  
なお、今回の DBT ベンチマークツールは開発当初は SAP DB を利用している。

### 1.3.4 性能評価ツール DBT について

#### 1.3.4.1 経緯

本評価では OSDL が提供している DBT を主なベンチマークツールとして利用する。  
ツールの選定に関して、以下の条件に着目し、利用可能と思われるツールの候補(DBT、ecperf、TPC シリーズ、等)を検討した。

- (1) DB ベンチマークツールとしての機能(実環境に近い性能評価ツールであること)
- (2) 環境構築の手軽さ(低コスト、環境構築に手間がかからないこと)
- (3) 利用時の制約(利用時の制限が少ないもの、コストがかからないもの)
- (4) 結果の公開についての制約(公開の制限が少ないもの)
- (5) 汎用性(オープンソース RDBMS への対応、多様な規模で利用できること)

DBT は TPC の簡易版性能評価ツールとしてオープンソースで提供されており、無償で利用でき、結果の公開制限も少なく、本プロジェクトで利用するものとして妥当と判断し採択した。

#### 1.3.4.2 オープンソース・性能評価ツール DBT の概要

DBT シリーズは表 1.3-1 に示す 3 つのベンチマークツールから構成されており、本評価ではこの中から DBT-1 と DBT-3 を利用している。

表 1.3-1 DBT シリーズ概要

OSDL-DBT-1	TPC-W の簡易版データベースベンチマークツール。 Web ベースのトランザクション・パフォーマンステスト。DBT-1 は、オンライン書店におけるユーザのアクティビティ(商品の検索、ショッピングカート処理、購入手続きなど)をシミュレートする。実行結果には、1 秒当たりのトランザクション数(BT /秒)、CPU の使用状況、I/O アクティビティおよびメモリの使用状況が含まれる。
OSDL-DBT-2	TPC-C の簡易版データベースベンチマークツール。 OLTP トランザクション・パフォーマンステスト。DBT-2 は、複数の作業者が 1 つのデータベースへアクセスし、顧客情報を更新し、部品の在庫を確認する部品の卸売業者をシミュレートする。実行結果には、1 秒当たりのトランザクション数、CPU の使用状況、I/O アクティビティおよびメモリの使用状況が含まれる。
OSDL-DBT-3	TPC-H の簡易版データベースベンチマークツール。 意思決定支援のためのワークロードを実行しパフォーマンスを測定する。DBT-3 は、業務用の特別なクエリや並行動作するデータ更新処理のスイートで構成される。

BT(Bogo Transaction) : 擬似的なトランザクションを表す。DBT-1 で実装される各トランザクションの処理数を指す。

URL OSDL Japan <http://www.osdl.jp/>

DBT [http://www.osdl.jp/lab\\_activities/kernel\\_testing/osdl\\_database\\_test\\_suite/](http://www.osdl.jp/lab_activities/kernel_testing/osdl_database_test_suite/)

## 2 全体の考察

この章では、3章以降に記述している各項目別の評価結果のサマリを述べる。

今回実施したDB層の評価項目は、次の通りである。

- (1) Web システムのための RDBMS 性能評価の共通手順作成および評価( 2.2)
- (2) DSS 環境のための RDBMS 性能評価の共通手順作成および評価( 2.3)
- (3) 大規模データベース環境における性能検証( 2.4)
- (4) DBT-1 のスコープ拡大のための Oracle 向け改変作業( 2.2.1.2)及び MySQL 向け改変調査作業( 2.5)

なお、個別の詳細な結果・考察は、該当する章を参照の事。

### 2.1 評価数値の取り扱い

本評価における測定環境(例えば PC サーバのスペック)は RDBMS、章ごとに異なるため単純に数値上の比較をしても意味のある結論とはならない。従って、3章以降に述べる定量的な評価結果については、同一環境(H/W、RDBMS、OS 等)にける数値比較は有用であるが、3章と4章の比較の様に異なる環境間の比較を数値のみによって単純に行うべきではない。

## 2.2 DBT-1(TPC-W 相当の Web ベースシステム環境)を利用した共通手

### 順の作成

#### 2.2.1 スコープ拡大のための DBT-1 の改変

##### 2.2.1.1 MaxDB 向けの改変作業

元々DBT-1 は MaxDB の前身である SAP DB 向けに作成されていたが、SAP DB7.4 以降、メンテナンスがされておらず、最新版である MaxDB7.5 では動作しなかった。当初は大きな変更無しに評価を行う予定であったが、今回、次の点について改変作業を行った。

- (1) MaxDB コマンドの修正
- (2) スレッド生成ロジック修正
- (3) データ生成モジュールの修正

これにより、最新の MaxDB での評価が可能となった。また、スレッド生成数を大幅に増加したことにより、高負荷状態での評価が可能となった。

##### 2.2.1.2 Oracle 向け改変作業

PostgreSQL 用の DBT-1 から Oracle9i 用に改変を行った。改変作業は次の点とおり。

- (1) PostgreSQL 用ストアードプロシージャの PL/SQL への変換
- (2) DB 接続部分の修正
- (3) 環境作成・実行のスキプトの修正

Oracle 環境で標準的な DBT-1 評価手順を確立することにより、商用 RDBMS とオープンソース RDBMS との性能比較が可能になった。また、改変作業において、PostgreSQL の PL/pgSQL を Oracle の PL/SQL へのポータリングノウハウを集積した。

## 2.2.2 DBT-1 による PostgreSQL、MaxDB の性能評価

本評価では、2つのオープンソース RDBMS(PostgreSQL、MaxDB)を利用し、3つの Linux ディストリビューション(SUSE、Miracle、RedHat)の環境で DBT-1 を動作させ測定し評価した。

以下に、各評価のサマリを述べる。

### 2.2.2.1 負荷性能の評価

DBT-1 では BT / 秒(BT とは BogoTransaction : DBT-1 が測定する擬似トランザクション数 詳細は 8 章参照)によって性能を表す。これを基に各環境での性能を考察する。

#### (1)チューニング前環境とチューニング後環境の違い

図 2.2-1 より、チューニング前(Linux、RDBMS とともに標準のインストールをしただけの状態でチューニング作業を行わない環境)では、仮想ユーザ数の増加による(DBT-1 の同時接続ユーザ数)負荷の増大につれて、一定の値までリニアに伸び、ピークを迎えた後、定常状態となる。 詳細は個々の評価結果を参照。

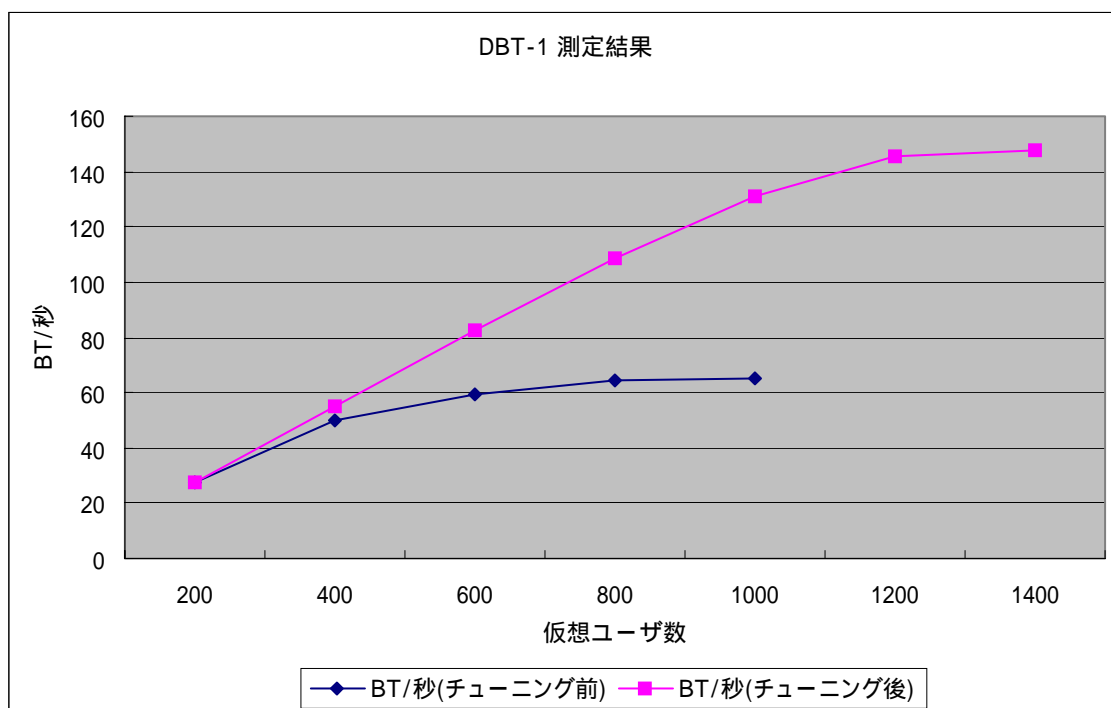


図 2.2-1 DBT-1 チューニング前・後の比較 (SUSE + MaxDB)

BT / 秒の上限は、RDBMS、ディストリビューションによって差がある。  
また、チューニング後環境では、概ねどの環境でも 2 倍程度の性能向上が見られた。  
従って適切なチューニングを行う事が非常に重要といえる。

ただし、チューニング方法は RDBMS によって異なったアプローチが必要であり、適用すべき環境の特性と利用する RDBMS によって適宜選択する必要がある。

## (2) 連続稼働時の性能劣化

duration(個々の仮想ユーザの操作時間)を長く取ったり、または連続稼働(毎回テスト毎に再起動をせず、複数回のテストを連続して実行する)させた場合に、徐々に BT / 秒が劣化するケースがあった。

今回は詳細な分析までは出来ていないが、ディストリビューションによって、連続稼働時の劣化する度合いが異なる結果となっている。

これら連続稼働時の性能劣化の発生は、実システムへの適用で問題の起こる可能性が高いことを示唆している。特に、無停止運転が必要なシステム等では重要な考慮点の一つと考えられる。

なお、DBT-1 の利用上の注意として、duration 値は測定目的によって適切に設定する必要がある。高負荷時の耐久性能を知りたい場合には数十時間を設定する必要があり、瞬間的な性能を測定する場合であれば、数千秒で計測可能である。

## (3) データサイズによる違い

DBT-1 のデフォルトのデータサイズは約 3GB(テキストファイル換算、商品数 10000、登録顧客数 1000)であり、これを基本として評価を実施した。データサイズを約 10 倍に増大させたケース(商品数・登録顧客数をそれぞれ 10 倍)では性能は大幅に劣化し、ユーザ要件によっては利用が現実的ではない RDBMS もみられた(なお、本評価では評価環境での限界値の測定まで行っていない)。

劣化の度合いは、RDBMS やチューニング内容により異なる。この劣化の要因のひとつとして、DBT-1 には途中一致の文字列比較を行う全文検索的な問い合わせが含まれていることが考えられる。

## 2.2.3 Linux カーネルの違いによる性能評価

本評価では、3種類のLinuxディストリビューション(2.6系カーネル1種類と2.4系カーネル2種類)を利用して性能評価を行った。測定結果から次の点で、カーネル2.6系がカーネル2.4系よりも性能が高く、安定的に動作することが確認できた。

### 2.2.3.1 性能

低負荷状態ではどのディストリビューションであっても差異を認められないが、高負荷時にはディストリビューションによってBT/秒の差が広がった。

全体を通して言えるのは、カーネル2.6は、カーネル2.4のLinuxディストリビューションよりも、高性能であった。

図2.2-3で示すとおり、2.6系カーネルの優位性が確認できた結果となっている。

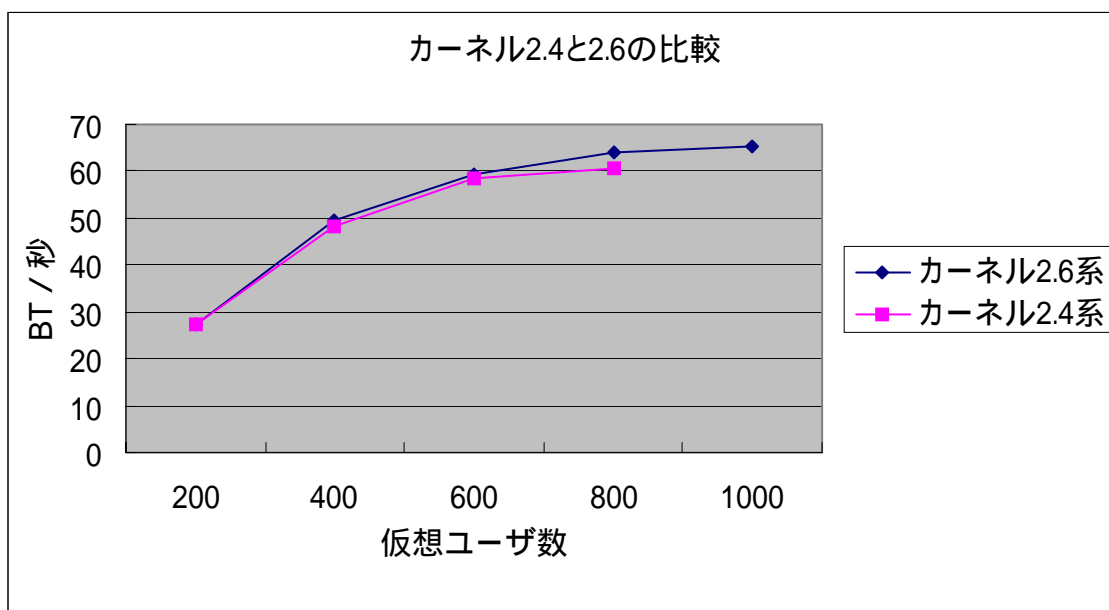


図 2.2-2 カーネル比較

カーネル2.4では仮想ユーザ数1000で測定できなかった。 2.2.3.2を参照

### 2.2.3.2 安定性

本評価では、仮想ユーザ数を増加させた高負荷時にDBT-1が正常に稼動しないケースが発生したため、次のOSパラメータを変更した。

(1) スレッド生成数の上限値

仮想ユーザ数を増加させるためにスレッド生成上限値を変更

## (2) セマフォ数の上限値(MaxDBのみ)

appServer と dbdriver の接続に必要なセマフォ制限を変更

これらはディストリビューションによって設定値が異なる。設定した値によって OS 動作が不安定になったり、ケースによっては OS がフリーズする(高負荷時で 2.4 系カーネルの特定ディストリビューションの特定バージョンのみ 対象 OS の最新のパッチを適用することで解決した。)などの現象が発生した。環境構築時に注意を要する点である。

実際のシステム構築時には DBT-1 を用いた検証でおおよその性能限界が見えると予想される。DBT-1 を用いる事でどんな性能曲線を描くのか、どこまで耐性があるかについての検証が可能になるとともに、安定的な稼働を検証するツールとしての利用が考えられる。

### 2.2.4 DBT-1 の課題

本評価で得られた結果、Web 系処理システムの評価ツールとして、測定したい Web システムの特性を考慮した上で、簡易に利用できるツールとして活用可能なことが解った。しかし、以下のような課題も散見された。

- (1) TPC-W では規定されているが、実装されていないトランザクションがある。
- (2) 測定結果のバラツキが見られる。特に仮想ユーザ数を増加させると BT / 秒の変動が大きくなる傾向が見受けられ、高負荷時での測定では 1 ショットの測定だけでなく、複数回の測定が好ましいと考えられる。
- (3) ストアドプロシージャによるビジネスロジックの実装では RDBMS に負荷が集中する傾向があるが、最近のシステム開発の場面ではこのようなストアドプロシージャ実装を行わない傾向があり DBT-1 による測定結果のみで判断すべきでない。
- (4) 今回の測定では、当初、仮想ユーザ数の増加に制限があり測定が出来なかった。これに対応する修正を行って(仮想ユーザ数を 1600 程度まで増加させた)測定を行ったが、更に高負荷の状態が必要な場合は、ひとつの dbdriver では対応できないと思われる。高性能な H/W 環境で測定をする場合に、OS のパラメータの最適化、複数の dbdriver の利用、RDBMS とベンチマークテストの分離などの措置を講ずる必要がある。

## 2.3 DBT-3(TPC-H 相当の DSS 環境)を利用した共通手順の作成

OSDL で公開されているデータベースのベンチマークツールである DBT-3 を利用した RDBMS の性能評価を行う。DBT-3 は、オープンソースとして公開されており、TPC-H の簡易版として大規模データベースにおける意志決定支援のための情報検索を行うベンチマークツールとして利用できた。

以下の点で有用であると考えられる。

- ・テストツールの利用が無償であること
- ・ソースが公開されており、処理動作が理解できること
- ・評価結果の扱いに関する制約が少なく、利用しやすいこと

これに加え、次の点を改良した。

- ・多くの不具合を修正して、PostgreSQL での利用を容易にしたこと
- ・DB 層における標準的な手順を確立したこと(テストによる差が無くなる)

これらにより、PostgreSQL を利用したシステム開発において手軽に性能評価が出来るツールとしての効果が確認できた。

今後のシステム開発において PostgreSQL の性能評価ツールとしての利用が可能であると思われる。

### 2.3.1 DBT-3 による PostgreSQL の性能評価

DBT-3 を利用した大規模データベースにおける複雑なクエリーの性能評価を、PostgreSQL7.4.6 と 8.0.0beta5 について、以下の観点から実行した。

- ・DBT-3 ワークロードの動作確認
- ・ディスク構成による違い
- ・スケールファクターによる違い
- ・チューニングによる違い

また、今回の評価期間が終了する直前に、PostgreSQL8.0.1 正式版がリリースされたので、代表的なパターンで実行したところ、8.0.0beta5 とほぼ同じ結果となった。従って、8.0.0beta5 で実行した結果についても、8.0.1 正式版と同様になると考えて問題ないと思われる。

今回の評価で分かったことについて、以下に概要を説明する。詳細は 6 章を参照のこと。

### 2.3.1.1 パワーテストの結果

パワーテストは、同時に実行するトランザクション数が1のベンチマークである。パワーテストの結果はブレが多かったので10回実行した。その結果を、図 2.3-1 に示す。

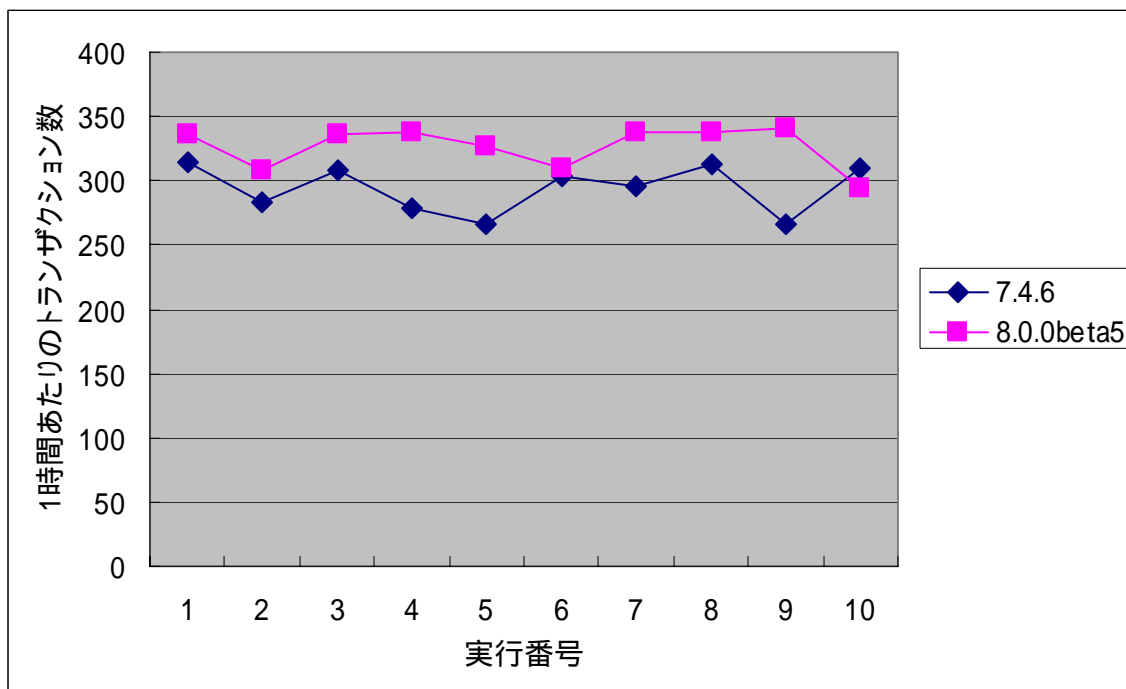


図 2.3-1 パワーテスト

パワーテストを10回実行した結果の集計を、表 2.3-1 に示す。

表 2.3-1 パワーテストの集計

	平均	最小	最大	最大 - 最小	誤差率
7.4.6	294.119	265.97	314.06	48.09	約 15.31%
8.0.0beta5	326.633	294.49	341.11	46.62	約 13.67%

平均を見ると、PostgreSQL8.0.0beta5 は 7.4.6 よりも約 10%ほど性能が良いことが分かった。

また、誤差率が高いのは PostgreSQL のオプティマイザが適切でない実行プランを選ぶ場合があるのが原因である。それを適切に選択させるには、オプティマイザの制御スイッチの指定を DBT-3 のクエリーに追記すれば良く、その方法は 6 章に記述した。

### 2.3.1.2 規模によるロード時間

スケールファクターは、DBT-3 で使用するデータベースに投入するテキストデータのサイズで、単位はGB(ギガバイト)である。スケールファクターの値を変えながら、DBT-3 のロードテストの結果を、1 スケールファクターあたりのロード時間で表したグラフを図 2.3-2 に示す。

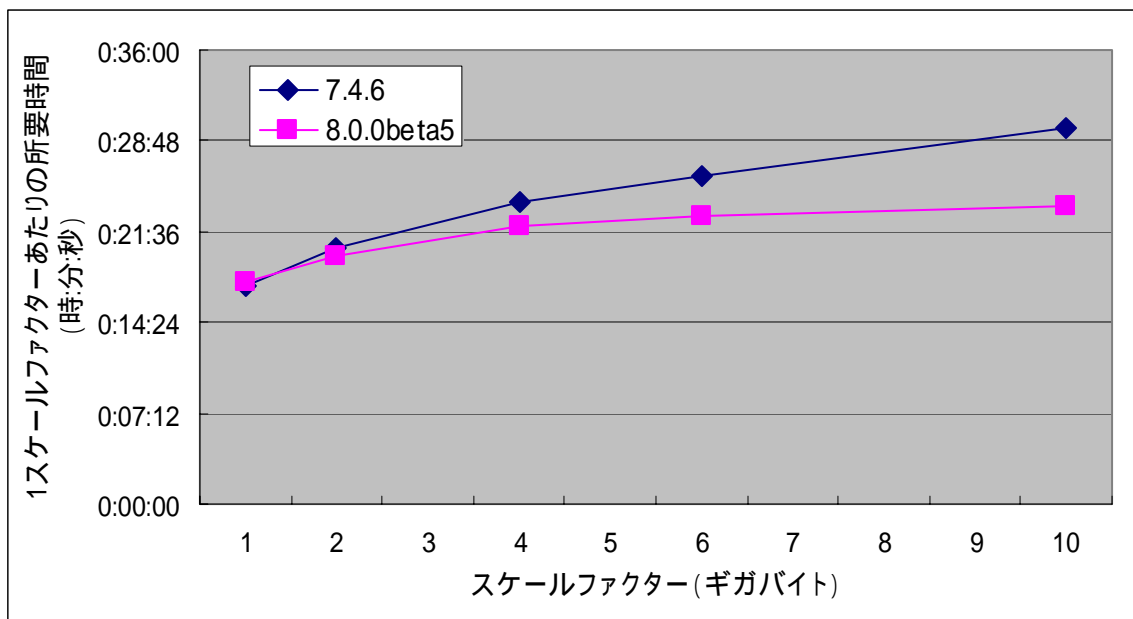


図 2.3-2 1 スケールファクターあたりのロード時間

PostgreSQL8.0.0beta5 は 7.4.6 と比べると、データの規模が大きくなればなるほど、データのロードに要する時間が短くなることが分かった。

### 2.3.1.3 その他

その他に分かったことのサマ리를、以下に述べる。

- ・ PostgreSQL8.0.0beta5 のオプティマイザは、7.4.6 より改良されていることが確認できた。DBT-3 のクエリーの中には、7.4.6 では3日待っても終わらない例があったが、8.0.0beta5 では数秒で実行できた。
- ・ PostgreSQL8.0.0beta5 のテーブルスペースは、ディスクをアンマウント/マウントすることでバッファキャッシュをクリアした場合に効果が確認できた。バッファキャッシュの効果が薄れる負荷の高い実環境や、より大規模なデータベースではテーブルスペースを利用するメリットがあると考えられる。

### 2.3.2 DBT-3 の評価

今回の評価では、PostgreSQL を利用した、DBT-3 ワークロードの測定手順を確立することが出来た。今回使用した DBT-3 バージョン 1.5 に付属するマニュアルは、バージョン 1.4 の記述と混在しており、このマニュアルの記述に従った手順では DBT-3 ワークロードを実行することはできなかった。

今回、修正した手順により、容易に DBT-3 ワークロードを実行して、性能を測定することが出来るようになった。

また、今回の評価を行った時点では、実際に利用しているユーザが少ない PostgreSQL8.0 においても、大規模データベースの性能測定を行う DBT-3 ワークロードを実際に動かして評価することができた。また、PostgreSQL7.4 と比較して評価することで、PostgreSQL8.0 は着実に成長していることが確認できた。

DBT-3 は、大規模データベースにおける複雑なクエリーの実行において、PostgreSQL の性能を測定するのに、非常に有効なツールであることを、今回の評価を通して確信できた。

## 2.4 大規模 DB 性能(運用性)の評価手順の作成および評価

PostgreSQL7.4.6 と 8.0.0beta5 において、大規模なデータベース環境では必須となる以下の機能が、実用的な時間内で完了するのかを検証するための手順を作成し、また実際に手順に従った測定を行った。

- ・バックアップとリカバリに要する時間の性能評価
- ・大容量データのロードに要する時間の性能評価
- ・インデックスの再構築に要する時間の性能評価
- ・PostgreSQL 8.0 の Point In Time Recovery(PITR)を使用し、クラッシュリカバリに要する時間の性能評価

PostgreSQL8.0 では、テーブルスペースと呼ぶ複数の HDD を使用できる機能が追加されたので、その機能を使う場合の性能評価は重要である。

また、PostgreSQL8.0 に追加された PITR を使用したクラッシュリカバリは、信頼性のあるデータベースを運用するための重要な機能である。PITR を機能だけでなく性能においても実用性があるかを検証することは、システム構築において PITR を採用するかの、一つの判断材料になる。

## 2.4.1 実際の評価結果

今回の評価で分かった事について、以下に概要を説明する。詳細は7章を参照のこと。

### 2.4.1.1 インデックス再構築時間

インデックスの再構築に要した時間を、図 2.4-1 に示す。

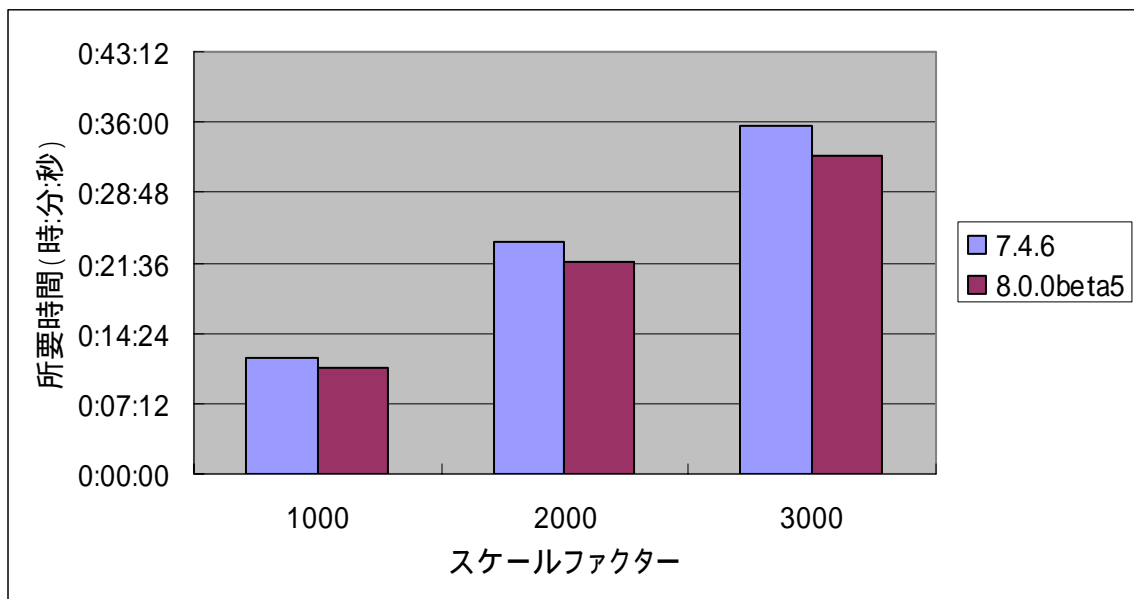


図 2.4-1 インデックス再構築時間

PostgreSQL8.0.0beta5 は 7.4.6 よりも約 10%ほど性能が良かった。

### 2.4.1.2 クラッシュリカバリ時間

クラッシュリカバリに要した時間を、通常のバックアップファイルからのリカバリ時間と比較したグラフを、図 2.4-2 に示す。

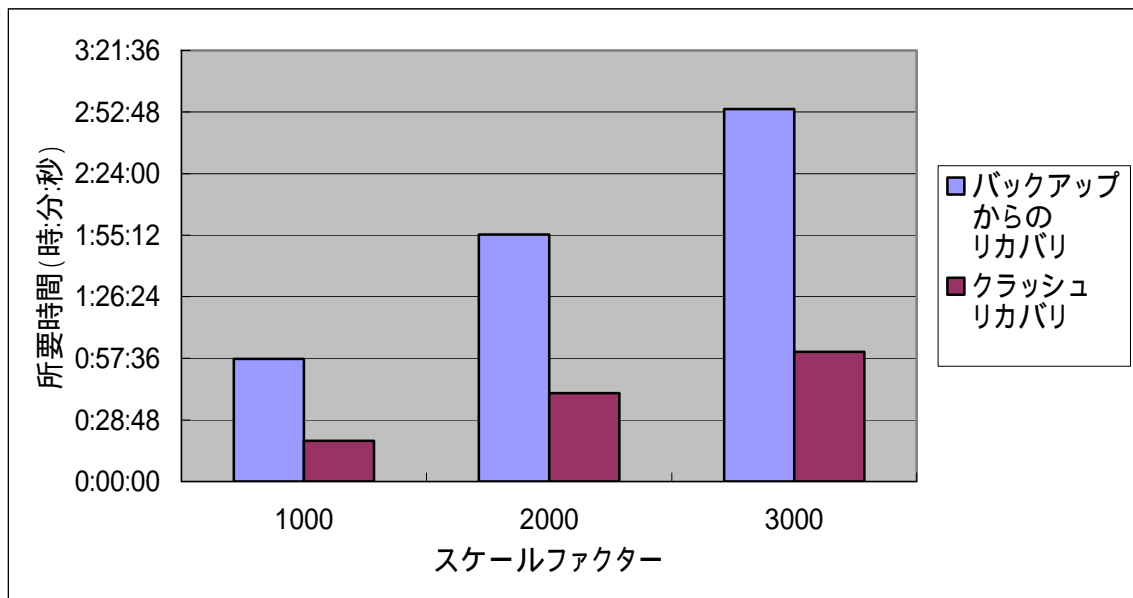


図 2.4-2 クラッシュリカバリ時間

PostgreSQL8.0 で追加されたクラッシュリカバリは、今回の評価条件では、通常のバックアップファイルからのリカバリと比べて、約 1/3 の時間でリカバリが完了した。

### 2.4.1.3 その他

その他に分かったことのサマ리를、以下に述べる。

- ・バックアップ、リカバリ、インデックス再構築、クラッシュリカバリの性能は、規模に応じてほぼ一定であった。ロードは、一定の場合と若干の変動がある場合があった。
- ・テーブルスペースを使用すると、リカバリとインデックス再構築で約 5%ほど速かった。ロードも若干速かったが、スケールファクター4000 では特に速かった。
- ・PITR を使用すると、リカバリで約 5%ほど、インデックス再構築で約 13%ほど遅くなった。
- ・PITR を使用すると、ロードとリカバリで、アーカイブログの増加量がデータベースクラスタのディスク使用量の約 1.12 倍となった。インデックス再構築でも、少しずつ増加した。

### 2.4.2 大規模 DB 性能(運用性)の評価

今回の評価では、PostgreSQL を利用した大規模データベースの運用性について、データベースのサイジングのための基礎データが得られた。また、今回の評価を行った時点では、実際に利用しているユーザが少ない PostgreSQL8.0 においても、大規模データベースの運用性について、実際に動かして評価することができた。特に、8.0 で追加されたテーブルスペースと、PITR を使用したクラッシュリカバリについて、効果があることが確認できた。

## 2.5 DBT-1のスコープ拡大のためのMySQL向け改変調査作業

### 2.5.1 DBT-1の問題点

今回の評価結果によって、適用範囲の拡大や、手順の確立等の一定の成果が得られたが、以下に関して問題が残っていると考えられる。

- ・RDBMS固有のストアプロシージャに依存するためにポータビリティがない
- ・RDBMSによって実装が異なる(ストアプロシージャそのものの違い・制約、データの取得方法)ので測定結果の比較がしづらい。
- ・今後、ストアプロシージャによるビジネスロジックの実装は減少すると考えられ、DBT-1の評価結果が実際のDB性能を表現できなくなる可能性がある。
- ・TPC-Wとの違いがある(HTTPリクエストを戻していない、未実装のトランザクションがある、など)
- ・結果が数値情報のみであり、視認性に欠ける。

特にストアプロシージャによる実装上の問題点は大きいと考えられる。

### 2.5.2 対応方針

DBT-1のスコープ拡大の施策として、当初はMySQL5.0に実装されるストアプロシージャを利用した改変を想定していたが、前述のとおりストアプロシージャによる実装の問題が大きいために、別な手段でこれを解決する事とした。

具体的には、トランザクション処理をストアプロシージャで記述せずに、DBT-1のLIB ODBCにポーティングする。また、測定結果のビジュアルライズも同時に検討した。

これによりDBT-1自体のポータビリティを向上させるとともに、より現実に近いRDBMS評価環境にする事を目標においた。

実際の開発作業は次期フェーズで、OSDLなどのオープンソースコミュニティと連携して行う予定である。

### 2.5.3 実装方式

現状、ストアプロシージャで実装されているトランザクション処理をRDBMSから分離し、DBT-1 LIB ODBCに移行する。ストアプロシージャで記述されているトランザクション部分をC言語で記述する。またトランザクション中で利用するSQLコマンドセットを別ファイルにまとめ、他RDBMS移行時でのポータビリティの向上を狙う。

結果のビジュアルライズはDBT-1本体とは別スクリプトとし、DBT-1のアウトプットファイルから、グラフ出力する機能を追加する。

## 2.5.4 見積り

第 8 章の通り、ターゲット RDBMS を MySQL5.0 とし、修正規模は表 2.5-1 のとおりに見積もった。

表 2.5-1 修正規模概算

修正内容	見積もりステップ数
トランザクション処理部分	8 1 8 0
データベーススクリプト部分	6 1 5

なお、オープンソースとして公開するために必要な諸作業(make 環境の構築など)は含んでいない。

結果のビジュアルライズは既存のグラフ出力のためのオープンソースプロダクトを利用する事とし、詳細な見積もりは行っていない。

## 2.6 各ソース・パッチ / 修正事項の概要

### 2.6.1 DBT-1

MaxDB7.5 版

- ・ make 環境の修正
- ・ エミュレートするユーザ数の上限を変更するためのスレッド実装修正
- ・ MaxDB 用に SQL コマンドを修正
- ・ 欠落していた ODBC 接続の再作成

PostgreSQL 版

- ・ エミュレートするユーザ数の上限を変更するためのパッチ

Oracle9i 版

- ・ OCI(Oracle Common Interface)による実装。ソース一式。

### 2.6.2 DBT-3

DBT-3 ver1.5

- ・ 多くの不具合を修正

### 2.6.3 大規模 DB 性能(運用性)の評価

PostgreSQL 版

- ・ 性能測定用のパッチ

## 3 DBT-1 による MaxDB の評価

### 3.1 概要

DBT-1 を利用して MaxDB7.5 の性能測定を行う。

DBT-1 は元々 SAP DB7.3、7.4(注1)用に開発されていたが、PostgreSQL 用に改変された時点で SAP-DB(ODBC 接続)用のメンテナンスが停止されていた。

本プロジェクトでは、ODBC 接続の DBT-1 をベースに MaxDB7.5 用に修正を加えた。MaxDB7.5 での評価を行うとともに、8 章で述べる MySQL 用改変調査の事前調査としての意味を持っている。

DBT-1 の概観については、8 章 MySQL 用改変調査 DBT-1 概観を参照の事。

注1) 独 SAP AG 社は同社の ERP 用 RDBMS として利用していた Software AG 社の Adabas D を、2000 年から SAP-DB の名前でオープンソース RDBMS として公開した。更に 2004 年から MySQL AB 社とのクロスライセンスを行い、現在は MySQL AB 社から MaxDB として公開している。

### 3.2 MaxDB へのポータリング

#### 3.2.1 ポータリング方針

以下の 2 点をポータリング方針としている。

- ・ DBT-1 オリジナルの ODBC 接続版(SAP DB7.3、7.4 向け)を MaxDB7.5 にポータリングする。
- ・ 既存の構造は出来る限り変更しない。変更が必要な場合は出来る限り最小限に留める。(ストアードプロシジャベースの MaxDB 用 DBT-1)

表 3.2-1 に、修正を加えたファイル及び、修正行数の一覧を掲載する。尚、行数は diff コマンドによって得られた差分を集計したものであり、追加行と削除行を合計した値としている。

表 3.2-1 修正ファイル一覧

ファイル名	修正行数
configure	16
configure.in	14
appServer/appServer.c	56
appServer/app_threadpool.c	5

cache/cache.c	2
common/common.c	4
data_collect/analyzeBT.pl	3
data_collect/dbt1_slave.pl	202
data_collect/pgsql/db_stats.sh	3
data_collect/pgsql/get_config.sh	51
datagen/main.c	4
dbdriver/eu.c	35
dbdriver/main.c	36
include/common.h	1
include/eu.h	2
include/odbc_interaction_shopping_cart.h	2
interfaces/db.c	1
interfaces/odbc/Makefile.in	40
interfaces/odbc/odbc_interaction.c	1
interfaces/odbc/odbc_interaction_admin_confirm.c	3
interfaces/odbc/odbc_interaction_admin_request.c	3
interfaces/odbc/odbc_interaction_best_sellers.c	3
interfaces/odbc/odbc_interaction_buy_confirm.c	23
interfaces/odbc/odbc_interaction_buy_request.c	3
interfaces/odbc/odbc_interaction_home.c	3
interfaces/odbc/odbc_interaction_new_products.c	3
interfaces/odbc/odbc_interaction_order_display.c	3
interfaces/odbc/odbc_interaction_order_inquiry.c	3
interfaces/odbc/odbc_interaction_product_detail.c	3
interfaces/odbc/odbc_interaction_search_request.c	3
interfaces/odbc/odbc_interaction_search_results.c	3
interfaces/odbc/odbc_interaction_shopping_cart.c	102
scripts/sapdb/add_aux_structure.sql	6
scripts/sapdb/backup_db.sh	16
scripts/sapdb/create_db.sh	66
scripts/sapdb/create_tables.sh	3
scripts/sapdb/define_medium.sh	6
scripts/sapdb/drop_db.sh	6
scripts/sapdb/drop_dbproc.sh	3
scripts/sapdb/load_db.sh	17
scripts/sapdb/load_dbproc.sh	60
scripts/sapdb/monitor.sh	22

scripts/sapdb/restart.sh	9
scripts/sapdb/restore_db.sh	3
scripts/sapdb/set_param.sh	12
scripts/sapdb/trace_off.sh	12
scripts/sapdb/trace_on.sh	3
scripts/sapdb/update_stats.sh	2
scripts/stats/collect_data.sh	24
scripts/stats/db_stats.sh	36
scripts/stats/dbt1.config	34
scripts/stats/get_xcons.sh	2
scripts/stats/run_dbt1.sh	124
storedproc/sapdb/DigSyl.sql	3
storedproc/sapdb/InsertCust.sql	10
storedproc/sapdb/addToSC.sql	3
storedproc/sapdb/admin_confirm.sql	2
storedproc/sapdb/admin_request.sql	2
storedproc/sapdb/buy_confirm.sql	3
storedproc/sapdb/buy_request.sql	3
storedproc/sapdb/createSC.sql	7
storedproc/sapdb/getCustInfo.sql	6
storedproc/sapdb/getCustName.sql	13
storedproc/sapdb/getOrderItems.sql	1
storedproc/sapdb/getPromoImages.sql	2
storedproc/sapdb/getSCDetail.sql	1
storedproc/sapdb/getSCSubTotal.sql	3
storedproc/sapdb/initOrderItems.sql	1
storedproc/sapdb/initSCItems.sql	1
storedproc/sapdb/order_display.sql	1
storedproc/sapdb/order_inquiry.sql	2
storedproc/sapdb/product_detail.sql	1
storedproc/sapdb/refreshSC.sql	6
storedproc/sapdb/shopping_cart.sql	4
storedproc/sapdb/updateSC.sql	4
tools/Makefile.in	6
tools/interaction_test.c	9
tools/load_db_proc.c	1
合計	1197

### 3.2.2 主な修正点

DBT-1 の最新バージョン 2.1 では、SAP DB に対するメンテナンスが停止した状態で、configure 並びに C ソースコード上では、条件分岐で切り分けられているものの、一部では PostgreSQL 専用のコードがハードコーディングされていた。

また、コードそのものにもバグがあり、初期に生成したバイナリーでは、DB への接続が出来ない現象が発生した。

その他の主な修正項目は以下の通り。

#### 3.2.2.1 MaxDB コマンドの修正

DBT-1 が実装した SAP DB7.3 と今回の実装対象である MaxDB7.5 では、DB 操作のコマンドに多くの差があるため、MaxDB のコマンドを使用するよう修正を加えた。また、SAP DB モジュールへの絶対パス指定を、環境変数(\$PATH)に取り込み、パスの記述を削除することで、MaxDB のインストール先を自由に設定できるようにした。

MaxDB7.5 のストアドプロシージャでは、コメントの記述方法や”;"だけの空文の扱いの仕様が変更されており、構文解釈でエラーが発生したため、該当部分の記述を削除した。

#### 3.2.2.2 スレッド生成ロジックの修正

各プログラムでは、仮想ユーザ数分のスレッドが生成される。この時、1 スレッド当たりのスタックサイズを調整せずプログラムを実行すると、250 スレッド前後でスレッド生成エラーが発生し、十分な仮想ユーザ数による計測を実施出来ない事態が発生した。そこで、スレッドが使用するスタックサイズを小さな値に変更し、生成できるスレッド数(仮想ユーザ数)を増加させるように修正を加えた。修正前に生成可能であったスレッド数を、表 3.2-2 に記す。

表 3.2-2 チューニング前 生成可能なスレッド数

ディストリビューション	スタック調整前接続件数	スタック調整後接続件数
SUSE	210	1000
MIRACLE LINUX	260	960
RedHat AS	260	960

スレッドスタックサイズを appServer で 16KB、dbdriver で 200KB に設定した結果、1000 近くまでスレッドを生成できるようになった。しかし、MaxDB のパラメータをチューニングすることにより、仮想ユーザ数 1000 前後では性能限界に達しなかった。最終的には、システムパラメータも調整することで、より多くの仮想ユーザによる計測が可能となり MaxDB の性能限界を測定することが出来た。

### 3.2.2.3 データ生成モジュールの修正

MaxDB データロードプログラム loadercli の構文解釈機能に問題があり、特定のデータパターンを正しく処理することが出来なかった。本来であれば、ロードプログラム側に対して修正を加えるところであるが、データそのものはパフォーマンスに影響を与えるものではないため、ロードプログラムが解釈可能なデータのみ生成されるように修正した。

具体的には、特定の特殊記号のコードパターンが生成されないようにデータ宣言部の特殊記号を通常の文字コードに変更した。

### 3.3 環境定義書

#### 3.3.1 システム構成

DBT-1 の評価構成を図 3.3-1 に示す。今回の測定では、全てのプロセスを 1 台のサーバ上で稼働させる。

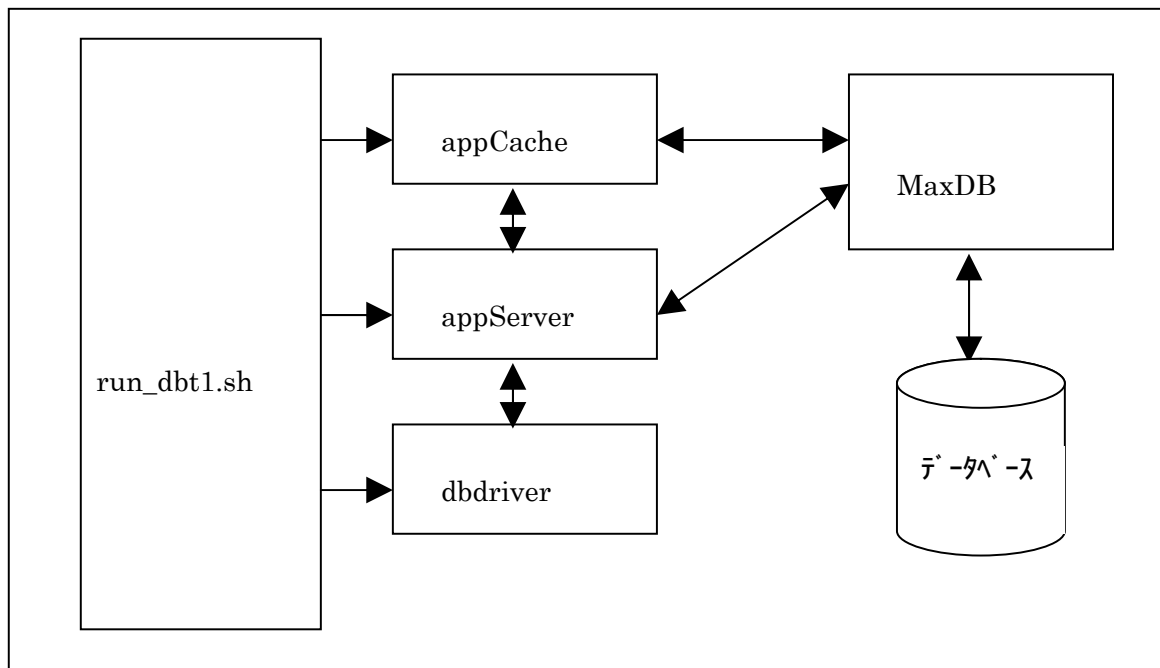


図 3.3-1 システム構成図

##### 3.3.1.1 ハードウェア

DELL PowerEdge 1850  
Xeon 3.6GHz × 2 (ハイパースレッド有効)  
MEMORY 4GB  
L2 CACHE 1MB  
DISK 72GB(10000rpm) × 2

##### 3.3.1.2 DISK 構成

表 3.3-1 DISK 構成

72GB	/boot	100MB
	swap	1GB
	/	RAID0
72GB	swap	swap
	/	RAID0

### 3.3.1.3 ソフトウェア

- SUSE LINUX Enterprise Server 9  
Linux 2.6.5-7.97-default SMP #1 SMP  
Fri Jul 2 14:21:59 UTC 2004 i686 i686 i386 GNU/Linux
- MIRACLE LINUX V3.0 – Asianux Inside  
Linux 2.4.21-9.30AXsmp #1 SMP  
Thu May 27 00:03:41 EDT 2004 i686 i686 i386 GNU/Linux
- Red Hat Enterprise Linux AS 3  
Linux 2.4.21-4.ELsmp #1 SMP  
Fri Oct 3 18:13:58 EDT 2003 i686 i686 i386 GNU/Linux  
Linux 2.4.21-27.ELsmp #1 SMP Wed Dec 1 21:59:02 EST 2004
- MaxDB 7.5.00.16
- DBT-1 V2.1 改訂版(dbt1-v2.1-maxdb-1.0.tar.gz)

各 Linux は、それぞれのディストリビュータに含まれる CD の内容のみを使用する事とし、パッチの適用は行わない事とした。

ただし、RedHatAS は、高負荷状態において計測できない現象が発生したため、一部の計測は入手可能な最新アップデート版を使用した。そのため二つのバージョンを記載した。

### 3.3.2 Linux のインストール

各ディストリビューションのデフォルトインストールを想定している。  
インストールに関しては各々のディストリビューションのインストール・マ  
ニュアル等に従うこと。

但し、今回の性能評価において、カーネルの違いに焦点を当てるため、ファイル  
システムは ext3 で統一した。

対象ディストリビューション： SUSE、MIRACLE LINUX、RedHat AS

注)ssh、gcc、sysstat、libstdc++6.2.2 環境は必ずインストールする。

### 3.3.3 MaxDB のインストール

#### 3.3.3.1 ダウンロード

root ユーザでログインし、MySQL の Web サイト(<http://www.mysql.com/>)より MaxDB アーカイブをダウンロードする。

以降の作業では IA32 用 Linux 版フルセット、バージョン番号は 7.5.00.16 を利用する。バージョンは環境に応じて選択すること。

#### 3.3.3.2 MaxDB の解凍

tar コマンドを使用し、ダウンロードした tarball を展開する。

```
# tar -xzvf maxdb-all-linux-32bit-i386-7_5_00_16.tgz
```

#### 3.3.3.3 MaxDB のインストール

展開先のディレクトリに移動し、インストールコマンド SDBINST を実行する。処理は「10 : all」を選択し、全てのモジュールを導入する。

```
# cd maxdb-all-linux-32bit-i386-7_5_00_16
# ./SDBINST
existing profiles:

0:  Server
1:  Runtime For SAP AS
2:  DB Analyzer
3:  C Precompiler
4:  Webtools
5:  ODBC
6:  JDBC
7:  Script Interface
8:  Loader
9:  XML Indexing Engine
10: all
11: none
please enter profile id: 10
```

インストール中、ユーザ/グループ及びインストール先ディレクトリを指定する必要がある。

```
please enter group name for database programs [sdba]:
unknown group - create "sdba" on local machine? (y/n) y
please enter owner name for database programs [sdb]:
unknown user - create "sdb" on local machine? (y/n) y
please enter independent data path [/var/opt/sdb/data]:
directory "/var/opt/sdb/data" does not exist, create? (y/n) y
please enter independent program path [/opt/sdb/programs]:
directory "/opt/sdb/programs" does not exist, create? (y/n) y
```

また、データベース カーネル インストールパスも指定する。

```
please enter dependent path [/opt/sdb/7500]:
directory "/opt/sdb/7500" does not exist, create? (y/n) y
```

インストールが成功した場合、最後に以下のメッセージが表示される。

```
installation of MaxDB Software finished successfully Tu, Dec 28, 2004 at
11:56:50
```

上記例は全てデフォルト値を設定する場合。

ここでは SDBINST を利用して標準的なインストールを行う。(推奨)

rpm コマンドで MaxDB のモジュールをインストールしている場合は、一旦全てをアンインストールし、再度 SDBINST にてインストールすること。

質問項目で[]で囲まれたデフォルト値が提示される。エンターキーを入力することで、デフォルト値が入力情報として反映される。

「(yes/no)」の問いに対しては必ず " y " 或いは " n " を入力する。

メッセージの日時は、各マシンの現在時刻が表示される。

### 3.3.4 DBT-1 用の事前環境設定

#### 3.3.4.1 ユーザの作成

root ユーザでログインし、sapdb 用のユーザを作成する。  
ここでは ユーザ ID=sapdb グループ=sapdb パスワード=sapdb としている。

```
# groupadd sapdb
# useradd -g sapdb -d /home/sapdb -s /bin/bash sapdb
# passwd sapdb
Changing password for user sapdb
New password:
Retype new password:
password:all authentication tokens update successfully
```

#### 3.3.4.2 sudo の準備

ログ情報には、root 権限が必要なコマンドがあるため、DBT-1 を実行するユーザが sudo を利用できるよう、/etc/sudoers ファイルを編集する。編集方法は、root ユーザで visudo を起動し、以下の 1 行を追加する。

```
%sapdb ALL=(ALL) NOPASSWD: ALL
```

#### 3.3.4.3 ユーザ環境設定

sapdb ユーザでログインし、~/bashrc ファイル 内に以下の設定を追加する。

```
export PATH=$PATH:/opt/sdb/programs/bin:/opt/sdb/7500/bin:.
export LD_LIBRARY_PATH=/opt/sdb/programs/lib:/opt/sdb/7500/lib
export SID1=DBT1
```

#### 3.3.4.4 ssh の準備

DBT-1 の各アプリケーションの起動は、リモート環境に対応した構成となっており、セキュリティを確保する意味で、ssh を使用している。ssh-keygen でキーを生成する。

入力するキーの格納場所とキーのパスフレーズは以下の通り。  
今回ディレクトリはデフォルト値を使用するため、Enter キーのみ入力する。

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/sapdb/.ssh/id_rsa):
Created directory '/home/sapdb/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/sapdb/.ssh/id_rsa.
Your public key has been saved in /home/sapdb/.ssh/id_rsa.pub.
The key fingerprint is:
ac:16:14:89:01:ee:31:1c:da:1d:b3:5b:84:b8:2d:68
sapdb@localhost.localdomain

$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 600 ~/.ssh/authorized_keys
```

### 3.3.5 DBT-1 のインストール

#### 3.3.5.1 DBT-1 のダウンロード

sapdb ユーザでログインする。

修正済み MaxDB 用 DBT-1 ソースをダウンロードする(OSDL or IPA サイトより)

#### 3.3.5.2 DBT-1 の解凍

次のコマンドで DBT-1 を展開する。

```
$ tar -xzvf dbt1-v2.1-maxdb-1.0.tar.gz
```

#### 3.3.5.3 DBT-1 のコンパイル

MaxDB 用オプションを利用して、コンパイルする。

```
$ cd ~/dbt1-v2.1  
$ configure --without-postgresql --with-sapdb  
$ make
```

### 3.3.6 パラメータの設定

#### 3.3.6.1 OSパラメータ

1 プロセス内で、同時に開くことが出来るファイル数の上限。プログラム内部では、スレッド単位にソケットを開くため、この数が少ないと、スレッドを生成しても DB 接続に失敗する。

表 3.3-3 ディストリビューション別 チューニング前の ulimit -a 状況

	SUSE	MIRACLE LINUX	RedHatAS
Core file size	0	0	0
data seg size	unlimited	unlimited	unlimited
file size	unlimited	unlimited	unlimited
max locked memory	unlimited	4	4
max memory size	unlimited	unlimited	unlimited
open files	1024	2048	1024
pipe size	8	8	8
stack size	unlimited	10240	10240
Cpu time	unlimited	unlimited	unlimited
max user processe	16383	2047	7168
virtual memory	unlimited	unlimited	unlimited

プログラム内部から、ログを出力する際にセマフォを使用し排他制御を行っている。そのため、スレッド数が増えるに従って、必要となるセマフォの数も多くなる。

表 3.3-4 各ディストリビューションのチューニング前のセマフォ数

	SUSE	MIRACLE LINUX	RedHatAS
セマフォ id ごとのセマフォの数	250	256	256
システム全体のセマフォの数	32000	32000	32000
一度に実施できるセマフォ操作の数	32	100	32
セマフォ id の最大値	128	142	128

今回の性能評価においてパフォーマンスを上げる為のチューニングを対象とはしないが、DB の性能限界を調査するため、必要に応じて拡張する。

open files はスレッド生成時に影響し、セマフォは、appServer から MaxDB への接続時に影響を与える。対処方法は、後述のチューニングにおいて記述する。

### 3.3.6.2 DB パラメータ

MaxDB パラメータの変更は、dbmcli を起動し行う。今回使用するデータベースとデフォルトユーザの場合、以下のコマンドとなる。

```
$ dbmcli -d DBT1 -u dbm,dbm param_directput パラメータ名 設定値
```

今回の試験では、表 3.3-5 のように修正を加えた。

表 3.3-5 MaxDB チューニングパラメータ

パラメータ名	設定値チューニング前	設定値チューニング後
MAXCPU	1	4
CACHE_SIZE	10000	50000
CAT_CACHE_SUPPLY	6432	25000

## 3.4 評価手順書

### 3.4.1 前提条件

各ディストリビューションによる違いを検証するために、システム環境を同一の状態にして検証する。今回使用するサーバは、複数の DISK ドライブが実装されているため、RAID0 構成とし ext3 にてフォーマットする。またファイルシステムは、”/”のみとした。

導入するパッケージは標準構成をベースとし、ディストリビューションに含まれる CD の内容のみ使用する。但し DBT-1 を実行するために必要なパッケージは適宜追加する。

### 3.4.2 テストデータの作成

#### 3.4.2.1 データ用ディレクトリの作成

sapdb ユーザでログインする  
データを格納するディレクトリを設定する

```
$ mkdir ~/data  
$ chmod a+w ~/data
```

#### 3.4.2.2 データの生成

次のコマンドでテスト用の基礎データ(Text ファイル)の生成を行う

```
$ cd ~/dbt1-v2.1/datagen  
$ datagen -d SAPDB -i 10000 -u 1000 -p /home/sapdb/data -T i -T c -T a -T d -T o
```

オプションの解説

-d : データベースタイプ(SAPDB または PGSQL。今回は SAPDB)  
-i : 生成するアイテム総数  
-u : 生成する仮想ユーザ数  
-p : 実際に生成するデータディレクトリ  
-T : データを生成するテーブル名  
    i -- item  
    c -- customer  
    a -- author  
    d -- address

o -- order

-p に与えるパスは、フルパスで指定する。

指定したディレクトリ内にデータが作成される。

実際に生成されたデータファイルには/tmp ディレクトリからシンボリックリンクが張られる。

データロードには、/tmp ディレクトリからのパスが使用される。

上記パラメータで作成されるデータの総容量は、約 3.5GB となる。

### 3.4.3 データベースの作成

次のコマンドで MaxDB にデータベースを作成する。デフォルトで作成されるデータベース名は DBT1 となる。

```
$ cd ~/dbt1-v2.1/scripts/sapdb
$ build_db.sh
```

DB の領域は、create\_db.sh 内で定義する。

以下の数値は、3.4.2.2 のパラメータで作成したデータが格納可能な最小容量である。

```
param_addvolume 1 LOG LOG_001 F 160000
```

```
param_addvolume 1 DATA DAT_001 F 800000
```

各数値は 8KB 単位で DISK を使用する。LOG は DATA の 20%程度の容量を指定する。必要な容量は”datagen”で生成したデータの約 1.5 倍を必要とする。

MaxDB では、データ領域として、6.4GB を確保する。

### 3.4.4 パラメータの設定

#### 3.4.4.1 DBT-1 の環境設定

エディタで DBT-1 設定ファイル `~/dbt1-v2.1/scripts/stats/dbt1.config` を編集する。

- ・ホスト名を `localhost` に修正
- ・ディレクトリを各バイナリが存在するディレクトリに修正
- ・総エミュレートユーザ数を計測を行う仮想ユーザ数に修正

`dbt1.config` の各行は以下の通り。

```
[database]
#hostname instance username password
localhost:DBT1:dbt:dbt
[cache]
#hostname port dbconnections items appCache_executable_directory
localhost:9999:5:10000:/home/sapdb/dbt1-v2.1/cache
[appServer]
#hostname                               appServer を実行するサーバ名
localhost
#port                                   appServer への接続ポート番号
9992
#dbconnection                           MaxDB コネクション数
20
#transaction_queue_size                 dbdriver 通信バッファ
1000
#transaction_array_size                 各処理バッファ
1000
#items                                   生成アイテム数
10000
#appServer executable directory         appServer 実行ディレクトリ
/home/sapdb/dbt1-v2.1/appServer
[dbdriver]
#hostname                               dbdriver を実行するホスト名
localhost
#items                                   生成アイテム数
10000
```

#customers	生成顧客数
2880000	
#eu	総エミュレートユーザ数
<u>200</u>	
#eu/min	ユーザ接続レート(ユーザ/分)
100	
#mean think_time	ユーザオペレーション間隔
7.2	
#run_duration in seconds	各ユーザ実行時間
4100	
#dbdriver executable directory	dbdriver 実行ディレクトリ
/home/sapdb/dbt1-v2.1/dbdriver	

今回の評価では、エミュレートユーザ数のみを変えて評価を実施する。(下線部) 複数のサーバに実行を分散する場合、サーバーの数だけ、全項目を":"で区切って定義を追加する。

### 3.4.5 起動方法

#### 3.4.5.1 データベースの起動

sapdb ユーザでログインし、MaxDB のサーバエージェント(x\_server)を起動する。データベースコマンド(dbmcli)で DBT1 データベースをオンラインにする。

```
$ x_server start
$ dbmcli -d DBT1 -u dbm,dbm db_online
```

システム起動時に毎回実行する必要がある。

#### 3.4.5.2 ssh-agent 起動

SSH 接続のためのエージェント(ssh-agent)を起動する。

```
$ eval `ssh-agent`  
Agent pid 4771  
$ ssh-add ~/.ssh/id_rsa  
Enter passphrase for /home/sapdb/.ssh/id_rsa:  
Identity added: /home/sapdb/.ssh/id_rsa (/home/sapdb/.ssh/id_rsa)
```

DBT-1 を実行するターミナルウィンドウにて実行すること。  
入力するパスフレーズは、3.3.4.4 にて入力した値を入力する。  
初回起動時は、可否を聞いてくるので、事前に一度 ssh コマンドを実行しておく必要がある。

#### 3.4.5.3 トランザクション・パラメータの設定

トランザクション作業領域の容量が不足すると、DB とは関係の無い箇所で処理待ちが生じるため、以下の値は”#eu”以上の値を指定する。

```
#transaction_queue_size  
#transaction_array_size
```

今回の試験では、エミュレートユーザ数による変化を検証するため、”#eu”以外の値は修正を加えない。使用する値は、OSDL-J にて公開されている結果をベースとしている。

#### 3.4.5.4 測定

次のコマンドで DBT-1 を測定する。

```
$ cd dbt1-v2.1/scripts/stats  
$ run_dbt1.sh /home/sapdb/U200
```

今回は、DB の性能評価であるため、appCache を使用しない。  
run\_dbt1.sh に引数として与えるディレクトリは事前に作成する必要はない。  
引数のディレクトリ(/home/sapdb/U200)に BT ファイルが作成されれば、試験完了。

### 3.4.5.5 結果の収集

#### (1) 測定時間

dbt1.config ファイル内、“# run\_duration in seconds” で指定した時間 + 全ユーザが接続するまでの時間処理が行われる。

#### (2) 測定結果

性能評価結果や、MaxDB のログ出力先は、測定コマンド run\_dbt1.sh のパラメータとして与えられる。また、各プログラムの測定結果は、/tmp ディレクトリに出力される。エラーログやデバッグ用の情報は、各アプリケーション実行ディレクトリに出力される。/tmp ディレクトリや実行ディレクトリに出力される情報ファイルの拡張子は、何れも“.log”として出力される。

出力される情報の詳細を表 3.4-1 DBT-1(MaxDB)ログ情報一覧に記す。

表 3.4-1 DBT-1(MaxDB)ログ情報一覧

情報種別	コマンド	出力ファイル名	備考
BT/秒	results	BT	DBT-1 コマンド
システム統計	sar -A	run.sar.data	10 秒間隔
I/O 情報	iostat -d	io.txt	10 秒間隔
プロセス一覧	top d 120 b	top.txt	120 秒間隔
DB ロックチェック	select * from DBA_LOCKS	lockstats*.out	*:取得回数
DB キャッシュ	select * from monitor_caches	m_cache*.out	*:取得回数
DB 負荷	select * from monitor_load	m_load*.out	*:取得回数
DB ロック	select * from monitor_lock	m_lock*.out	*:取得回数
DB ログ	select * from monitor_log	m_log*.out	*:取得回数
DB ページ	select * from monitor_pages	m_pages*.out	*:取得回数
DB カラム	select * from monitor_row	m_row*.out	*:取得回数
DB トランザクション	select * from monitor_trans	m_trans*.out	*:取得回数
DB データ情報	info data	datadev0.txt	MaxDB コマンド
DB ログ情報	info log	logdev0.txt	MaxDB コマンド
DB ステータス一覧	x_cons show all	x_cons*.out	*:取得回数

システム統計情報の解析は、run.sar.data ファイルより必要な情報を取り出す。

## 3.5 測定結果と考察

### 3.5.1 測定結果

#### 3.5.1.1 測定結果の収集

各ディストリビューション上にて、データベースパラメータをインストール時の状態で、DBT-1 による測定を実施する。以後この状態での測定を、「チューニング前」と呼称する。

データベースの性能限界は、仮想ユーザ数 100 から開始し 200 以降は、BT/秒の伸びが頭打ちになるまで、200 単位で増やしていく事で測定した。しかし、システム構成が十分なパフォーマンスを発揮する環境では、仮想ユーザ数 200 以下では特性に差が出ない事が判明したため、仮想ユーザ数 100 は評価対象外とし、以降の評価には含まない事とした。評価対象のデータは、各仮想ユーザ数にて 5 回測定し、BT/秒及びインタラクション応答時間の平均を算出したものとする。尚、その他の詳細な情報を示す場合は、平均 BT/秒に近似する値を持つ結果セットを使用する。仮想ユーザ数の設定方法は、「3.4 評価手順書」を参照のこと。

DBT-1 実行パラメータは、OSDL-J にて公開されている DBT-1 の結果を基準にして、ThinkTime を 7.2 秒、仮想ユーザ数最大値を 800 と想定したが、BT/秒の伸びが頭打ちになることを確認できなかったため、仮想ユーザ数を更に拡張して測定を継続した。

その際の修正事項として、3.2.2.2 スレッド生成ロジックの修正を行った。また、各 Linux のシステムパラメータである、オープン可能なファイル数を確認した結果、SUSE のチューニング前では、オープン可能なファイル数が 1024 であり、この制限のため、1000 以上のスレッドを生成した際、エラーが発生した。そのため、以下のファイルを編集し制限を拡張した。

```
/etc/security/limits.conf
```

```
* hard nofile 4096
* soft nofile 4096
```

“nofile”の記述が無い場合は、上記 2 行を追加し、記述がある場合は値を修正する。

また、appServer から DB へ接続する際、セマフォ不足も発生しエラーログも出力されたため、以下のコマンドをシステムブート毎に実行した。

エラーログ ~/dbt1-v2,1/appServer/error.log

```
Sun Feb  6 14:03:15 2005
2098600880
/home/sapdb/dbt1-v2.1/interfaces/odbc/odbc_interaction.c:53
[1] sqlstate 08001 : [MySQL MaxDB][LIBSQLLOD SO][MaxDB] Unable to
connect to data source;-709 CONNECT: (cannot create communication
semaphore)
Sun Feb  6 14:03:15 2005
2098600880
app_threadpool.c:150
db_connect error
```

対処コマンド

```
$ echo "8192 1024000 256 256" > /proc/sys/kernel/sem
```

sem は 4 つの値を取り、順番に セマフォ id ごとのセマフォの数、システム全体のセマフォの数、一度に実施できるセマフォ操作の数、セマフォ id の最大値を指す。

その結果、BT/秒の伸びが頭打ちになることが確認できるまで、仮想ユーザ数を拡張する事が可能となった。

### 3.5.1.2 BT/秒

BT/秒は DBT-1 が出力する擬似トランザクションの処理数を表し、値が高いほど性能が高いとされる。以下に各ディストリビューションにて、DB パラメータ設定をインストール時の状態(以後、チューニング前と記す)で測定した結果を示す。

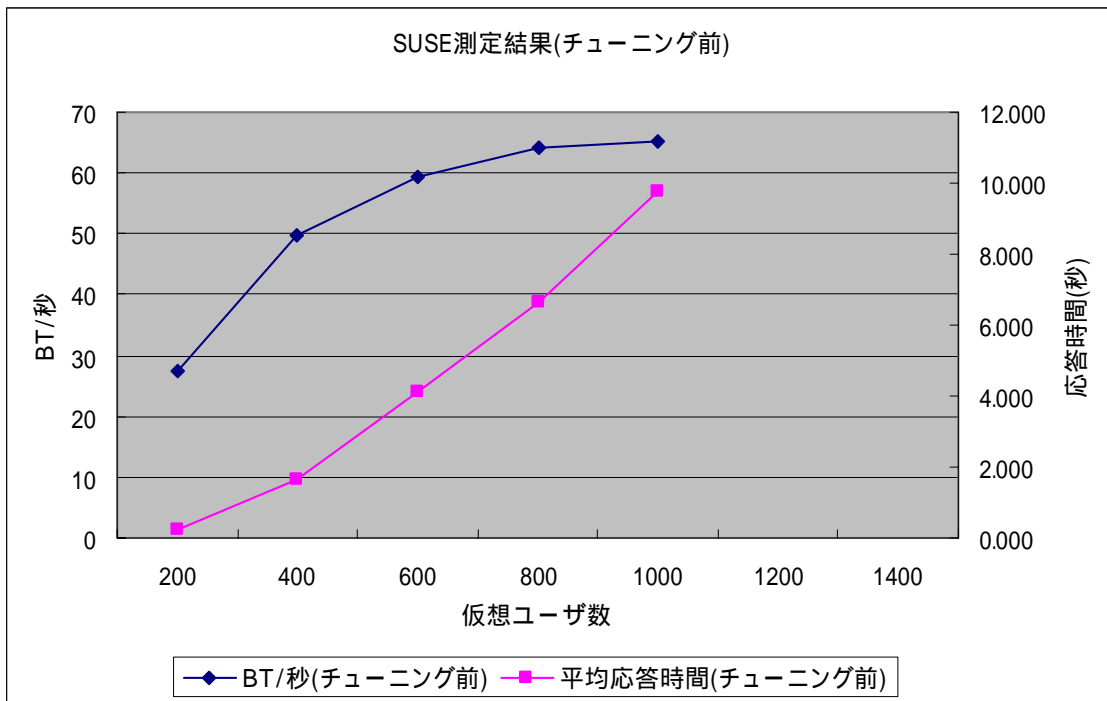


図 3.5-1 SUSE BT/秒推移(チューニング前)

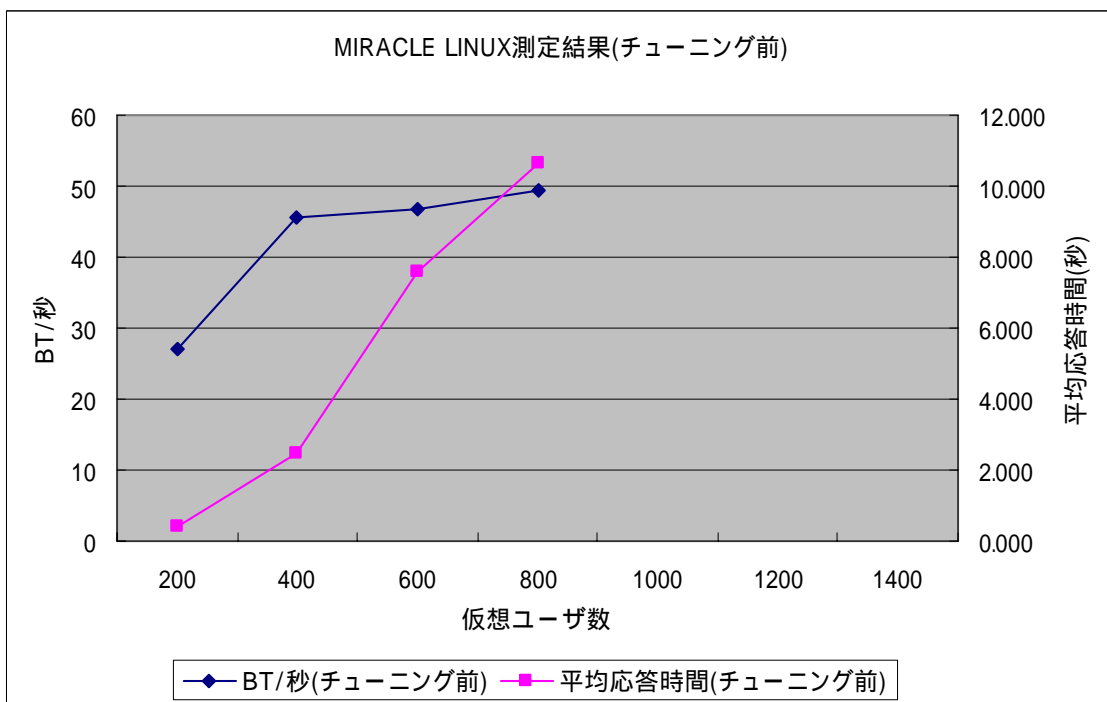


図 3.5-2 MIRACLE LINUX BT/秒推移(チューニング前)

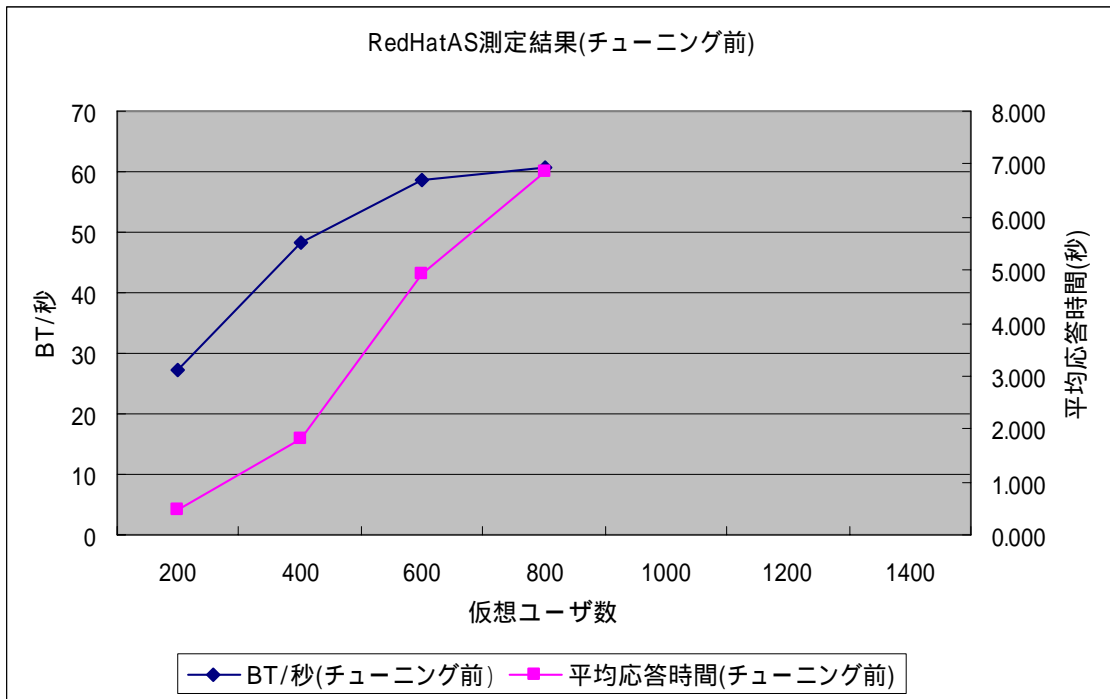


図 3.5-3 RedHat AS BT/秒推移(チューニング前)

### 3.5.1.3 リソース利用率

次は、同様の条件での CPU 利用率の推移である。

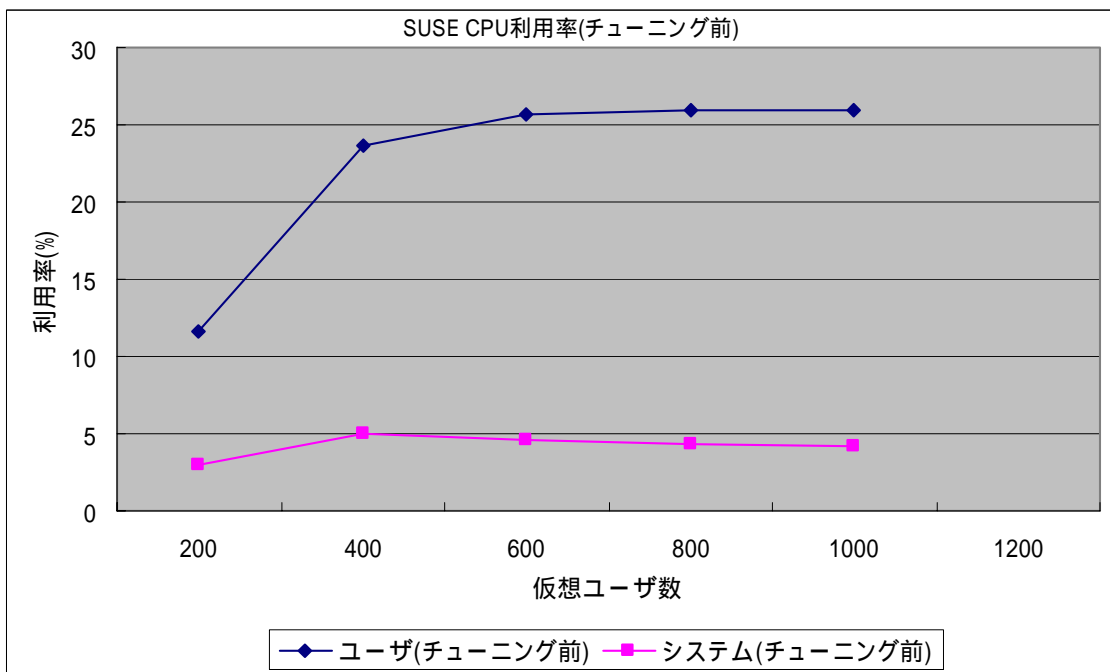


図 3.5-4 SUSE CPU 利用率(チューニング前)

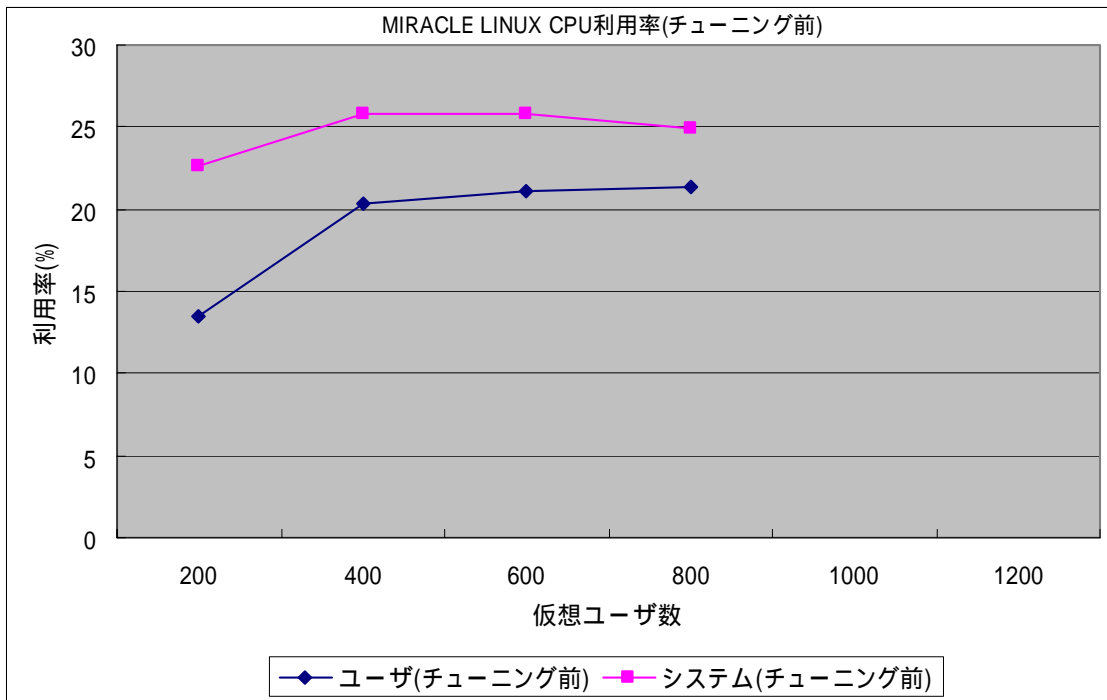


図 3.5-5 MIRACLE LINUX CPU 利用率(チューニング前)

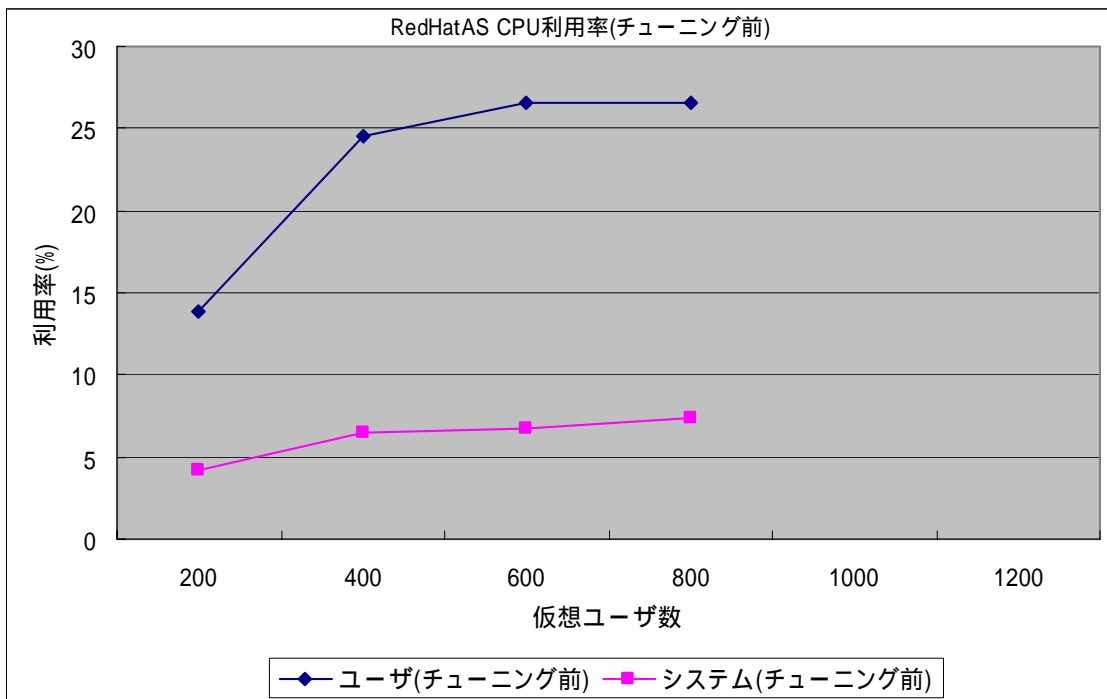


図 3.5-6 RedHatAS CPU 利用率(チューニング前)

### 3.5.2 チューニング前の結果の考察

チューニング前のパラメータで測定した場合、仮想ユーザ数 400 から鈍化し 800 で最大値に到達した。この時の出力されたログを見ると、CPU 利用率平均 25%、DB 内部では、IO Wait が多く発生しており、CPU 利用率からはシステムリソースにはまだ余裕がある状態である。プログラム内部のキュー領域は十分なサイズを確保 (transaction\_array) した状況で、IO Wait が多いということは、MaxDB がユーザからの要求に対応しきれない状況が見て取れる。また、MaxDB キャッシュヒット率も平均 90%との結果がえられた。

以上より、パラメータの調整項目として、MaxDB が使用する CPU の数をハードウェアに合わせる。IO Wait は DB の遅延ではなく、アプリケーションから DB への接続待ちの状態となっているため、ここのパイプを太くすることで、IO Wait の解消を図る。

IOWait : DB 処理の「I/O」要求を出そうとするが、DB 側で要求を受け付ける余裕がないため、「I/O」リクエストがサスペンドしている状態。

表 3.5-1 SUSE IOWait の推移

ユーザ数	200	400	600	800
開始直後	82,038,956	119,366,717	137,972,442	31,605,098
2000 秒後	82,617,115	123,390,421	144,862,147	40,931,113
差	578,159	4,023,704	6,889,705	9,326,015

表 3.5-2 MIRACLE LINUX IOWait の推移

ユーザ数	200	400	600	800
開始直後	25,848,088	78,379,984	129,030,838	168,204,782
2000 秒後	27,179,121	84,530,311	136,437,359	176,590,371
差	1,331,033	6,150,327	7,406,521	8,385,589

表 3.5-3 RedHatAS IOWait の推移

ユーザ数	200	400	600	800
開始直後	17,356	2,163,196	371,630	14,939,457
2000 秒後	993,014	6,703,183	9,170,782	24,026,501
差	975,658	4,539,987	8,799,152	9,087,044

x\_cons\*.out IOWait:MaxDB 付属のコマンド。DB が使用している全システムリソース情報がえられる。

表 3.5-4 SESU キャッシュヒット率の推移

ユーザ数	200	400	600	800
DATA(%)	87	88	88	90
CATALOG(%)	93	93	93	93

表 3.5-5 MIRACLE LINUX キャッシュヒット率の推移

ユーザ数	200	400	600	800
DATA(%)	87	88	88	88
CATALOG(%)	93	93	93	93

表 3.5-6 RedHatAS キャッシュヒット率の推移

ユーザ数	200	400	600	800
DATA(%)	86	87	88	88
CATALOG(%)	93	93	93	93

m\_cache\*.out:MaxDB のキャッシュ情報

表 3.5-7 SUSE 使用メモリの推移

ユーザ数	200	400	600	800
メモリ使用率(%)	99.70	99.70	99.69	99.71
平均使用メモリ(MB)	618	597	737	694

表 3.5-8 MIRACLE LINUX 使用メモリの推移

ユーザ数	200	400	600	800
メモリ使用率(%)	99.54	99.52	99.56	99.50
平均使用メモリ(MB)	721	752	778	801

表 3.5-9 RedHatAS 使用メモリの推移

ユーザ数	200	400	600	800
メモリ使用率(%)	41.66	51.74	46.01	70.90
平均使用メモリ(MB)	423	464	473	503

平均使用メモリ：memory.txt より算出

全使用メモリ(kbmemused)-システムキャッシュ(kbcached)

以上の情報より、より高い DB 性能を得るための方向性を、次の 4 つに絞って検討した。

### 3.5.2.1 DB 接続数の拡張

ログより、IOWait となっているユーザタスクが多く発生していることが判明した。これは、appServer 側でキュー待ちの状態となっていると考えられる。チューニング前の設定では、appServer と MaxDB の間には、20 本の接続しか設定していないため、より多くの接続を確立するよう設定する。最大値は、MaxDB の MAXUSERTASKS の設定に依存する。また、この MAXUSERTASKS は、値 1 に対して 4MB の領域を必要とするため、現実の最大値はおよそ 200 となる。この値は、DB 生成時に指定済みである。DB 接続数を拡張することの効果を確認するため、20 から 100 に拡張して試験を実施する。

表 3.5-10 DB 接続数の変更

パラメータ名	設定値チューニング前	設定値チューニング後
dbt1.config dbconnection	20	100

### 3.5.2.2 最大使用 CPU 数の拡張

ハードウェアスペックに応じて、MaxDB 定義内の最大利用 CPU 数を変更する。今回使用したハードウェアは、2CPU(ハイパースレッド有効)構成であるので、見かけ上 CPU は 4 個に見える。そのため、MaxDB のパラメータ MAXCPU を 4 にセットするのが、リソースを最も有効に利用できる値となる。

表 3.5-11 MAXCPU の変更

パラメータ名	設定値チューニング前	設定値チューニング後
MAXCPU	1	4

### 3.5.2.3 MaxDB キャッシュの拡張

MaxDB 内に保持するキャッシュエリアを拡張した。チューニング前では、データキャッシュは 88%前後で推移していたが、MaxDB では 99%以上が推奨値であるため、容量を拡大し、より多くの情報をキャッシュに格納出来るよう設定する。

表 3.5-12 キャッシュサイズの変更

パラメータ名	設定値チューニング前	設定値チューニング後
CACHE_SIZE	10000	50000
CAT_CACHE_SUPPLY	6432	25000

値 1 に付き 8KB が確保される

### 3.5.2.4 データボリュームの分割

MaxDB の I/O 処理は、DISK 構成に関係なく、データボリュームの数に依存して並列処理効率を向上させる仕組みとなっている。そのため、データボリュームを分割し、アクセスを分散することで、BT/秒の向上を図る。

### 3.5.3 チューニング後の測定結果

図 3.5-7 以降に、チューニング後の DBT-1 測定結果を掲載する。尚、効果が明確になるよう、チューニング前のデータも重ねて表示する。

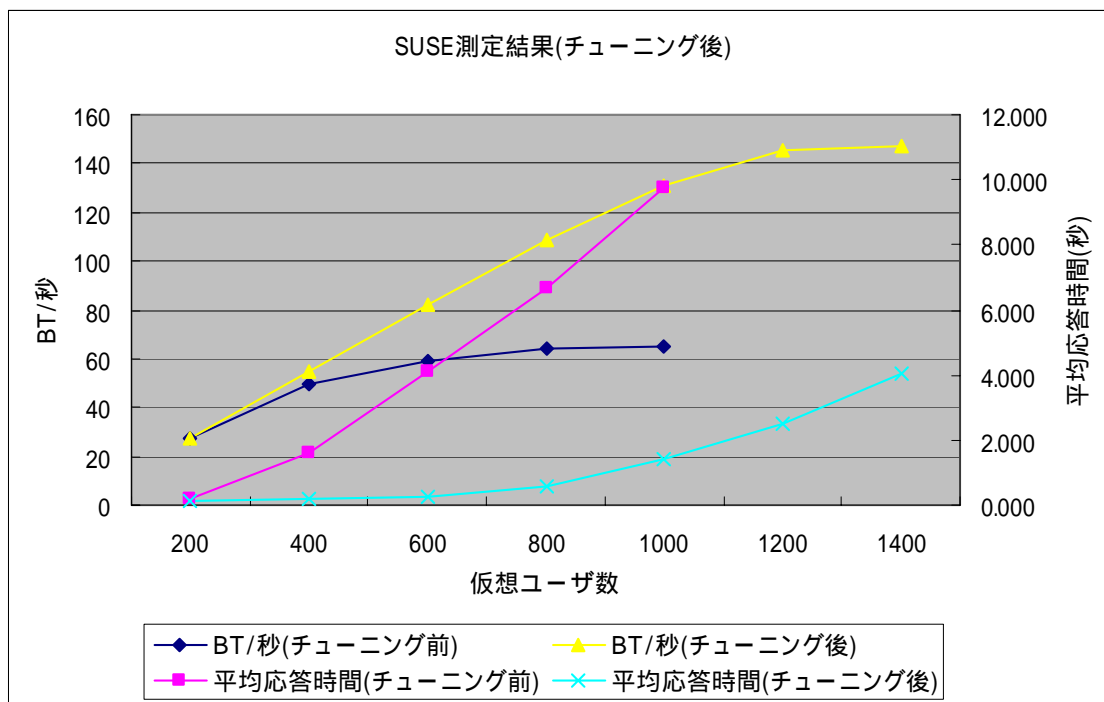


図 3.5-7 SUSE 測定結果(チューニング後)

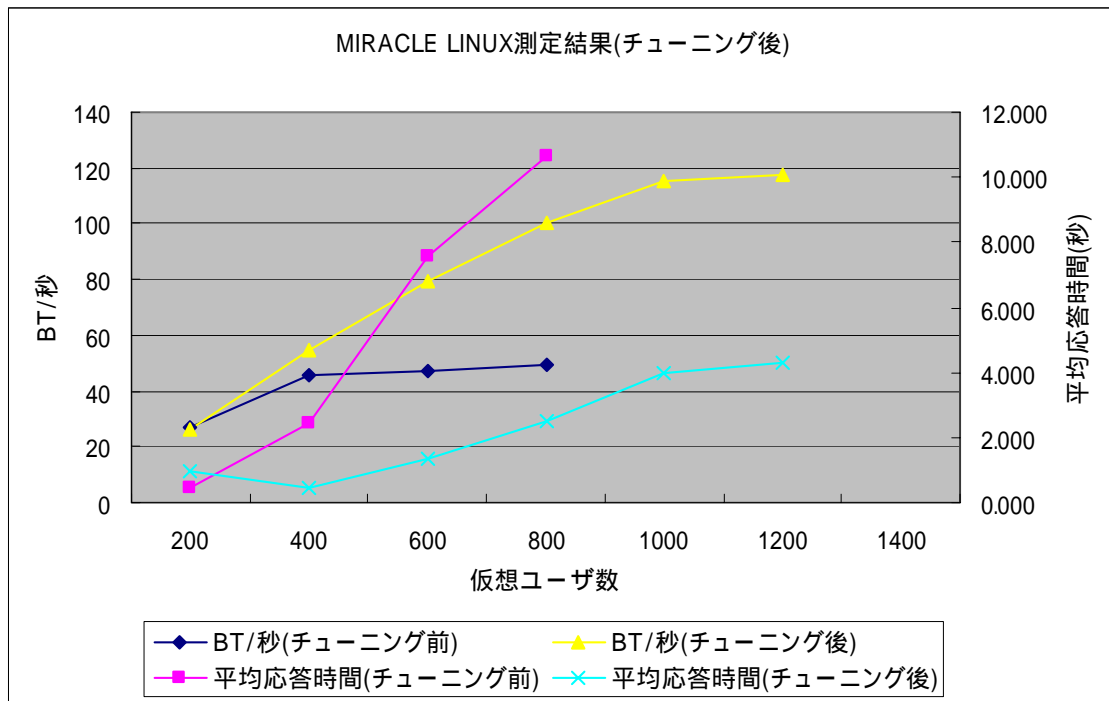


図 3.5-8 MIRACLE LINUX 測定結果(チューニング後)

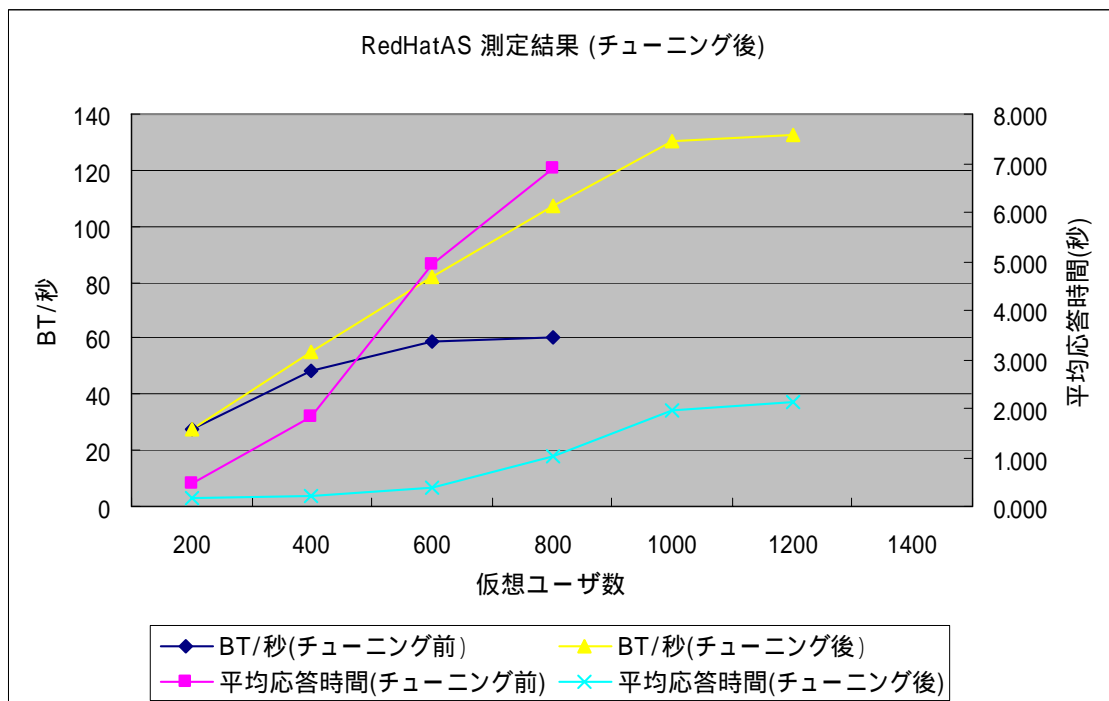


図 3.5-9 RedHat AS 測定結果(チューニング後)

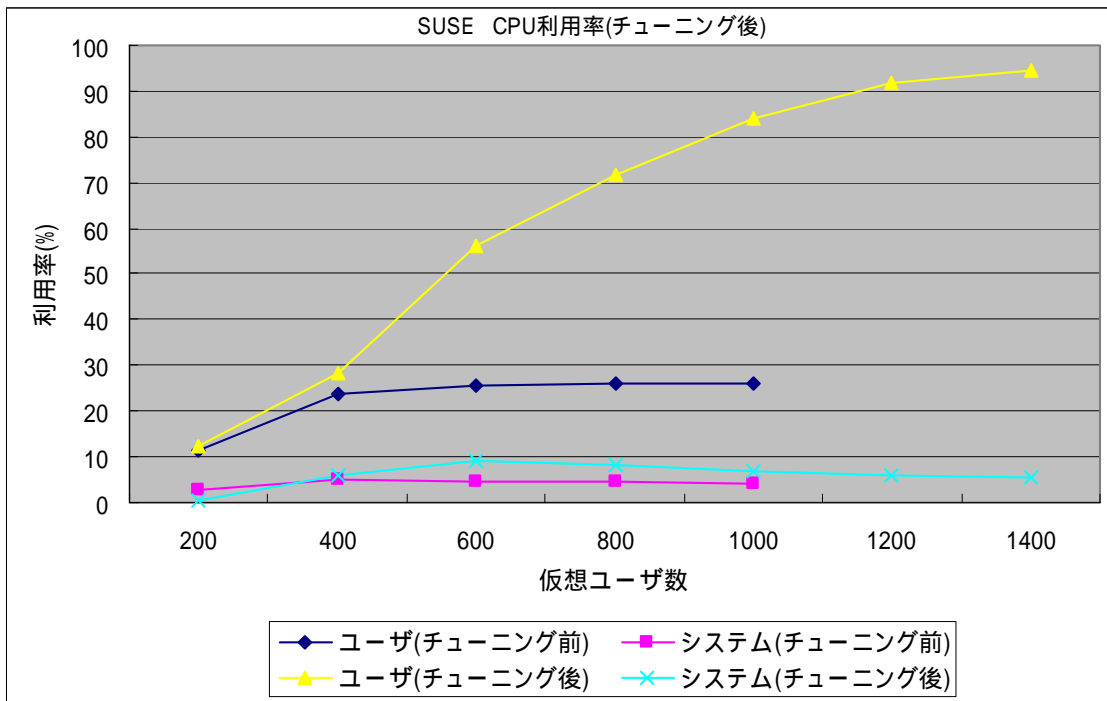


図 3.5-10 SUSE CPU 利用率(チューニング後)

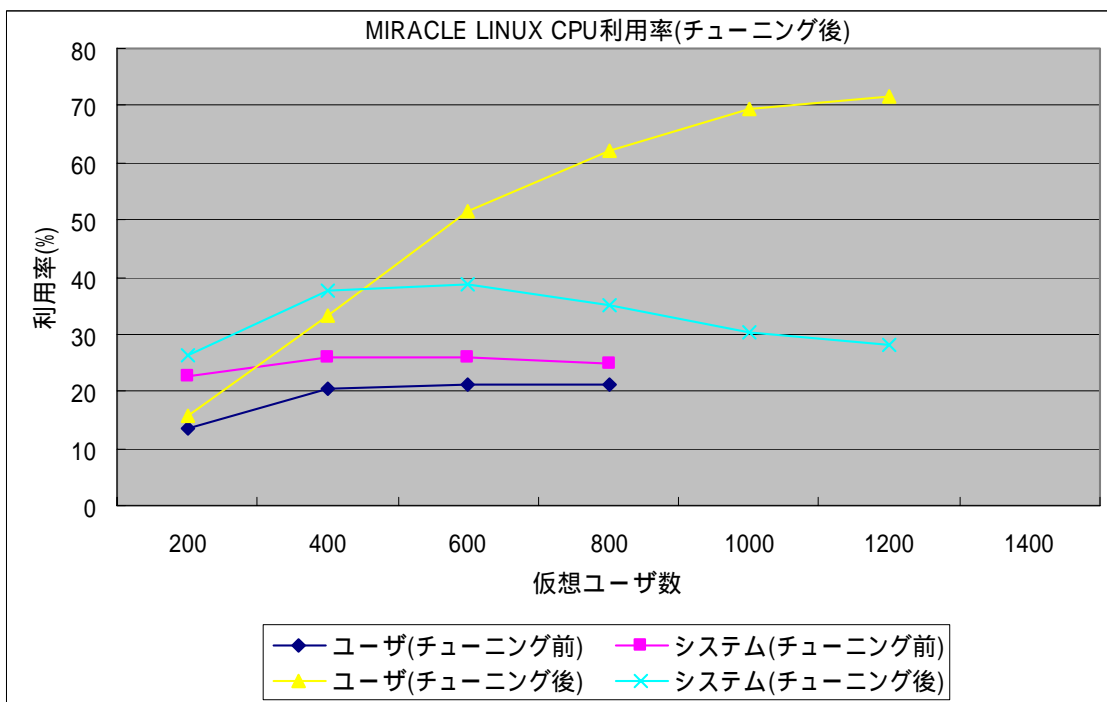


図 3.5-11 MIRACLE LINUX CPU 利用率(チューニング後)

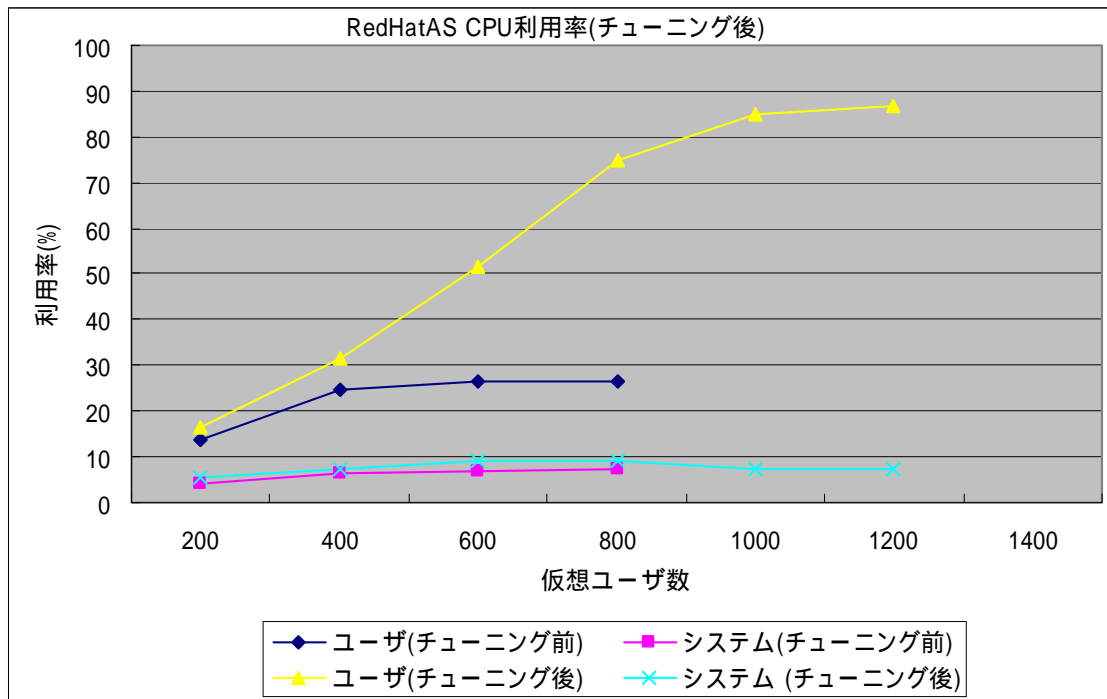


図 3.5-12 RedHatAS CPU 利用率(チューニング後)

### 3.5.4 チューニング後の結果の考察

仮想ユーザ数の増加に伴い、BT/秒は仮想ユーザ数 1000 までリニアな伸びを見せた。

CPU 数並びにデータボリュームの分割は、ハードウェアリソースに依存する項目なので、最大効率を引き出す値を明確に定義することは可能であるが、DB 接続数並びにキャッシュサイズは最適な値は、各アプリケーションによって異なるため、複数のパターンによって見極めることが肝要と思われる。

DBT-1 では、理論 BT/秒は仮想ユーザ数に応じてリニアに伸びる<sup>\*1</sup>ため、運用で想定する最大ユーザ数を超えるまで、リニアな伸びを示すよう DB パラメータを調整する必要がある。

\*1:ThinkTime7.2 秒の場合、100 ユーザ毎に約 13.9 トランザクション/秒ずつ増えることになる。理論上の BT/秒は次の式で表すことができる。

$$\text{BT/秒理論値} = \text{ユーザ数} / 100 * 13.9$$

どのディストリビューションにおいても、CPU 使用率は仮想ユーザ数 1000 以降で 90%以上になっている。SUSE のみ 1200 まで BT/秒が伸びた。但し、2.4 系カーネルを使用する MIRACLE LINUX 及び RedHatAS では、異なる振る舞いを見せた。

MIRACLE LINUX では、初回測定時の BT/秒が最大であり、以降連続して測定すると BT/秒は若干低下する傾向が見られた。この傾向は 3 回以降の計測では解消され、安定した計測値を得ることが出来た。

RedHatAS では、アップデートを施さないと計測に障害をきたす状況が発生した。尚、RedHatAS は最新のリリース(UPDATE4)ではこの障害は解消され、安定した計測を行うことができた。後掲する、表 3.5-23 ディストリビューション別 BT/秒に\* を付けた値は、最新リリースでの測定結果を示す。この値は、2005/02/17 時点で入手可能な最新バージョンである、Red Hat Enterprise Linux 3 Update 4 AS (x86) 2.4.21-27.ELsmp #1 SMP Wed Dec 1 21:59:02 EST 2004 を使用した。

### 3.5.4.1 DB 接続数の拡張による効果

接続数の違いによるインタラクションのレスポンスタイムの比較を行なう。

表 3.5-13 CONNECT 数の増減に伴う BT/秒、平均レスポンスの変化(RedHatAS)

< DB 接続数=20 >		< DB 接続数=100 >	
Interaction	Avg. Response Time (s)	Interaction	Avg. Response Time (s)
Admin Confirm	6.745	Admin Confirm	4.662
Admin Request	6.777	Admin Request	4.929
Best Sellers	6.860	Best Sellers	6.364
<b>Buy Confirm</b>	<b>18.827</b>	<b>Buy Confirm</b>	<b>66.046</b>
Buy Request	6.623	Buy Request	4.695
Customer Registration	0.000	Customer Registration	0.000
Home	6.527	Home	4.625
New Products	6.897	New Products	6.370
Order Display	6.433	Order Display	4.981
Order Inquiry	6.433	Order Inquiry	4.534
Product Detail	6.591	Product Detail	4.735
Search Request	0.000	Search Request	0.000
Search Results	6.978	Search Results	6.670
Shopping Cart	6.526	Shopping Cart	4.582
<b>63.9</b> bogotransactions per second		<b>65.7</b> bogotransactions per second	
62.4 minute duration		60.4 minute duration	
total bogotransactions 231545		total bogotransactions 237860	
total errors 0		total errors 0	

DB 接続数を拡張後、Buy Confirm 処理の平均応答時間が悪化した原因は、更新処理における同期処理の効率が悪くなったことによるものと考えられる。Buy Confirm 処理は複数の更新処理が実行されるため、DB 接続数を拡張したことにより待ち状態が多く発生し、平均レスポンスの増加が顕著に現れたと考えられる。

### 3.5.4.2 最大 CPU 数の拡張による効果

最大 CPU 数の変化による BT/秒の変化を比較する。

表 3.5-14 MAXCPU 変更後の BT/秒の推移(SUSE 仮想ユーザ数 800)

MAXCPU	1	4
BT/秒	63.0	108.6
平均ユーザ CPU (%)	25.85	70.13

表 3.5-15 MAXCPU 変更後の BT/秒の推移(MIRACLE LINUX 仮想ユーザ数 800)

MAXCPU	1	4
BT/秒	49.5	101.6
平均ユーザ CPU (%)	21.35	59.76

表 3.5-16 MAXCPU 変更後の BT/秒の推移(RedHatAS 仮想ユーザ数 800)

MAXCPU	1	4
BT/秒	62.6	107.4
平均ユーザ CPU (%)	27.03	74.71

表 3.5-14,15,16 MAXCPU 変更後の BT/秒の推移より、MAXCPU を 1 から 4 に変更した結果、2 倍程度 BT/秒が向上した。理論値よりは低いですが効果があったと思われる。

### 3.5.4.3 キャッシュ拡張の効果

キャッシュ拡大の効果を知るために SQL 処理数、I/O およびキャッシュヒット率を比較する。

表 3.5-17 SQL 当たりの I/O 数とヒット率(SUSE 仮想ユーザ数 800)

DATA キャッシュ	80MB	400MB
SQL 数	5,151,845	7,675,163
I/O 数	20,045,888	22,381,887
SQL 当たりの I/O 数	3.89	2.92
キャッシュヒット率(%)	90	92

表 3.5-18 SQL 当たりの I/O 数とヒット率(MIRACLE LINUX 仮想ユーザ数 800)

DATA キャッシュ	80MB	400MB
SQL 数	4,033,099	8,580,526
I/O 数	20,498,714	11,669,957
SQL 当たりの I/O 数	5.08	1.36
キャッシュヒット率(%)	88	95

表 3.5-19 SQL 当たりの I/O 数とヒット率(RedHatAS 仮想ユーザ数 800)

DATA キャッシュ	80MB	400MB
SQL 数	5,101,907	7,804,338
I/O 数	24,590,018	20,881,605
SQL 当たりの I/O 数	4.82	2.68
キャッシュヒット率(%)	88	93

表 3.5-17,18,19 SQL 当たりの I/O 数とヒット率より、1SQL 当たりの I/O 数が どのディストリビューションにおいても削減されており、キャッシュヒット率も数% の向上がみられる事から、I/O の削減に効果があったと考えられる。

m\_cache.out/m\_trans.out より

#### 3.5.4.4 データボリュームの分割による効果

DATA ボリュームを 8 分割し、IOWait 数を比較する。

表 3.5-20 DATA ボリューム数と IOWait、I/O 数、BT/秒(SUSE 仮想ユーザ数 800)

DATA ボリューム数	1	8
IOWait 数	2,923,215	4,837,303
SQL 数	7,675,163	8,022,077
I/O 数	22,381,887	25,153,131
SQL 数当たりの I/O 数	2.92	3.14
BT/秒	108.6	107.3

表 3.5-21 DATA ボリューム数と IOWait、I/O 数、BT/秒 MIRACLE LINUX 仮想ユーザ数 800)

DATA ボリューム数	1	8
IOWait 数	8,397,964	7,972,557
SQL 数	8,580,526	9,139,847
I/O 数	11,669,957	14,099,093
SQL 数当たりの I/O 数	1.36	1.54
BT/秒	101.6	100.4

表 3.5-22 DATA ボリューム数と IOWait、I/O 数、BT/秒(RedHatAS 仮想ユーザ数 800)

DATA ボリューム数	1	8
IOWait 数	5,145,414	4,156,375
SQL 数	7,804,338	7,994,388
I/O 数	20,881,605	22,959,852
SQL 数当たりの I/O 数	2.68	2.87
BT/秒	107.4	108.1

データボリュームの分割を行うと、並列処理の効果により、IOWait 数が減少し BT/秒が向上する事が期待された。表 3.5-20,21,22 DATA ボリューム数と IOWait サスペンド、I/O 数、BT/秒 より、MIRACLE LINUX 及び RedHatAS では、期待通り IOWait 数の減少が確認できたが、BT/秒が増加することは無かった。これは、並列処理により同時に要求されるデータ量が増加したため、I/O 数も増加し、応答時間にも効果が現れなかったと考えられる。SUSE も IOWait 数が増加した以外は、同様の傾向が見られた。IOWait 数はディストリビューション間の相違に起因することと推測も出来るが、原因の究明には至らなかった。この事は今後の調査課題としたい。

以上より、ボリューム分割は、I/O の並列処理が可能な環境でなければ、その効果は期待できないと判断できる。

結論として、ボリューム分割は DISK 数(コントローラの数)に合わせた数で構築することが最善と考えら、今回のように RAID0 で 1 つのファイルシステムとして構築された環境に於いて、有効なチューニング方法とはならない。

測定結果の表は、各条件にて5回測定した結果より、平均 BT/秒に近い値が得られたデータセットを基にしているが、各ディストリビューション間の差について考える意味で、各結果を表にまとめる。

表 3.5-23 ディストリビューション別 BT/秒

		200	400	600	800	1000	1200	1400
SUSE	最小(チューニング前)	27.4	48.6	56.6	61.0	61.3	-	-
	最大(チューニング前)	27.6	51.4	61.0	66.0	68.0	-	-
	平均(チューニング前)	27.44	49.80	58.66	63.42	64.87	-	-
	差(チューニング前)	0.2	2.8	4.4	5.0	6.7	-	-
	最小(チューニング後)	27.5	55.1	82.0	108.3	125.7	140.9	144.9
	最大(チューニング後)	27.6	55.3	82.5	109.0	133.9	146.6	152.3
	平均(チューニング後)	27.58	55.16	82.26	108.56	130.68	144.16	147.4
	差(チューニング後)	0.1	0.2	0.5	1.3	8.2	5.7	7.4
MIRACLE LINUX	最小(チューニング前)	26.5	42.7	44.5	47.3	-	-	-
	最大(チューニング前)	27.3	46.8	50.3	52.2	-	-	-
	平均(チューニング前)		44.88	47.14	49.54	-	-	-
	差(チューニング前)	0.8	4.1	6.4	4.9	-	-	-
	最小(チューニング後)	27.3	54.2	79.7	100.1	111.0	113.9	-
	最大(チューニング後)	27.5	54.4	80.5	104.0	129.0	131.7	-
	平均(チューニング後)		54.30	80.14	101.94	116.14	119.14	-
	差(チューニング後)	3.2	0.2	0.8	3.9	19.0	17.8	-
RedHat AS	最小(チューニング前)	27.2	47.5	52.7	60.6	-	-	-
	最大(チューニング前)	27.4	50.3	56.9	63.1	-	-	-
	平均(チューニング前)	27.32	48.60	56.24	62.10	-	-	-
	差(チューニング前)	0.2	2.8	4.2	2.5	-	-	-
	最小(チューニング後)	27.4	54.8	81.7	107.2	130.0	130.0	-
	最大(チューニング後)	27.6	55.2	82.2	107.6	131.5*	132.6*	-
	平均(チューニング後)	27.50	54.96	81.92	107.40	130.60	131.50	-
	差(チューニング後)	0.2	0.4	0.5	0.4	1.5	2.6	-

MIRACLE LINUX において、1000 ユーザ以上のケースで BT/秒の差が大きいのは、システムリブート直後の測定結果で最大値を示し、継続して測定した場合、BT/秒が低下したことに起因する。

RedHatAS チューニング後の仮想ユーザ数 1000 及び 1200 は、Red Hat Enterprise Linux 3 Update 4 AS (x86) 2.4.21-27.ELsmp #1 SMP Wed Dec 1 21:59:02 EST 2004 による試験結果である。

### 3.5.5 インタラクションにおける性能の違い

各インタラクションの平均応答時間をグラフ化し、チューニングの前後及びディストリビューション間の違いを検証する。

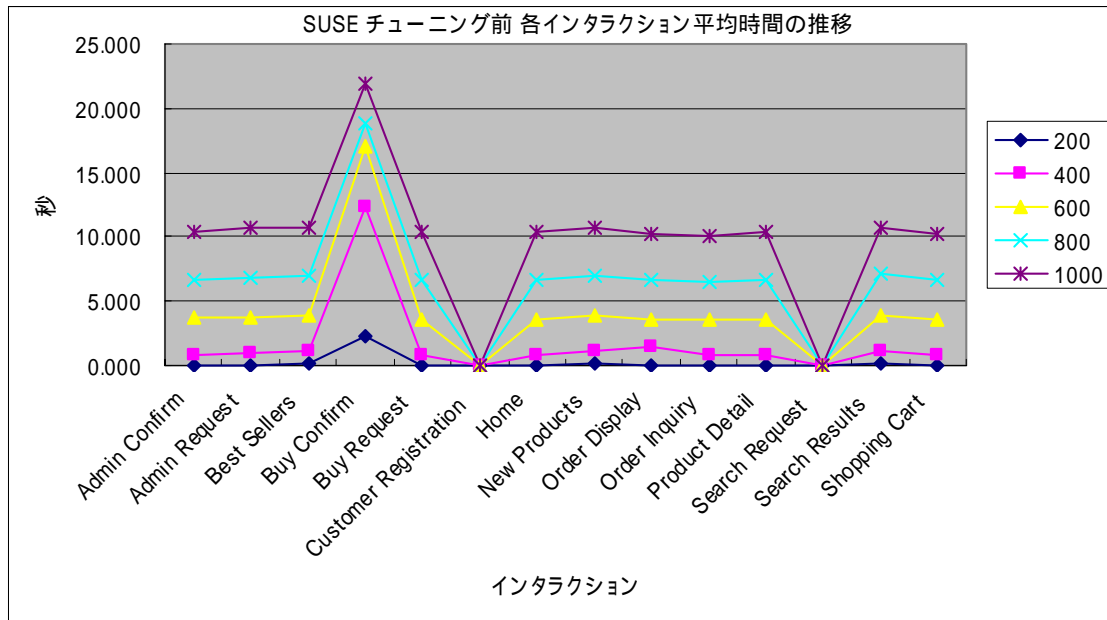


図 3.5-13 SUSE チューニング前 各インタラクション平均時間の推移

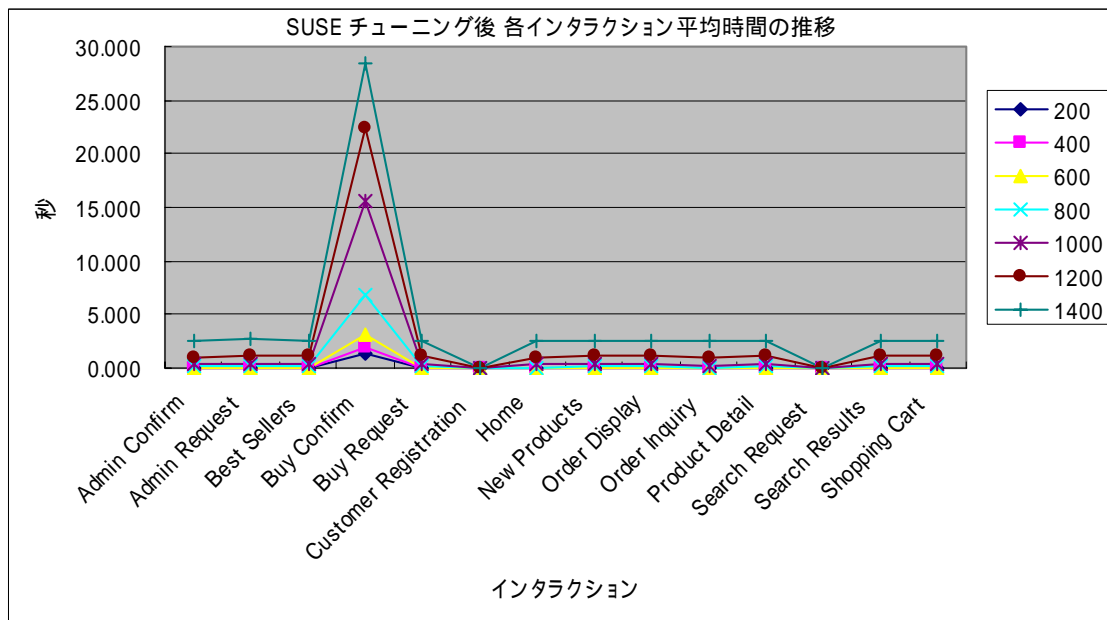


図 3.5-14 SUSE チューニング後 各インタラクション平均時間の推移

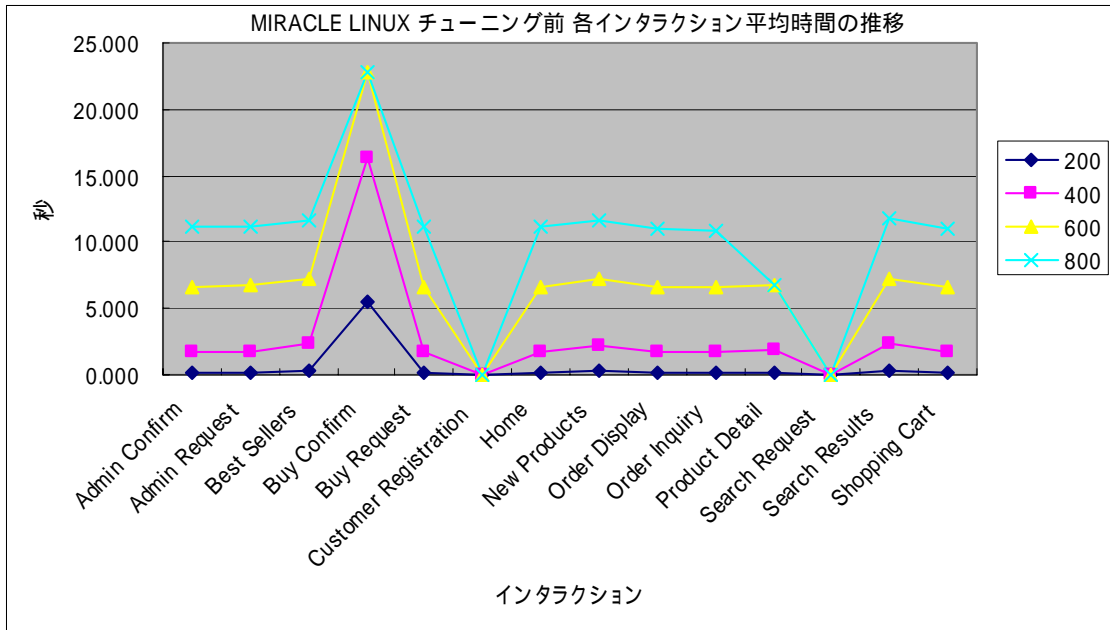


図 3.5-15 MIRACLE LINUX チューニング前 各インタラクション平均時間の推移

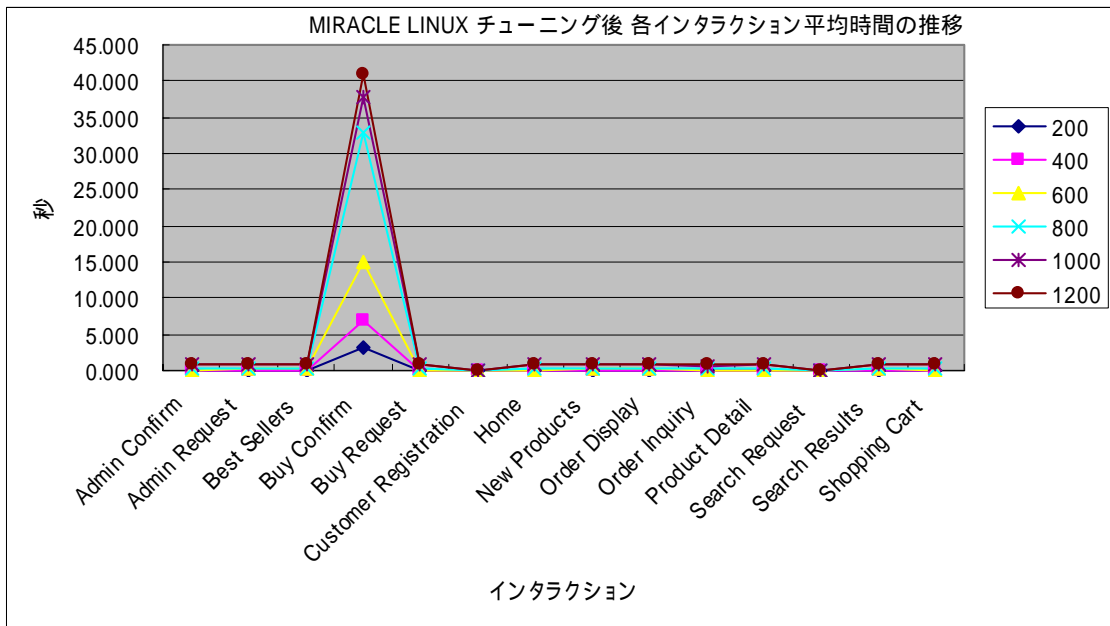


図 3.5-16 MIRACLE LINUX チューニング後 各インタラクション平均時間の推移

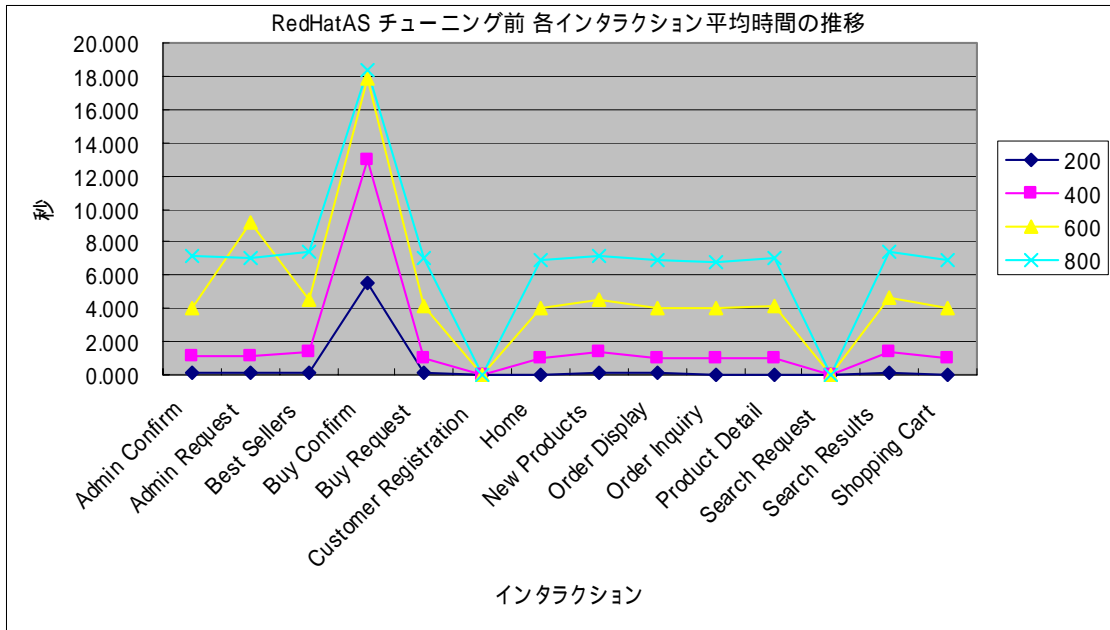


図 3.5-17 RedHatAS チューニング前 各インタラクション平均時間の推移

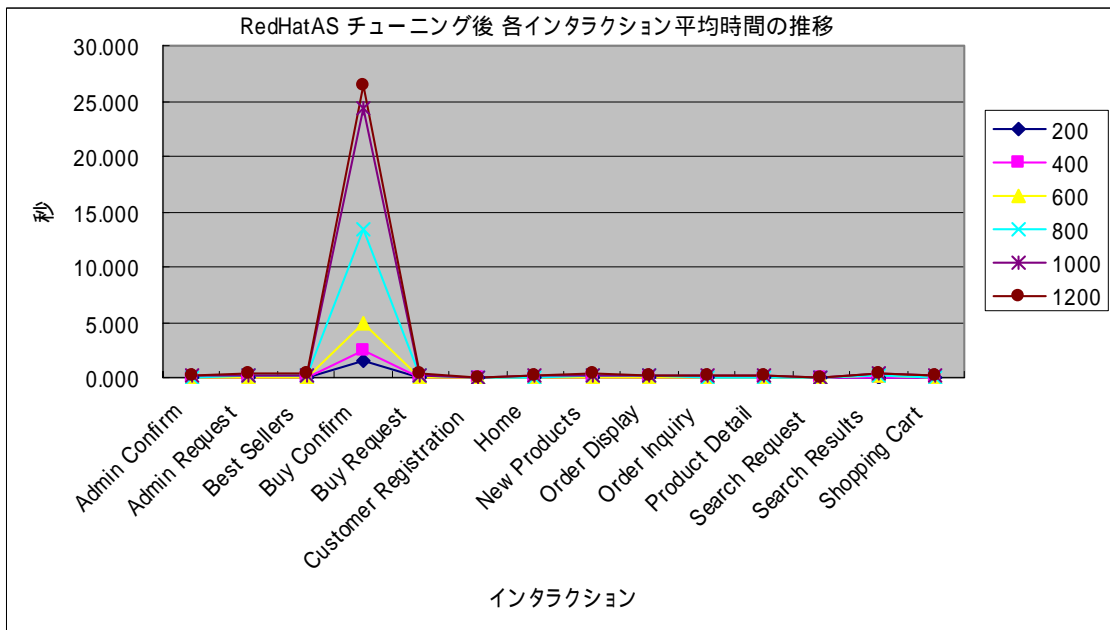


図 3.5-18 RedHatAS チューニング後 各インタラクション平均時間の推移

Buy Confirm を除く他の処理は、チューニング前後とも、ほぼ同じ応答時間となっている。また、チューニング後では、どのディストリビューションにおいても、Buy Confirm に比べ十分小さな値を示すことが見て取れる。

Buy Confirm 処理が突出して高い値を示すのは、orders テーブルに INSERT を行った後、sc\_id で関連付けられた shopping\_cart\_line テーブルのデータを全て読み、

そのデータ分、order\_line テーブルへの INSERT と、item テーブルへ UPDATE を行っているため、他のインタラクションより処理コストを要する事に起因する。

その他の処理は、インタラクションに関係なく処理時間は同じ値となっている。

### 3.5.6 計測データ

#### 3.5.6.1 インタラクション平均時間(SUSE)

表 3.5-24 SUSE インタラクション平均値(参考)

		200	400	600	800	1000	1200	1400
チューニング前	Admin Confirm	0.057	0.828	3.763	6.695	10.383	—	—
	Admin Request	0.068	0.929	3.771	6.876	10.679	—	—
	Best Sellers	0.117	1.085	3.915	7.030	10.689	—	—
	Buy Confirm	2.198	12.286	17.071	18.802	21.953	—	—
	Buy Request	0.057	0.809	3.585	6.666	10.321	—	—
	Customer Registration	0.000	0.000	0.000	0.000	0.000	—	—
	Home	0.042	0.789	3.538	6.647	10.323	—	—
	New Products	0.117	1.091	3.946	7.061	10.708	—	—
	Order Display	0.067	1.469	3.598	6.579	10.306	—	—
	Order Inquiry	0.050	0.754	3.515	6.512	10.128	—	—
	Product Detail	0.045	0.824	3.596	6.712	10.361	—	—
	Search Request	0.000	0.000	0.000	0.000	0.000	—	—
	Search Results	0.101	1.072	3.962	7.110	10.757	—	—
	Shopping Cart	0.048	0.781	3.521	6.612	10.271	—	—
BT/秒	27.44	49.80	58.66	63.42	64.87	—	—	
チューニング後		200	400	600	800	1000	1200	1400
	Admin Confirm	0.054	0.057	0.067	0.111	0.311	1.057	2.561
	Admin Request	0.058	0.061	0.078	0.141	0.389	1.167	2.701
	Best Sellers	0.063	0.069	0.085	0.144	0.370	1.137	2.577
	Buy Confirm	1.284	1.857	3.180	6.776	15.609	22.439	28.494
	Buy Request	0.052	0.056	0.053	0.117	0.330	1.102	2.520
	Customer Registration	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Home	0.042	0.042	0.048	0.090	0.295	1.063	2.495
	New Products	0.063	0.069	0.085	0.117	0.365	1.135	2.577
	Order Display	0.064	0.069	0.083	0.132	0.348	1.144	2.589
	Order Inquiry	0.042	0.045	0.052	0.092	0.287	1.069	2.483
	Product Detail	0.046	0.045	0.050	0.100	0.317	1.087	2.524
	Search Request	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Search Results	0.068	0.060	0.092	0.155	0.388	1.168	2.613
Shopping Cart	0.046	0.049	0.057	0.105	0.323	1.095	2.530	
BT/秒	27.58	55.16	82.26	108.56	130.68	144.16	147.40	

### 3.5.6.2 インタラクション平均時間(MIRACLE LINUX)

表 3.5-25 MIRACLE LINUX インタラクション平均値(参考)

		200	400	600	800	1000	1200	1400
チューニング前	Admin Confirm	0.09	1.69	6.641	11.129	—	—	—
	Admin Request	0.13	1.76	6.798	11.140	—	—	—
	Best Sellers	0.28	2.29	7.237	11.614	—	—	—
	Buy Confirm	5.55	16.2	22.778	22.778	—	—	—
	Buy Request	0.10	1.80	6.660	11.125	—	—	—
	Customer Registration	0.00	0.00	0.000	0.000	—	—	—
	Home	0.07	1.79	6.670	11.115	—	—	—
	New Products	0.27	2.27	7.213	11.651	—	—	—
	Order Display	0.11	1.76	6.662	11.027	—	—	—
	Order Inquiry	0.08	1.72	6.557	10.794	—	—	—
	Product Detail	0.09	1.87	6.764	6.764	—	—	—
	Search Request	0.00	0.00	0.000	0.000	—	—	—
	Search Results	0.24	2.29	7.296	11.755	—	—	—
	Shopping Cart	0.08	1.77	6.628	11.061	—	—	—
	BT/秒	27.0	44.8	47.14	49.54	—	—	—
チューニング後		200	400	600	800	1000	1200	1400
	Admin Confirm	0.05	0.07	0.120	0.290	0.774	0.819	-
	Admin Request	0.06	0.09	0.150	0.315	0.791	0.873	-
	Best Sellers	0.08	0.10	0.155	0.308	0.784	0.845	-
	Buy Confirm	3.15	6.81	14.891	32.852	37.645	41.080	-
	Buy Request	0.06	0.08	0.131	0.282	0.822	0.920	-
	Customer Registration	0.00	0.00	0.000	0.000	0.000	0.000	-
	Home	0.04	0.05	0.084	0.216	0.722	0.796	-
	New Products	0.08	0.10	0.155	0.311	0.791	0.849	-
	Order Display	0.07	0.09	0.157	0.306	0.793	0.856	-
	Order Inquiry	0.04	0.05	0.088	0.209	0.661	0.806	-
	Product Detail	0.04	0.05	0.101	0.249	0.756	0.818	-
	Search Request	0.00	0.00	0.000	0.000	0.000	0.000	-
	Search Results	0.08	0.11	0.169	0.339	0.882	0.967	-
	Shopping Cart	0.05	0.06	0.105	0.242	0.739	0.806	-
BT/秒	27.4	54.3	80.14	101.94	120.76	121.54	-	

### 3.5.6.3 インタラクション平均時間(RedHatAS)

表 3.5-26 RedHatAS インタラクション平均値(参考)

		200	400	600	800	1000	1200	1400
チュ ー ニ ン グ 前	Admin Confirm	0.064	1.150	4.004	7.134	—	—	—
	Admin Request	0.075	1.104	9.192	7.047	—	—	—
	Best Sellers	0.169	1.352	4.553	7.392	—	—	—
	Buy Confirm	5.548	12.993	17.857	18.371	—	—	—
	Buy Request	0.064	1.024	4.092	7.018	—	—	—
	Customer Registration	0.000	0.000	0.000	0.000	—	—	—
	Home	0.043	0.985	4.078	6.962	—	—	—
	New Products	0.171	1.362	4.525	7.210	—	—	—
	Order Display	0.081	1.023	4.083	6.902	—	—	—
	Order Inquiry	0.048	0.992	4.011	6.815	—	—	—
	Product Detail	0.049	1.046	4.154	7.037	—	—	—
	Search Request	0.000	0.000	0.000	0.000	—	—	—
	Search Results	0.139	1.354	4.607	7.463	—	—	—
	Shopping Cart	0.050	0.981	4.037	6.921	—	—	—
	BT/秒	27.32	48.60	56.24	62.10	—	—	—
		200	400	600	800	1000	1200	1400
チュ ー ニ ン グ 後	Admin Confirm	0.057	0.060	0.065	0.092	0.228	0.267	—
	Admin Request	0.056	0.064	0.074	0.112	0.277	0.304	—
	Best Sellers	0.064	0.074	0.088	0.120	0.282	0.309	—
	Buy Confirm	1.570	2.529	4.845	13.399	24.359	26.394	—
	Buy Request	0.057	0.061	0.071	0.097	0.254	0.284	—
	Customer Registration	0.000	0.000	0.000	0.000	0.000	0.000	—
	Home	0.041	0.042	0.047	0.064	0.215	0.243	—
	New Products	0.064	0.074	0.088	0.120	0.281	0.310	—
	Order Display	0.055	0.063	0.076	0.108	0.264	0.278	—
	Order Inquiry	0.042	0.046	0.052	0.070	0.214	0.234	—
	Product Detail	0.045	0.043	0.048	0.073	0.236	0.265	—
	Search Request	0.000	0.000	0.000	0.000	0.000	0.000	—
	Search Results	0.070	0.080	0.095	0.129	0.313	0.347	—
	Shopping Cart	0.047	0.050	0.058	0.079	0.230	0.257	—
	BT/秒	27.50	54.96	81.92	107.40	131.10	131.06	—

### 3.5.6.4 ThinkTime と仮想ユーザ数(参考)

当初十分な仮想ユーザ数で測定を行えなかった際、ThinkTime を短くすることで、仮想ユーザ数を増やした場合と同様の結果が得られるものと想定し、ThinkTime を短くして測定を実施した。仮想ユーザ数 / ThinkTime = 理論 BT/秒 が成り立つことから、BT/秒が同じであれば負荷も同じと想定したからである。表 3.5-27 理論 BT/秒参照。

表 3.5-27 理論 BT/秒

仮想ユーザ数	ThinkTime	理論 BT/秒
200	3.6	55.6
400	7.2	55.6

仮想ユーザ数を十分増やせるようになった後、この仮説の検証を行った。図 3.5-19 は、SUSE にて ThinkTime3.6 秒と 7.2 秒で測定した BT/秒の推移をグラフ化したものである。また、ThinkTime3.6 の仮想ユーザ数を 2 倍にしものを仮想 ThinkTime7.2、ThinkTime7.2 の仮想ユーザ数を半分にしたものを仮想 ThinkTime3.6 とした。

ThinkTime3.6 と仮想 ThinkTime3.6 及び、ThinkTime7.2 と仮想 ThinkTime7.2 をそれぞれ比較した場合、BT/秒のピークは同じ値であるが、ThinkTime が短い方が、性能低下が早く発生した。結果として仮説を証明することは出来なかった。ThinkTime と負荷の相関関係に関しては、今後の調査課題としたい。

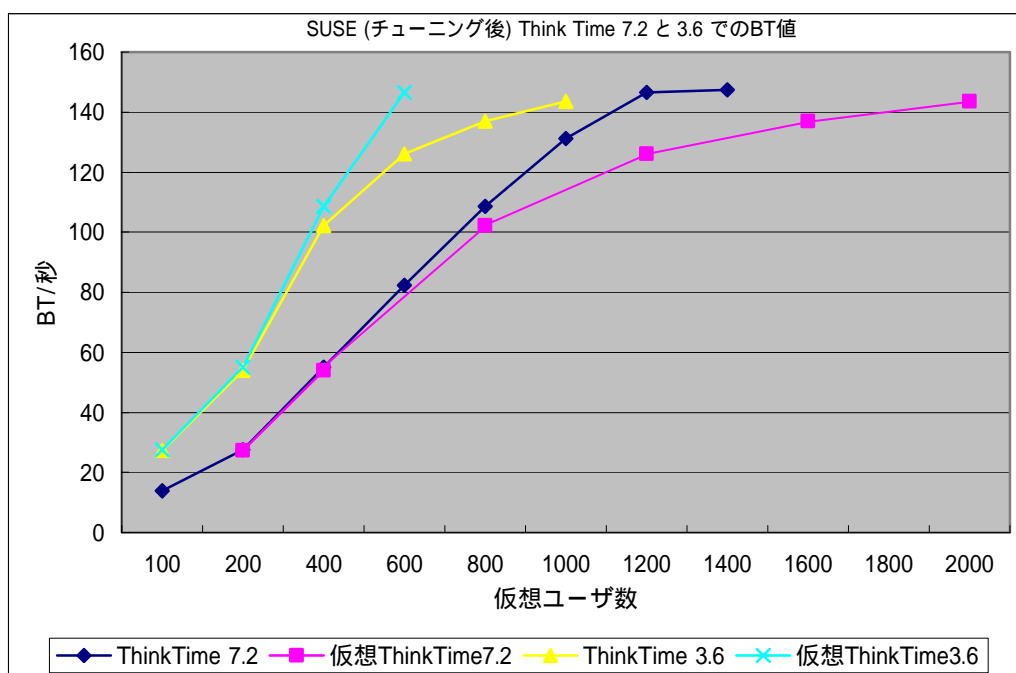


図 3.5-19 ThinkTime3.6 と 7.2 における BT/秒の推移

## 3.6 まとめ

セマフォ不足並びに、オープン可能なファイル数が不足したため、システム設定を拡張した。これは、パフォーマンスを上げるためでは無く、MaxDB の性能限界を確認するための設定である。

また、RedHatAS では、計測に支障をきたす事態が発生した事があった。最新版に更新する事で回避することは出来たが、このような場合の原因追求に関しては、今後の調査課題としたい。

DBT-1 に関しては、TPC-W に規定されている、HTML やイメージファイルの取得が考慮されていない事や、dbdriver と appServer 間でセッションを張り続ける等、Web アプリケーションの性能評価ツールと位置づけるには不十分な実装ではあるが、仮想ユーザ数に対する DB の負荷状況を確認する用途には適していると考ええる。

DBT-1 の負荷特性に対応するよう MaxDB をチューニングすることで、MaxDB のチューニングのためのノウハウを習得する事が出来た。

今回の DBT-1 による MaxDB の評価では、SAPDB7.3,7.4 以降のメンテナンスが中断されていた状態であった DBT-1 を、SAP DB の最新版である MaxDB7.5 に対応させる事ができた。また、ストアードプロシージャを使用しない、SQL による DB アクセスを実現することが、より汎用性の高い有用なプロダクトになるとの確信を得た。

## 4 DBT-1 による PostgreSQL の評価

### 4.1 概要

「あるオープンソース RDBMS の機能・性能・拡張性・信頼性はどのようなものか？」という質問のうち、性能面、もう少し範囲を特定しておくとして、Web トランザクション系のアプリケーションの性能面での PostgreSQL7.4.6 の評価を行った。評価には、OSDL が開発した DBT-1 ベンチマークを利用した。本章では、DBT-1 を PostgreSQL へ適用する際のひとつの手順を明らかにするとともに、Web エンドユーザの接続数の違いによる性能特性、データベースサイズによる性能特性、簡単なチューニングの効果、さらには、最近の Linux OS ディストリビューション間による性能特性について検討する。

### 4.2 環境定義

#### 4.2.1 システム構成

今回の評価では、以下の環境を利用した。システム構成は、表 4.2-1 の通りである。

表 4.2-1 ハードウェア

製品名	Dell PowerEdge4600
プロセッサ	インテル Xeon プロセッサ 2.40GHz、2CPU
メモリ	12Gbyte(ただし grub.conf で 4GB に制限している)
ハードディスク	内蔵ドライブベイに、UltraSCSI320 72GB 8 台

PostgreSQL8.0 では、データベースクラスタを複数のハードディスクに割り当て、テーブル毎にテーブルのデータ・インデックスを分散させるテーブルスペース機能を使用することができるが、7.4.x にはこの機能がないため、表 4.2-2 のような単純な構成をとった。今回は特にディスクのベンチマーク測定は行っていないが、一般に RAID0 では、性能的には乱暴な目安であるが、シーケンシャル Read/Write の速度は 60MB/s \* ディスクの本数あるいは、Raid ボードの最大転送速度の低いほう、ランダム Read/Write は 5MB/s \* ディスクの本数の性能が得られる可能性がある。

表 4.2-2 ディスク構成

72GB	/	20GB
	Swap	2GB
72GB	/work (ロード用データ等)	すべて
72GB*6 本 RAID0	/data	すべて

## 4.2.2 インストール

### 4.2.2.1 Linux のインストール

本章で対象にする PostgreSQL は、Linux のさまざまなディストリビューションで稼働実績がある。Linux のインストールは、各ディストリビューションのマニュアルを参照されたい。注意点のいくつかを下記に列挙する。

- PostgreSQL のバージョン

DBT-1 測定者が測定したい PostgreSQL のバージョンをインストールする必要がある。現在多くのディストリビューションが標準の RDBMS サーバとして、PostgreSQL をバンドルしている。バンドルされた PostgreSQL のバージョンが評価対象ならば、Linux インストール時に、インストール対象パッケージとして、PostgreSQL を選択されたい。そうでないならば、インストール対象から外し、後に述べる PostgreSQL のソースからのインストールを行う。尚、本ドキュメントでは、後者の方法を前提として記述している。

- sysstat のインストール

sysstat は、システムの稼働統計を取得するための一連のツール群である。ディストリビューションによっては sysstat はデフォルトでインストールされないため、Linux インストール時にインストールしておく必要がある。

- ファイルシステムの選択

現在の Linux の場合、ファイルシステムとして、ext3、ReiserFS、XFS などのファイルシステムが選択肢としてある。どれを選択するかはユーザが決定するマターである（DBT-1 や PostgreSQL はこの選択に依存しない）。

尚、ディストリビューションがサポートしているファイルシステムはディストリビューションにより異なる。各ファイルシステムはそれぞれの設計思想があり、特性も異なる。RDBMS を動作させるファイルシステムとしてどれが適当か、という点についての体系的な比較検討研究はまだ少ない。最も利用されているファイルシステムはおそらく ext3 であり、本章で扱うディストリビューションがすべてサポートするファイルシステムは ext3 のみであるので、本章の測定では ext3 のデフォルトモード(ordered)を選択している。

尚、本章で測定に用いた OS は、以下の三種である。

- SUSE LINUX Enterprise Server 9
  - Linux 2.6.5-7.97-default SMP #1 SMP
- MIRACLE LINUX V3.0 Asianux Inside
  - Linux 2.4.21-9.30AXsmp #1 SMP
- Red Hat Enterprise Linux AS 3
  - Linux 2.4.21-4.ELsmp #1 SMP

今後、それぞれ、SuSE、Miracle、RedHat と省略する。

#### 4.2.2.2 PostgreSQL のインストール

PostgreSQL は、ここではソースからインストールする。一般的なインストール方法と異ならないが、ここでは PostgreSQL のスーパーユーザ名などを以下で指定している。第 6 章 DBT-3 による PostgreSQL の評価の 6.2 の手順と同一で重複するが、手順は以下の通りである。

今回の測定開始時点で、安定版はバージョン 7.4.6 であったため、測定については 7.4.6 を採用する。尚、DBT-1 は、少なくとも 7.4.x では動作が確認されている模様である。

root ユーザで、PostgreSQL の所有者となる Linux ユーザとして、pgsql ユーザと pgsql グループを作成する。

```
# groupadd pgsql
# useradd pgsql -g pgsql
```

PostgreSQL のソースコードアーカイブを展開するディレクトリと、PostgreSQL をインストールするディレクトリを作成して、ディレクトリの所有者を pgsql ユーザにする。

```
# mkdir /usr/local/src/postgresql-7.4.6
# chown pgsql /usr/local/src/postgresql-7.4.6
# mkdir /usr/local/pgsql
# chown pgsql /usr/local/pgsql
```

pgsql ユーザで、PostgreSQL のソースコードアーカイブを展開し、展開したディレクトリに移動する。PostgreSQL のソースコードアーカイブは、/tmp ディレクトリにあるものとする。

```
# su - pgsql
$ cd /usr/local/src
$ tar xzf /tmp/postgresql-7.4.6.tar.gz
$ cd postgresql-7.4.6
```

展開した PostgreSQL のソースコードをコンパイルし、インストールする。

```
$ ./configure
$ make all
$ make install
```

#### 4.2.2.3 DBT-1 のインストール

本章では、測定開始時点で取得可能であった、DBT-1 の最新版である DBT-1 v2.1 を採用した。PostgreSQL は DBT-1 がサポートしている DBMS であるが、インストール部分にいくつかの不具合があり、今回は以下の手順で行った。

##### 4.2.2.3.1 前提条件

- autoconf の version が 2.59 以降であることが必要。autoconf のバージョンを確認するためには下記をコマンドを実行すること。2.58 以前の場合は、autoconf をアインストール後、autoconf ソースを入手して、インストールする必要がある。

```
$ autoconf -V
```

- PostgreSQL がインストールしてある。
- Linux ユーザ pgsqll が必要。pgsqll ユーザは PostgreSQL のスーパーユーザでなければならない。
- PostgreSQL のデータベースクラスタ作成はしなくても良い。(データベースクラスタを作成していない場合にはスクリプトの中で自動的に実行される。)
- DBT1 のソースコード(dbt1-v2.1.tar.gz)を入手している。  
入手先 <http://sourceforge.net/projects/osdldb1>

##### 4.2.2.3.2 インストールの手順

以下のことを想定して、DBT-1 をインストールする前提条件を説明する。

- 取得したファイルは/tmp に置いてある。
- PostgreSQL は/usr/local/pgsqll にインストールしてある。
- データベースクラスタは/data/DBT1 に作成する。
- DBT1 のインストール先を/home/pgsqll/dbt1-v2.1 とし、\$DBT1\_HOME と省略する

/tmp にダウンロードした DBT1 のソースコードを適切な場所にインストールする。このとき pgsqll というユーザで実行しなければならない。

##### 4.2.2.3.3 インストールの手順

1. はじめに、pgsqll ユーザのホームディレクトリに、DBT-1 のソースコードを展開するディレクトリを作成する。

```
$ su - pgsqll
$ cd ~
$ tar xzf /tmp/dbt1-v2.1.tar.gz
```

2. 環境変数の設定を行うために、設定ファイルを書き込み可能に変更する。

```
$ chmod +w $DBT1_HOME/scripts/pgsql/set_run_env.sh.in
```

3. テキストエディタで、set\_run\_env.sh.in を以下のように設定する。

```
export SID1=DBT1
export PATH=/usr/local/pgsql/bin:$PATH
export PGDATA=/data/DBT1
export PGUSER=pgsql
export DBT1_PERL_MODULE=@TOPDIR@/perlmodules
```

ここでユーザが変更すべき箇所は SID1、PATH、PGDATA である。SID1 は、作成するデータベース名、開発者は DBT1 を前提にテストをしているようなので、DBT1 が望ましい。PATH は、PostgreSQL のコマンドサーチパス。PGDATA は、データベースクラスタの格納場所を示す（今回はディスク構成の項で述べた通り、/data/DBT1 とした）。PGUSER は先にインストールした PostgreSQL のスーパーユーザである pgsql を入れてある。

4. スレッドのスタックサイズを特定するパッチをあてる。これは、オリジナルの DBT-1 では、現在の最新の OS ディストリビューションでは同時接続ユーザが 200 程度までしか増加できない問題への対応である。パッチ dbt1-v2.1-pgsqlpatch.tar.gz を解凍、2 つのソースファイルにパッチをあてる。

```
$ tar xzf /tmp/dbt1-v2.1-pgsqlpatch.tar.gz
$ mv appServer.diff $DBT1_HOME/appServer/
$ mv eu.diff $DBT1_HOME/dbdriver/
$ cd $DBT1_HOME/appServer/
$ patch < appServer.diff
$ cd $DBT1_HOME/dbdriver/
$ patch < eu.diff
```

5. DBT-1 マニュアル通りの方法で (autoconf, ./configure, make, make install) DBT-1 のインストールを行いたいところだが、4.2.2.3 の冒頭で述べた不具合の対応が必要がある。

```
$ export CFLAGS=-I/usr/local/pgsql/include
$ export LDFLAGS=-L/usr/local/pgsql/lib
$ cd $DBT1_HOME
$ autoconf
$ ./configure --with-postgresql --without-sapdb
```

ここまでで作成されているインストール関連スクリプトの修正を行う。まず、

\$ DBT1\_HOME の make.common の 4 行目付近にある

```
DBI_NAME=odbc
```

を、以下のように修正する（下線部の変更）

```
DBI_NAME=libpq
```

次に、\$ DBT1\_HOME の make.common の 12 行目付近にある

```
>CFLAGS = -g -Wall -I$(INCLUDE_DIR) -Isapdb/include -I/usr/local/pgsql/include -DHAVE_CONFIG_H  
-DGET_TIME ¥
```

を、以下のように修正する（下線部の追加）

```
CFLAGS = -g -Wall -I$(INCLUDE_DIR) -DLIBPQ -Isapdb/include -I/usr/local/p  
gsql/include -DHAVE_CONFIG_H -DGET_TIME ¥
```

最後に以下のコマンドをたたいてインストール終了である。

```
$ make  
$ make install
```

## 4.2.3 パラメータの設定

### 4.2.3.1 grub の設定

後述する測定の際に用いた環境では、12GB のメモリを搭載した IA32 PC サーバを利用した。現在、オープンソース RDBMS を搭載した DB サーバにこれだけのメモリをつむことはあまり一般的でないように思われ、これを/etc/grub.conf のブートパラメータ(mem)で 4 GB に制限した。通常はこの設定は必要ない。

### 4.2.3.2 sudo の設定

DBT-1 の実行中にシステムの root 権限が必要なプログラムを実行するので、sudo の設定を行う必要がある。sudo の設定は root ユーザで visudo を起動し、User privilege specification を次の通りに編集する。

```
# User privilege specification  
root    ALL=(ALL) ALL  
pgsql  ALL=(ALL) NOPASSWD:ALL
```

### 4.2.3.3 PostgreSQL パラメータ

ワークロード開始前にデータベースクラスタの作成が必要である。下記の手順で行う事。

```
$ initdb --no-locale --encoding =EUC_JP -D /data/DBT1
```

ここでは、日本語環境でのワークロードを行うので、上記の様に --no-locale --encoding=EUC\_JP オプションを付けた。

尚、DBT-1 実行中の PostgreSQL の稼働情報は、PostgreSQL の統計情報収集機能 (stats collector) により取得できる。この機能を有効にする場合は、エディタで \$PGDATA/postgresql.conf の関連項目を編集する。

```
stats_start_collector = true
stats_command_string = true
stats_block_level = true
stats_row_level = true
stats_reset_on_server_start = true
```

#### 4.2.3.4 OS のパラメータ

本章での測定では、OS のパラメータは以下のようにした。特に、最大ファイルディスクリプタ数(-n)は 4096 としないと、後述する同時接続ユーザ数を数千まであげた場合、エラーによって測定不可能になる場合があるため注意する。/etc/limits.conf に以下の行を追加するなどして対応する。

```
pgsql          hard   nofile          4096
pgsql          soft   nofile          4096
```

参考のため、ulimit -a の出力結果を表 4.2-3 に示す。

表 4.2-3 ulimit a の出力結果

RedHat EL AS3	SuSE ES9	Miracle Asian Linux
core file size (blocks, -c) 0	core file size (blocks, -c) 0	core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited	data seg size (kbytes, -d) unlimited	data seg size (kbytes, -d) unlimited
file size (blocks, -f) unlimited	file size (blocks, -f) unlimited	file size (blocks, -f) unlimited
max locked memory (kbytes, -l) 4	max locked memory (kbytes, -l) unlimited	max locked memory (kbytes, -l) 4
max memory size (kbytes, -m) unlimited	max memory size (kbytes, -m) unlimited	max memory size (kbytes, -m) unlimited
open files (-n) 4096	open files (-n) 4096	open files (-n) 4096
pipe size (512 bytes, -p) 8	pipe size (512 bytes, -p) 8	pipe size (512 bytes, -p) 8
stack size (kbytes, -s) 10240	stack size (kbytes, -s) unlimited	stack size (kbytes, -s) 10240
cpu time (seconds, -t) unlimited	cpu time (seconds, -t) unlimited	cpu time (seconds, -t) unlimited
max user processes (-u) 7168	max user processes (-u) 30207	max user processes (-u) 7168
virtual memory (kbytes, -v) unlimited	virtual memory (kbytes, -v) unlimited	virtual memory (kbytes, -v) unlimited

また、カーネルパラメータは以下の通りであった。

- cat /proc/sys/kernel/sem の出力結果
  - RedHat
    - 250 32000 32 128
  - SuSE
    - 250 32000 32 128
  - Miracle
    - 256 32000 100 142
- cat /proc/sys/kernel/shmmax の出力結果
  - ( 3 OS とも )
    - 2147483648

## 4.3 評価手順

### 4.3.1 環境変数の設定

DBT-1 ワークロードを実行する前に、インストール時に生成された環境設定ファイルをロードし、さらに、ライブラリパスを設定する必要がある。実行するターミナルにおいて、実行前に、pgsql ユーザで、

```
$ cd $DBT1_HOME/scripts/pgsql
$ source set_run_env.sh
$ export LD_LIBRARY_PATH=/usr/local/pgsql/lib
```

### 4.3.2 データベースの作成

#### 4.3.2.1 サイズの決定

DBT-1 では、データサイズを決定するファクターとして、ITEM(商品)数と、CUSTOMER(顧客)数がある。ここでは説明のため、それぞれ、10000、1000 とする(参考:この設定で総データは、テキスト換算で約 3GB)。測定者の興味(ユーザ数が多いが、商品が少ないアプリケーションを模したい、など)によりこれらの値を決定する。

#### 4.3.2.2 ロードデータの生成

第三章の 3.1 で述べた通りの手順で作成する。環境変数\$DBT1\_RAWDATA(たとえば、pgsql ユーザに読み書きパーミッションを与えたディレクトリ/work/medium)を設定し、ロード用テキストデータの実体の在り処、とすると、pgsql ユーザで、

```
$ cd $DBT1_HOME/datagen
$ ./datagen -d PGSQLE -i 10000 -u 1000 -p $DBT1_RAWDATA
```

3GB のデータを作成するには、環境によっては数時間かかることがある。また、実際に生成されたデータファイルには/tmp からシンボリックリンクが張られる。データロードには、データ参照パスが/tmp に固定されていることに注意されたい。

#### 4.3.2.3 データベースの生成

PostgreSQL のデータベースを作成する。pgsql ユーザで、

```
$ cd $DBT1_HOME/scripts/pgsql
$ ./build_db.sh "-c tcpip_socket=on" 0 0
```

build\_db.sh シェルのオプションの値は、DBT-1 PostgreSQL マニュアルに従っている。これにより、

- DBT1 データベースの削除 (DBT1 データベースがすでに存在した場合のみ、実体は./drop\_db.sh)
- データベース・ユーザなどの作成(./create\_db.sh)

- テーブルの作成(/create\_tables.sh)
- データのロード(/load\_db.sh)
- インデックスの作成(/create\_indexes.sh)
- 外部キーの作成(/create\_fk.sh)
- ストアードプロシージャの作成(/load\_dbproc.sh)
- 統計情報の作成(/update\_statistics.sh)

が実行される。尚、load\_db.sh は、複数テーブルのデータをシーケンシャルにロードしていく。パラレルにロードすることも考えられるが、PostgreSQL version 8 で実現したテーブルスペース機能を使わない限り、トランザクション性能の劣化が予想されるパラレルロードは避けたほうがよい。

#### 4.3.2.4 DBT-1 パラメータの設定

ここではあらかじめDBT-1に準備される起動スクリプトを使ってDBT-1を動かすことを想定する（すると、システム稼働統計などを裏で取得してくれる）。この場合、\$DBT1\_HOME/data\_collect/pgsql/run.config を編集する。主な項目の意味は以下の通りである。

- items: 商品数。データ生成時のパラメータ値と同様でなければならない。
- dcustomers: 顧客数。2880\*<データ生成時の顧客数>を設定する
- dbhost: データベースサーバホスト名
- Cache: 0 – もしキャッシュサーバを使わない場合。1 – キャッシュサーバを使う場合。
- xcache\_host: キャッシュサーバホスト名
- lcache\_port: キャッシュサーバの Port 番号
- mconnection: キャッシュサーバがデータをキャッシュするときのデータベースサーバとの接続数。
- server\_host0: 第一アプリケーションサーバのホスト名
- nserver\_port0: dbdriver が接続するポート番号。
- q\_size0: 第一アプリケーションサーバのトランザクションキューのサイズ
- z\_size0: 第一アプリケーションサーバのトランザクション配列のサイズ
- rconnection0: 第一アプリケーションサーバとデータベースサーバの接続数
- ydriver\_host0: 第一 dbdriver のホスト名
- vrate0: 第一 dbdriver のアプリケーションサーバとの初期接続のレート。接続数/秒。
- eus0: 第一 dbdriver がエミュレートするユーザ数。
- zduration0: 第一 dbdriver の実行時間。
- think\_time0: 第一 dbdriver の各ユーザの平均 Think time。各ユーザの Web リクエストの間隔（秒）を表す
- jdriver\_server\_host0: 第一 dbdriver が接続しに行くアプリケーションサーバホスト名
- kdriver\_server\_port0: 第一 dbdriver が接続しに行くアプリケーションサーバのポ

ート番号

- out\_dir: 出力ディレクトリ名。

DBT-1 は第 8 章にあるように、Web 2 層モデル、Web 3 層モデルのどちらでも動作させることが可能であるが、現在世間でよく採用されている Web 3 層モデルを測定対象としている。すなわち、dbdriver ( Web クライアントエミュレータ ) <->AppServer ( コネクションプール付 ) <->DBMS の構成である。この構成における、今回の測定の標準設定は ( チューニングなどによりいくつかの項目を変更を適宜しているが ) 表 4.3-1 の通りである。

表 4.3-1 DBT-1 の測定パラメータ

#	項目名	値
1	# database config	
2	items	10000
3	gcustomers	2,880,000
4	dbhost	localhost
5	bdbname	DBT1
6	username	pgsql
7	password	pgsql
8	# cache config	
9	cache	0
10	xcache_host	localhost
11	lcache_port	9999
12	mconnection	10
13	# appServer config	
14	appserver	1
15	server_host0	localhost
16	nserver_port0	9992
17	q_size0	1000
18	a_size0	1000
19	rconnection0	80
20	# dbdriver config	
21	ydriver_host0	localhost
22	vrates0	100
23	eus0	100
24	zduration0	1200
25	think_time0	7.2
26	jdriver_server_host0	localhost
27	kdriver_server_port0	9992
28	out_dir	/tmp
29	db_param	-c tcpip_socket=on
30	redirect_tmp	1
31	redirect_xlog	1

トランザクション作業領域の容量が不足すると、DB とは関係の無い箇所で処理待ちが生じるため、Think\_time の値によるが、できれば、上記のパラメータのうち、q\_size0 と z\_size0 は eu の設定値以上の値を指定することが望ましい。

尚、同時接続ユーザ数や、appServer の DB コネクション数などの DBT-1 の実行時設定は、dbdriver や appServer を直接起動する場合は、それぞれの起動オプションで与えることができる。

### 4.3.3 DBT-1 の実行

#### 4.3.3.1 データベースサーバの起動

最初に PostgreSQL を起動する。起動している場合は必要ないが、PostgreSQL 内部のキャッシュなどをクリアするには、再起動することが必要である。pgsql ユーザで、

```
$ pg_ctl stop
$ pg_ctl start
```

#### 4.3.3.2 DBT-1 アプリサーバの起動

pgsql ユーザで、4.3.1 で述べた環境設定を行ったあと、

```
$ cd $DBT1_HOME/data_collect
$ ./dbt1_slave.pl
```

を実行する。

#### 4.3.3.3 DBT-1 クライアントエミュレータの起動

DBT-1 のワークベンチを起動する。pgsql ユーザで、4.3.1 で述べた環境設定を行ったあと、

```
$ cd $DBT1_HOME/data_collect/pgsql
$ ./dbt1_master.pl -f run.config
```

を実行する。尚、-f オプションでコンフィギュレーションファイルを指定できる。複数のケースについてベンチマークをする場合、コンフィギュレーションを別ファイルに保存し、適宜-f オプションで指定するとよい。

尚、以下に述べる測定結果は、4.2.3.3 で述べた PostgreSQL の統計情報収集のためのパラメータは、False にして測定したものであることをお断りしておく。各測定前に、ファイルシステムのアンマウント・再マウントを行い、ファイルシステムのキャッシュはクリアしている。また、測定用データは同一のものをを用いている。

## 4.4 実行及び考察

### 4.4.1 実行結果の収集

実行結果は、run.config ファイルの out\_dir に記述した出力ディレクトリに出力される。主に参照されるべきは、BT というファイルの中に現れる、トランザクション種別毎の比率、平均レスポンス (秒) ならびに、BT/秒 (単位: ボゴ (擬似) トランザクション/sec = web 経由のリクエスト数/sec) である。この他にいくつかの情報が取得できる。代表的なものを表 4.4-1 に示す。

表 4.4-1 実行結果を記録するファイル

ファイル名	概要	備考
BT	トランザクション種別毎の比率・平均応答時間、ボゴトランザクション	要はスループット。
config.txt	実行時のOSパラメータやCPU、メモリサイズなどのシステム環境情報、DBT-1の設定の記録。	
param.out	PostgreSQLの設定パラメーター一覧	
run.meminfo0.out	DBT-1起動時のメモリ関連情報	
run.meminfo1.out	DBT-1終了後のメモリ関連情報	
indexes.out	DBT-1起動時点でのPostgreSQLのユーザインデックスの利用統	
run.iostat.out	システムのIO関連の情報	iostat d 60と等価。man.iostatを参照のこと。
run.vmstat.out	システムのメモリ・IO・CPUに関連する統計。	vmstat 60と等価。Man vmstatを参照のこと。
result.mix.log	トランザクション毎のトランザクション種別、要求時刻、返答時刻。	これを編集したものがBT。
ips.csv	30秒区間ごとの平均トランザクション数の推移。	入力はresult.mix.log
db_stat/db_activity[n].out	開始後n分後時点でのPostgreSQL接続セッションに関する情報。以下、postgresql.confにて各統計情報出力オプションをTRUEにしたと	select * from pg_stat_activity;
db_stat/db_load[n].out	開始後n分後時点でのPostgreSQLからみた、内部Cache・ファイルシステムI/O等の統計。	select * from pg_stat_database where datname='DBT3';
db_stat/index_info[n].out	開始後n分後時点でのPostgreSQLのユーザインデックスに関する内部Cache・ファイルシステムI/O等の統計。	select relid, indexrelid, relname, indexrelname, idx_blks_read, idx_blks_hit from pg_statio user indexes;
db_stat/indexes_scan[n].out	開始後n分後時点でのPostgreSQLのユーザインデックスの問い合わせ数等の統計。	select * from pg_stat_user_indexes;
db_stat/lockstats[n].out	開始後n分後時点でのPostgreSQLのロックに関する情報。	select relname,pid, mode, granted from pg_locks, pg_class where relfilenode = relation;
db_stat/table_info[n].out	開始後n分後時点でのPostgreSQLのユーザテーブルに関する内部Cache・ファイルシステムI/O等の統計。	select relid, relname, heap_blks_read, heap_blks_hit, idx_blks_read, idx_blks_hit from pg_statio user tables;
db_stat/table_scan[n].out	開始後n分後時点でのPostgreSQLのユーザテーブルの問い合わせ方法の統計。	select * from pg_stat_user_tables;
db_stat/tran_lock[n].out	開始後n分後時点でのPostgreSQLのロックに関する情報。	select * from pg_locks where transaction is not NULL;

尚、DBT-1にも、DBT-3において提供されている測定結果をグラフ化・HTMLに変換する機能が提供されているように見受けられるが、今回は時間の関係で動作を確認しなかった。DBT-1 v2.1のPostgreSQL版向けマニュアルには、「もともとOSDLの自動テストスイート環境向けに開発されたものであり、(中略)であると、結果はmessyになる。」と記述されていたためである。

#### 4.4.2 測定の観点

今回は、以下のような観点で測定・実行を行った。

- 同時接続数(システムに同時に接続するWebエンドユーザ数)によるスループットと平均応答時間の特性
  - 前述の通り、今回の測定ではWeb 3階層を採用している。AppServerとデータベースサーバの間のコネクションを一定にしつつ、Webクライアントのエミュレートユーザ数を変化させた場合の特性をみる。
- データベースの規模による特性の変化
  - item数10000かつcustomer数1000(ロード用テキストファイルにして約3GB)と、サイズの約10倍であるitem数100000かつcustomer数10000(ロード用テキストファイルにして約35GB)にて測定を行い、サイズの違いによる特性の変化を調査する。
- 簡単なチューニングによる性能の変化
  - データベースのチューニングは、そのパラメータが多く、非常に奥が深いものであるが、限られたHWリソース+時間リソースの中で、トランザクション性能を

げる一般的なチューニングを行う。チューニングの効果を明確にするために、チューニング前の測定結果とチューニング後の測定結果を比較する。「チューニング前」とは、PostgreSQL の設定 (Buffer Cache や Max Connection 数) を PostgreSQL のデフォルト設定 (比較的古いロースペックの HW 上でも動作する設定) + DBT-1 のデフォルト実行手順とする環境である。

尚、本章で述べるチューニング結果は、実務上ほとんどのチューニングがそうであるように、完璧ではないことをあらかじめお断りしておく。

- OS ディストリビューションによる性能特性の違い

今回測定対象とする OS ディストリビューションは、RedHat、Miracle、SuSE の 3 つとした (詳細は 4.2.2.1 を参照のこと)。SuSE は Kernel は 2.6 ベース、そのほかの二つは Kernel 2.4 ベースであり、Kernel 2.6 での改善によって性能に開きが出る可能性があるが、前者 2 つのディストリビューションも、Kernel 2.6 の改善内容の多くをバックポーティングしたものであるため、性能差がほとんどないことが予想される。これを確認する。

#### 4.4.3 測定結果と考察

以下、前項で述べた「同時接続数 (システムに同時に接続する Web エンドユーザ数) によるスループットと平均応答時間の特性」以外について、測定の結果と考察をまとめる。それぞれの項において、同時接続数によるスループットと平均応答時間の特性の情報を掲載しているので、特に、「同時接続数 (システムに同時に接続する Web エンドユーザ数) によるスループットと平均応答時間の特性」についての独立の項を設けない。

##### 4.4.3.1 データベースサイズの違いによる性能特性

###### 4.4.3.1.1 目的

最初に、データベースのサイズによる性能特性の違いをみることを目的とした測定を行った。

###### 4.4.3.1.2 測定結果

item 数 10000 かつ customer 数 1000 (ロード用テキストファイルにして約 3GB) のデータベースと、サイズの約 10 倍である item 数 100000 かつ customer 数 10000 (ロード用テキストファイルにして約 35GB) のデータベースである。ここでは、「チューニング前」の設定、すなわち PostgreSQL の設定 (Buffer Cache や Max Connection 数) を PostgreSQL のデフォルト設定+DBT-1 のデフォルト実行手順とする環境で測定した。

1 秒あたりの BT 値 (ボゴトランザクション/秒: スループット) と、平均応答時間を以下に示す。横軸には、エンドユーザ同時接続数 (eu: emulated user) をとっている。OS ディストリビューションは、RedHat である。DBT-1 の設定パラメータの run.config のうちこれらの測定で異なる点は、接続ユーザ数 (グラフ中の eu と同じ) ならびにデータベースのサイズによって gcustomer を 2880000 あるいは 28800000 としている点である。

データサイズ小の結果を図 4.4-1 に示す。

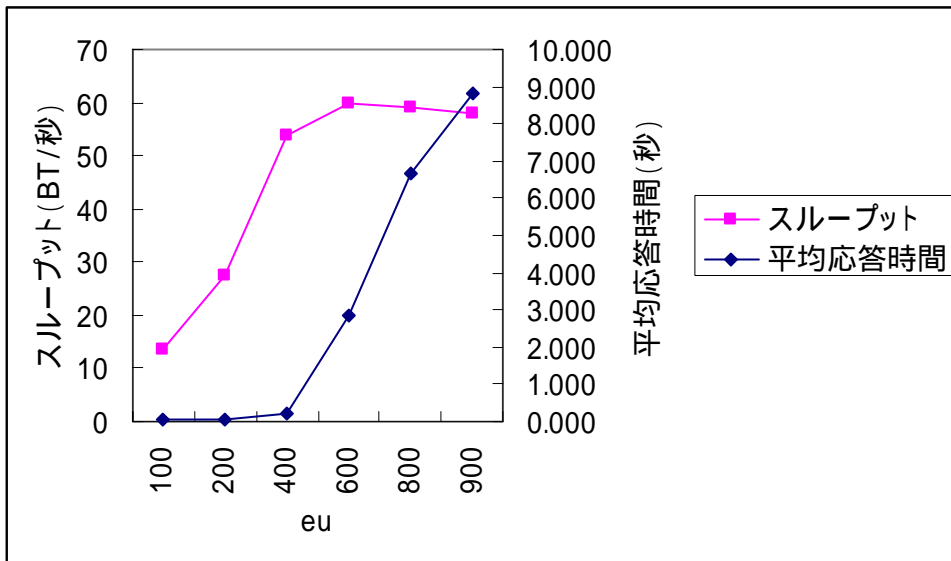


図 4.4-1 eu による性能推移 RedHat データベースサイズ小 チューニング前

eu=600 付近で BT/秒が約 60 程度で頭打ちになり、平均応答時間が急激に悪化しはじめる。eu=600 において 2.84 秒であった。トランザクション種別によっては平均 6 秒以上かかっているものもあり、最近のユーザにとっては決して快適な応答時間ではない。

次に、同一設定環境で、データサイズを約 10 倍にしたデータベースでの測定結果を図 4.4-2 に示す。

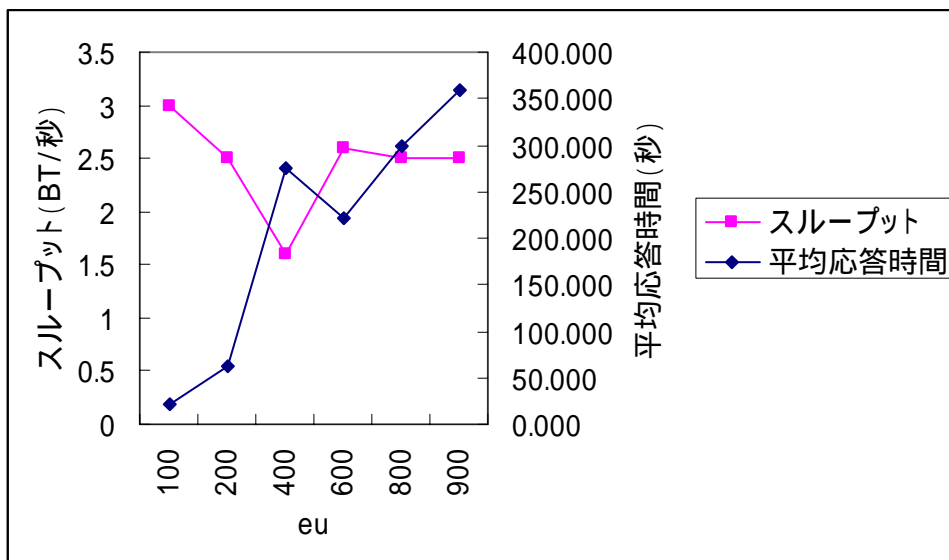


図 4.4-2 eu による性能推移 RedHat データベースサイズ大 チューニング前

データベースの負荷的には非常に高い測定となり、スループットは3トランザクション/秒以下まで落ち込んだ。平均応答時間も eu=100 でさえ、21.9 秒であり（最も重いトランザクション種別では平均 80 秒） 実用に耐えられる状況ではない。今回の測定範囲では eu の違いによるスループットの傾向に意味のある解釈はできなかった。

#### 4.4.3.1.3 考察

デフォルト設定においても、データサイズがテキスト換算で 3GB 程度であれば、エンドユーザ同時接続数 400 程度において、PostgreSQL は平均応答時間も 1 秒以下であり、実用上問題なく動作することが判明した。参考のため、データサイズ小・eu=600 の vmstat 情報を以下に示す。

横軸には経過時間を取っている。凡例は、vmstat の表記に従っているが、念のため簡単なコメントを加えると、bi は、block read /sec、bo は、block write /sec、us は、user CPU 利用率、sy は、system CPU 利用率、id は CPU アイドル率、wa は CPU の IOWAIT 率である。CPU の利用率は 70%程度、wa は、3%程度で安定して推移する。

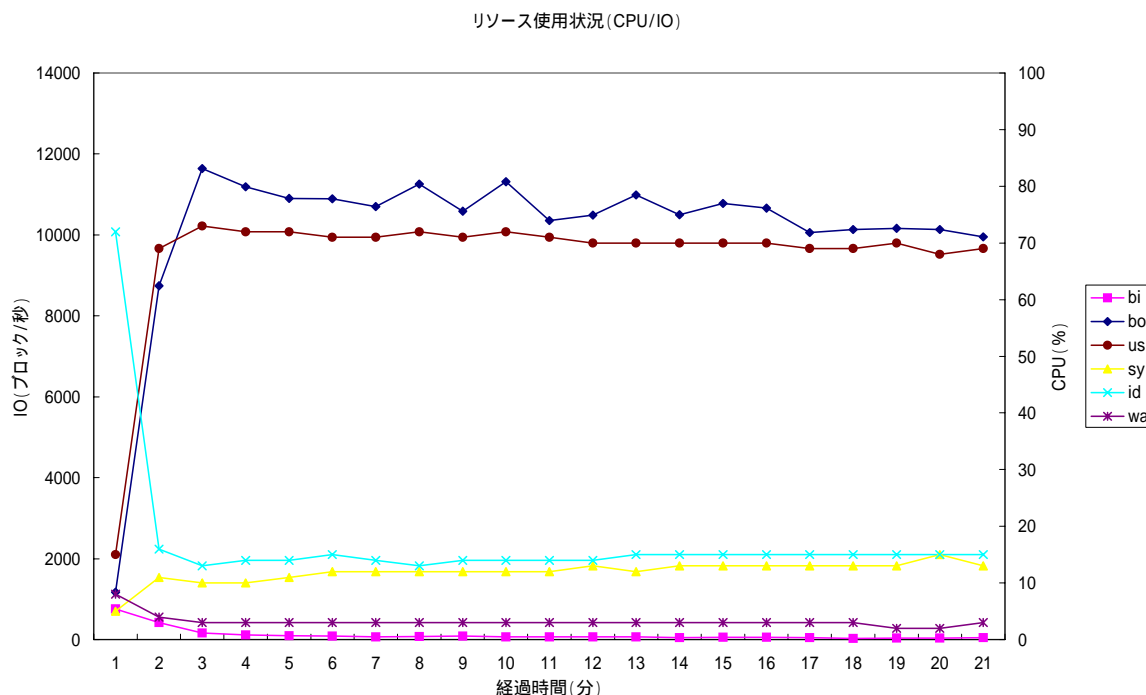


図 4.4-3 eu=600 における vmstat 情報 RedHat データベースサイズ小 チューニング前

次に、データサイズ大でのシステムリソース情報をみてみると、wa が時に 90%を超えていて、ディスクネックでシステムが高負荷になり、不安定な状態であることがわかる。以下にデータサイズ大・eu=100 の時の例を図 4.4-5 に示す。

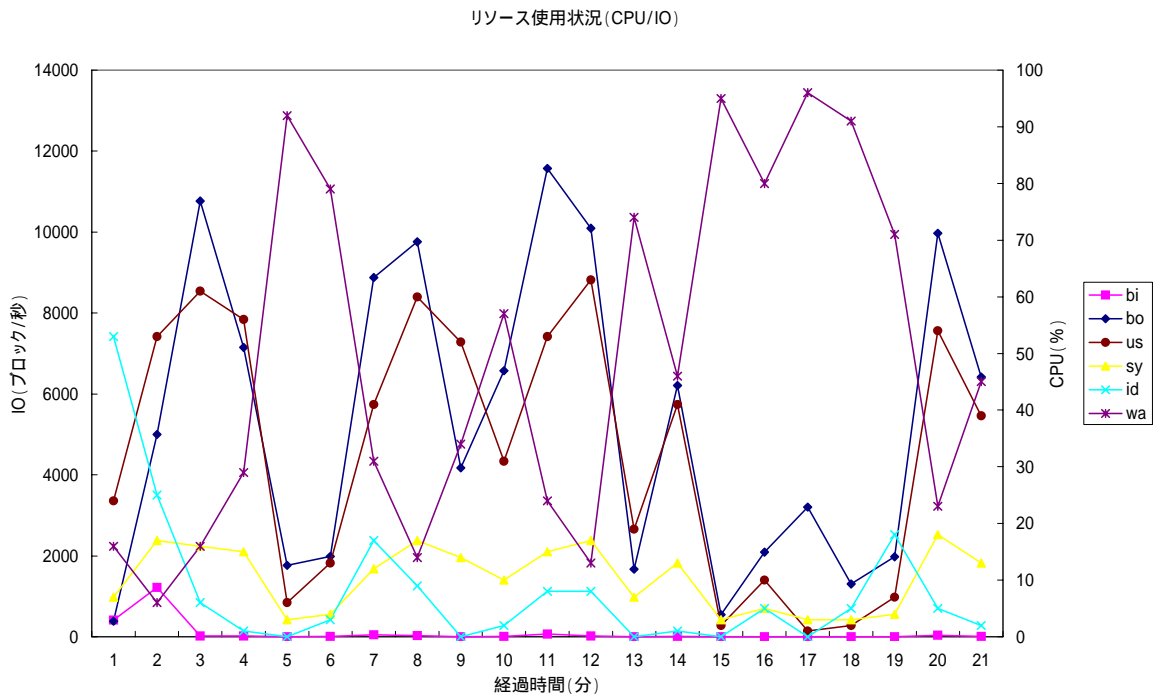


図 4.4-4 eu=600 における vmstat 情報 RedHat データベースサイズ大 チューニング前

同一のアプリケーションであるにもかかわらず、データサイズを増やした結果、これほどまでに DB サーバの挙動が不安定になったという事実が示すのは、DB サーバのチューニングの必要性である。現実世界では、多くの場合、そもそもデータサイズが小さかったときに起きていた問題が、顕在化した、とまずは理解するのが妥当である。そこで、次項では、データサイズ小のデータベースに対し、問題点の分析を行い、ある程度のチューニングを施し、データサイズ小とデータサイズ大で測定した結果を示す。

#### 4.4.3.2 簡単なチューニングによる性能の変化

##### 4.4.3.2.1 目的

データサイズ小のデータベースにおいて、前項で測定したデフォルト設定の問題を分析し、簡単なチューニングを施した場合の性能特性の変化を調べる。

##### 4.4.3.2.2 問題の分析とチューニング

前項で測定したデフォルト設定・データサイズ小のデータベースの問題を簡単に分析してみる。ここではまず、ややアプリケーション寄りのアプローチをとる。

ミクロな観点からの入り口として、重たいトランザクション種別をピックアップしてみる。前項で測定した BT ファイルを参照すると、トランザクション種別毎の平均応答時間がわかる。以下は一例である。すべてのトランザクションについてチューニングしたいところだが、時間の制約から、ここでは Buy Request と、Best Sellers に注目することにする。

以下は、4.4.3.1 で測定した、データベースサイズ小、eu が 600 のときの値の例である。

Interaction	%% Avg.	Response Time (s)
Admin Confirm	0.08	2.029
Admin Request	0.1	1.948
Best Sellers	5.1	6.196
Buy Confirm	1.13	3.052
Buy Request	2.48	3.637
Customer Registration	2.91	0
Home	16.79	1.95
New Products	5.02	6.194
Order Display	0.69	2.125
Order Inquiry	0.76	1.811
Product Detail	16.73	1.898
Search Request	19.92	0
Search Results	16.82	5.816
Shopping Cart	11.48	3.124

次に、マクロな観点からの情報として、データベース内のテーブルのアクセス状況を参照する。これには、4.2.3.3 でのべた、PostgreSQL 統計情報取得のための postgresql.conf のパラメータを TRUE にして、統計情報リセット ( postgresql.conf 内の stats\_reset\_on\_server\_start パラメータを TRUE にしてあれば、PostgreSQL の再起動でリセットされる ) 後、DBT-1 を動作させてみる。動作終了後 DBT-1 結果ディレクトリの db\_stat/table\_scan[n].out を参照する。Item テーブルの seq\_scan (シーケンシャルスキャン) の回数が非常に多いのと、shopping\_cart テーブルならびに shopping\_cart\_line テーブルのインデックスを使ったアクセス ( idx\_scan ) が 0 件 (全てシーケンシャルスキャンであるということ) であるということが気になる。これを念頭に、ミクロな分析を進める。

先に注目した Buy Request と Best Sellers の処理中で発行される SQL 文を DBT-1 ソースの中 (\$DBT1\_HOME/storedproc/pgsql の下にあるストアードプロシージャソース) から拾って、SQL の実行計画の分析を試みる。実行計画の分析の方法は、本 DB 層報告書添付資料トランザクション特性の解析に詳しく解説してあるので参照されたい。実行計画の分析の結果、以下の事実が判明した。

- item テーブルの i\_subject 項目 ( VARCHAR 形式 ) にインデックスが張っており、かつ、i\_subject のカーディナリティもそれなりであるにもかかわらず、WHERE i\_subject = 'MYSTERY' とした SELECT 文で、item テーブルのシーケンシャルスキャンが起きている。インデックススキャンを強制的に行うよう指定すると、問い合わせ性能は数倍に向上する。
- すべての shopping\_cart テーブルならびに shopping\_cart\_line へのアクセスは Primary Key インデックスを参照せずに、シーケンシャルスキャンになっている。原因は、DBT-1 の実行手順にあった。PostgreSQL のクエリオプティマイザが実

行計画（たとえばインデックスを使うかシーケンシャルスキャンをするか）を決定する際に参照するテーブルの統計情報作成(analyze table 文)は、データのロード直後に行われるが、これら二つのテーブルは、その時点では0件である。これが原因となって、オプティマイザはこの二つのテーブルに対しては必ずシーケンシャルスキャンとする(PostgreSQLの仕様と思われる)ようになってしまった。これでは、この二つのテーブルの件数が増大するにしたがって、イクスポネンシャルに性能が劣化してしまう。

これらの事実への対処は以下の通りである。前者については、postgresql.conf の random\_page\_cost パラメータを変更し、オプティマイザがインデックスを使用する傾向を強める。デフォルトが4であるが、これを2とすることでクエリプランは、インデックスを使用するようになった。

後者については、これら二つのテーブルにある程度データがたまった段階で統計情報を再作成するより他に手はない。本測定では、DBT-1 実行前にこれらのテーブルに数百行のデータを作成し、analyze table shopping\_cart;と、analyze table shopping\_cart\_line;を実行してから、DBT-1 を実行するよう手順を変更した。この手順により、これら二つのテーブルへのアクセス(参照・更新)のうち、Primary Key インデックスを参照すべきものは参照されるようになった。念のため、この状態でDBT-1 を実行、db\_stat/table\_scan[n].out を参照して、item テーブルのシーケンシャルスキャンが激減したこと、shopping\_cart テーブルならびに shopping\_cart\_line へのアクセスはインデックススキャンに切り替わったことを確認した。

次に、デフォルトの postgresql.conf は、比較的ロースペックの HW 環境でも動作するよう設定されているため、ソートエリアサイズやバッファキャッシュサイズを増やす変更をした。複数に渡る試行錯誤をしたが、その範囲では目立った性能の向上はみられなかった。

この他に、AppServer と PostgreSQL の間のコネクション数(アプリケーションや HW 環境などにより異なるが、最適値が存在する-多すぎても少なすぎても良い性能が引き出せない)や、デッドロックの検知サイクル(短すぎると DB に負担がかかる)などがチューニングポイントと言われている。これらも若干変更した。チューニング前の値とチューニング後の値を表 4.4-2 以下に示す。

表 4.4-2 PostgreSQL の設定内容

postgresql.confのパラメータ名	チューニング前	チューニング後
max_connections	100	100
shared_buffers (8KB each)	1,000	131,072
sort_mem (KB)	1,024	15,146
random_page_cost	4	2
deadlock_timeout (millisec)	1,000	100,000

#### 4.4.3.2.3 測定結果

item 数 10000 かつ customer 数 1000 (ロード用テキストファイルにして約 3GB) のデ

データベースに対し、前項で述べたチューニングを施した状態で DBT-1 の測定をした。

1 秒あたりの BT 値（ポゴトランザクション/秒：スループット）と、平均応答時間を図 4.4-5 に示す。OS ディストリビューションは、RedHat である。DBT-1 のパラメータ run.config は、4.4.3.1 と同様である。

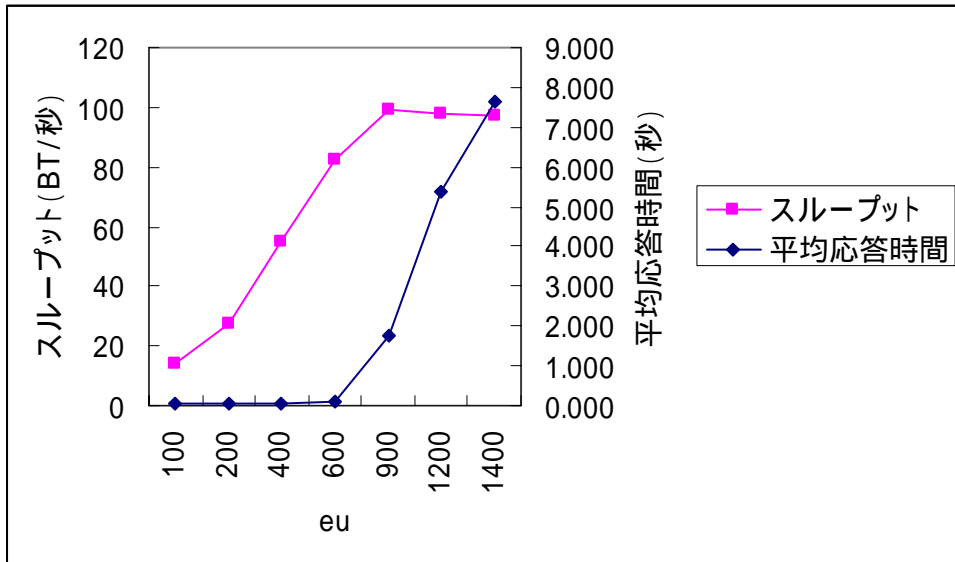


図 4.4-5 eu による性能推移 RedHat データベースサイズ小 チューニング後

チューニング前は、eu=600 付近で BT/秒が約 60 程度で頭打ちになり、平均応答時間は 2.84 秒であったが、チューニング後は大幅に改善され、eu=900 において、BT/秒が 99、平均応答時間が 1.78 秒となった。

参考のために、各トランザクションのレスポンスがチューニングによってどれほど改善されたかを示すために、RedHat、データベースサイズ小、eu=600 の例を以下に示す（括弧内は同一条件・チューニング前の値の例）。

Interaction	%% Avg.	Response Time (s)
Admin Confirm	0.09	0.062 (2.029)
Admin Request	0.1	0.046 (1.948)
Best Sellers	5.09	0.185 (6.196)
Buy Confirm	1.16	0.057 (3.052)
Buy Request	2.56	0.075 (3.637)
Customer Registration	2.99	0 (0)
Home	16.64	0.035 (1.95)
New Products	4.93	0.179 (6.194)
Order Display	0.65	0.092 (2.125)
Order Inquiry	0.74	0.043 (1.811)
Product Detail	16.79	0.016 (1.898)

Search Request	19.9	0	(0)
Search Results	16.88	0.311	(5.816)
Shopping Cart	11.48	0.057	(3.124)

次に、同一環境で、サイズ的には上記測定データベースの約 10 倍である item 数 100000 かつ customer 数 10000 で測定した。結果を図 4.4-6 に示す。

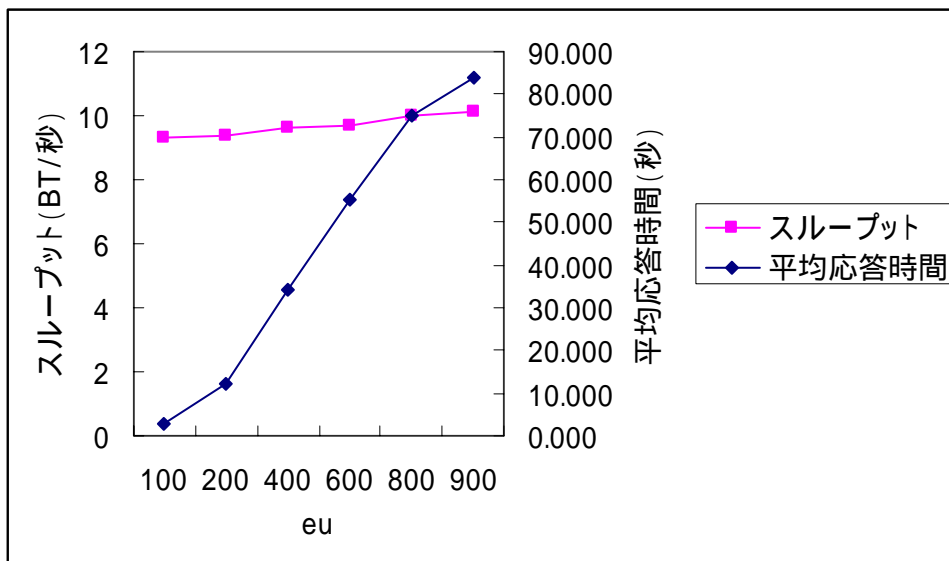


図 4.4-6 eu による性能推移 RedHat データベースサイズ大 チューニング後

チューニング前は、eu=100 において、3BT/秒、平均応答時間は 21.9 秒であったが、チューニング後は、eu=100 において 8.7BT/秒、平均応答時間は 3.34 秒と大幅に改善された。

#### 4.4.3.2.4 考察

4.4.3.1 において、何の工夫もしないデフォルト設定においても、データサイズがテキスト換算で 3GB 程度であれば、本測定では、エンドユーザ同時接続数 400 程度において、PostgreSQL は平均応答時間も 1 秒以内で動作していたが、SQL 文の実行計画をみると、システムを稼働させ続けると性能が著しく劣化していくデータベースであったことが予想された。

実際に、Buy Request というトランザクションの応答時間の推移を観察すると、時間がたつにつれ（トランザクションテーブルの件数が増えていくにつれ）応答時間が増大していくことが確認できる。以下にデータサイズ小のチューニング前/チューニング後の性能劣化の様子を図 4.4-7 に示す。

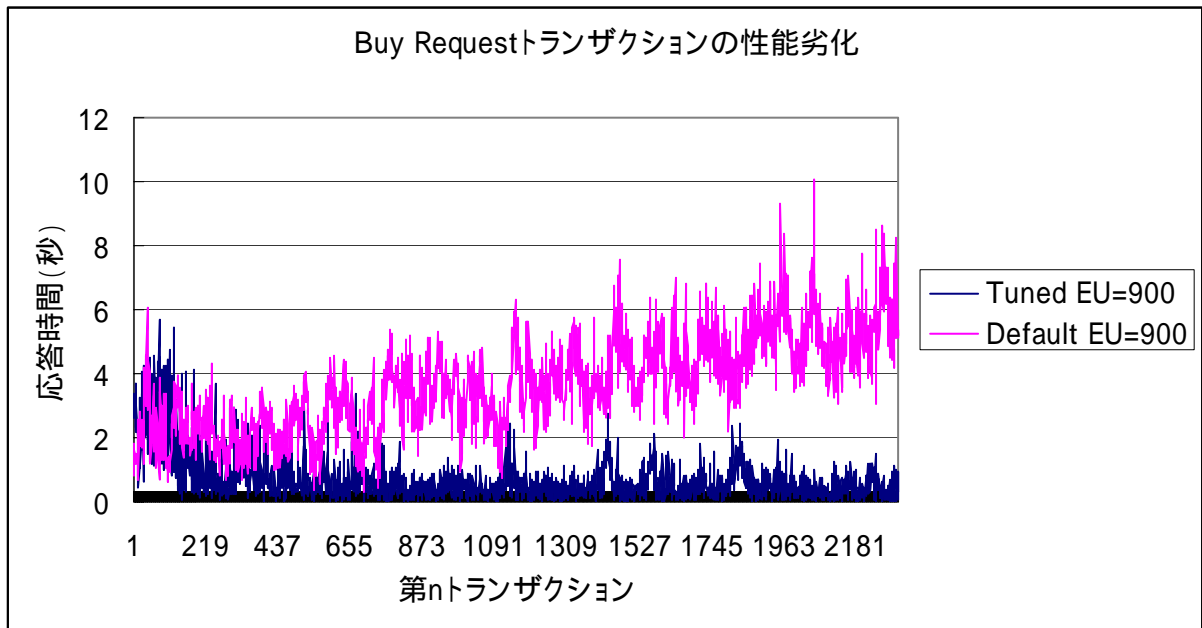


図 4.4-7 Buy Request トランザクションの経時性能変化

適切な処置（シーケンシャルスキャンではなくインデックスを利用すべきところは、インデックスを利用するアクセスに切り替える）をすることにより測定時間中、性能が一定に保たれることが確認できる。

DBT-1 をただ実行しただけでは、PostgreSQL においては、オプティマイザが必要とする適切な統計情報が取得できないという、実行手順上の問題がオリジナルの DBT-1 にはある。細かく言うと、トランザクションテーブルである、shopping\_cart テーブルと shopping\_cart\_line テーブルの Primary Key インデックスが利用されることはない。理由は、DBT-1 をただ実行する手順においては、これらのテーブルの統計情報が適切でないためである。PostgreSQL ユーザの中で強く推奨されている vacuum 動作（削除レコードの掃除と、統計情報の再作成）をしない（これは PostgreSQL にとって現実的な運用ではない）劣悪なデータベースをベンチマークしていることになってしまう。

そこで、DBT-1 による PostgreSQL の性能測定手順に関し、以下の提案をする。要は、autovacuum（vacuum の自動化）使用を標準測定手順にするということである。手順として、dbdriver（あるいはその起動スクリプトである、dbt1\_slave.pl）を起動すると同時に、

```
$PGBINS/pg_autovacuum -s 10 -S 2 ... [other arguments]
```

を実行する。ここで、other argument は、

- v 1000
- V 2
- a 100
- A 4

を推奨する。今回問題になっている shopping\_cart\*テーブルは、データロード後初期状態で 0 件、意味のある統計情報をなるべく早い段階で作成したいため、-a 100 とした。そのかわり、その後のスケールファクタを 4 と少し大きめの値にしている。

Item テーブルに関しては、チューニングを施しても、シーケンシャルスキャンが激減するにせよある程度残るのは前述したとおりである。実は、これは、DBT-1 の Search Results トランザクションの中に、item テーブルに対する i\_title LIKE "%||\_i\_title||%" (文字列項目の途中一致) 検索があるためである。DBT-1 では、item テーブルは、実行中も件数が一定に保たれるが、item テーブルの件数が多いケースでは、シーケンシャルスキャンの存在が一因となる性能の劣化が顕著になることが予想される。言い換えると、Disk I/O が多発する様相である。次項では、これに関連し、Kernel のバージョンの違いによる性能特性を比較する。

#### 4.4.3.3 ディストリビューションの違いによる性能特性

##### 4.4.3.3.1 目的

Kernel 2.6 ベースの OS ディストリビューションと、Kernel 2.4 ベースの OS ディストリビューションによる PostgreSQL の性能特性の違いを把握する。

##### 4.4.3.3.2 測定結果

ディストリビューションは、RedHat、Miracle(以上 Kernel 2.4 ベース)、SuSE(Kernel 2.6 ベース)の3つとして、性能特性を比較した。

まず、データサイズ小・チューニング後のデータベースでの測定結果を RedHat と SuSE ならびに Miracle の順に示す。

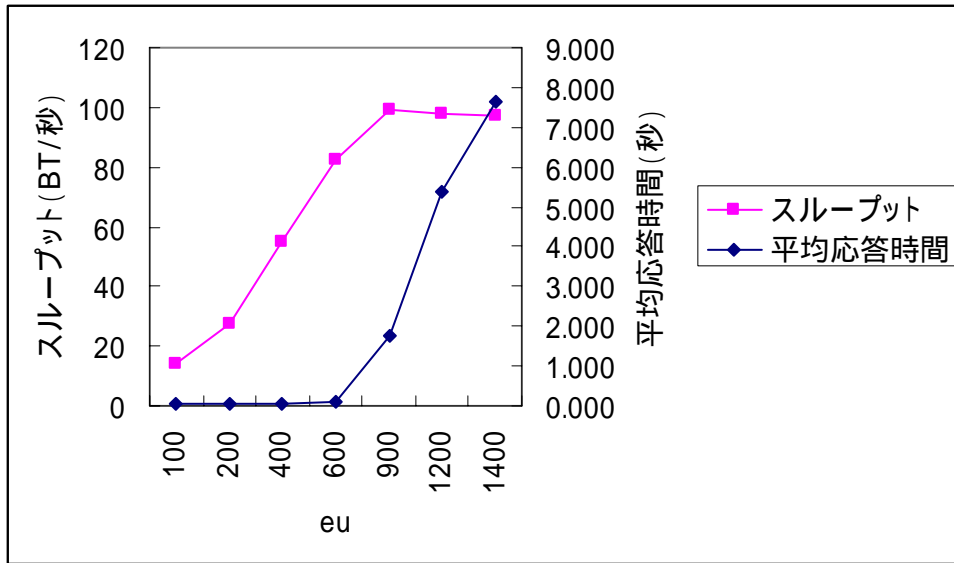


図 4.4-8 eu による性能推移 RedHat データベースサイズ小 チューニング後

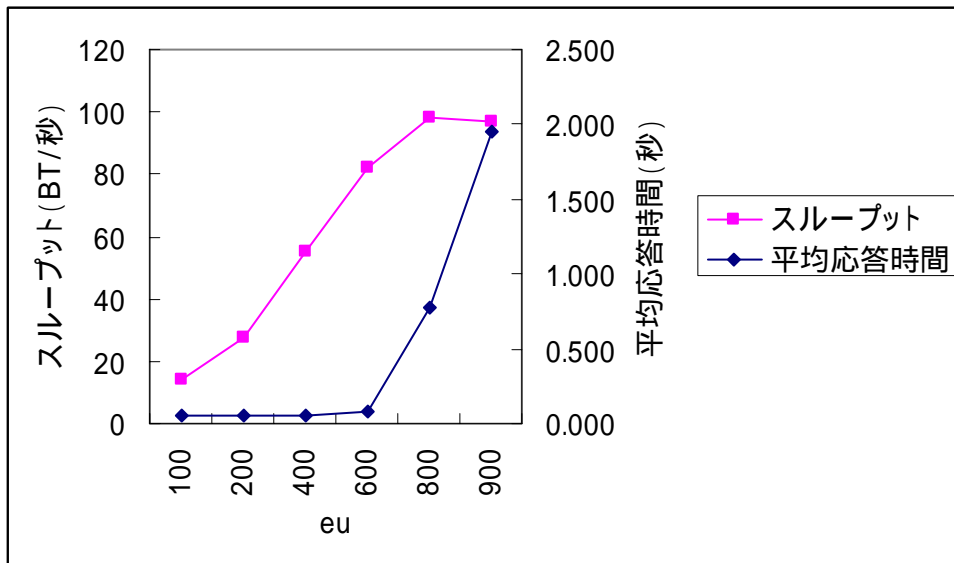


図 4.4-9 eu による性能推移 SuSE データベースサイズ小 チューニング後

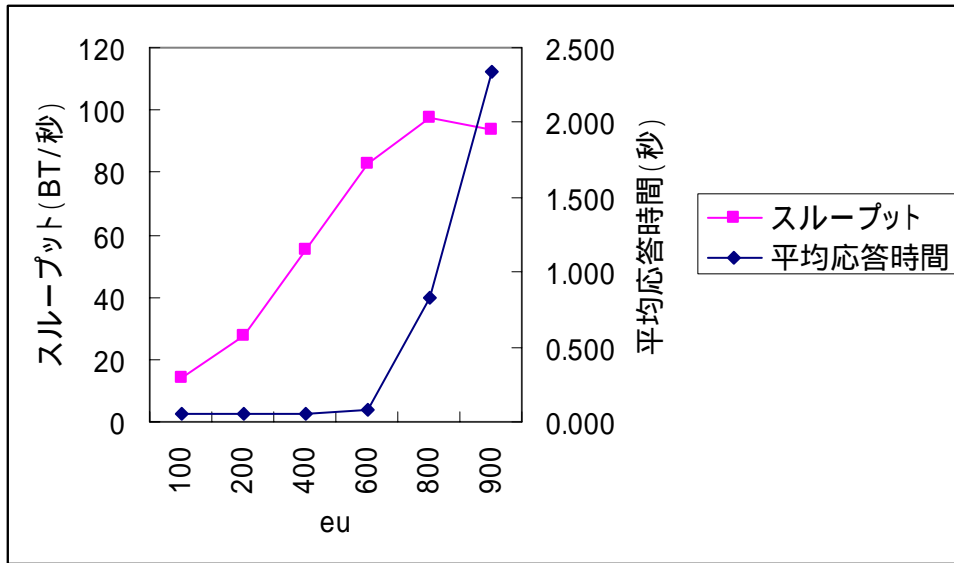


図 4.4-10 eu による性能推移 Miracle データベースサイズ小 チューニング後

RedHat と SuSE が最大 BT/秒を記録した、eu=800 で比較すると RedHat は 100.7 に対し SuSE は 98.5BT/秒、Miracle は 97.8BT/秒、平均応答時間は RedHat の 0.60 秒に対し、SuSE0.77 秒、Miracle0.828 秒である。顕著な差は現れていないが、性能の順にいうと、RedHat>SuSE>Miracle の順となる。

次に、データベースにとってはより負荷の高い(この場合 I/O 負荷が高い)データサイズ大のチューニング後のデータベースでの比較測定結果を以下に示す。

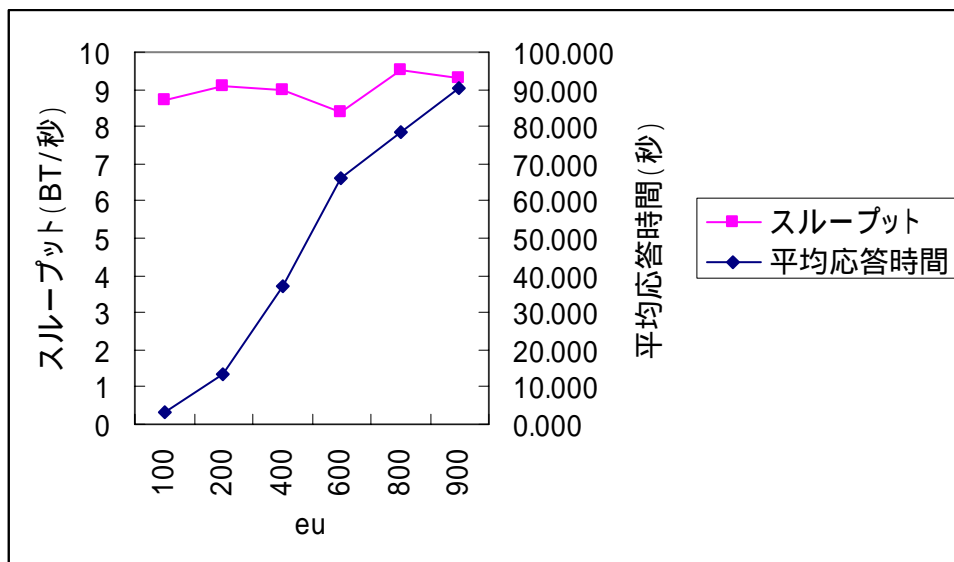


図 4.4-11 eu による性能推移 RedHat データベースサイズ大 チューニング後

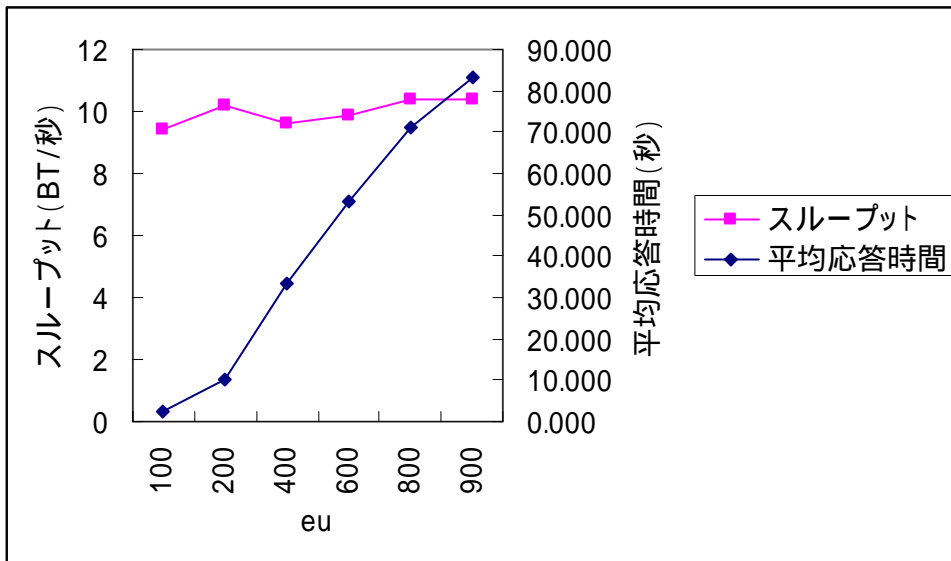


図 4.4-12 eu による性能推移 SuSE データベースサイズ大 チューニング後

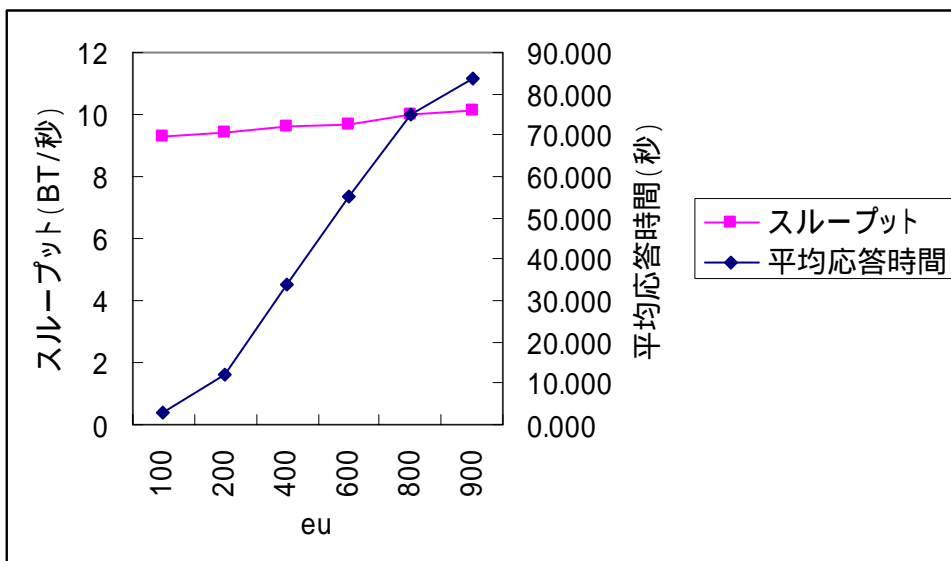


図 4.4-13 eu による性能推移 Miracle データベースサイズ大 チューニング後

性能順にいうと、データサイズ小とは状況が一転し、SuSE>Miracle>RedHat の順となった。SuSE と Miracle はかなり互角である。SuSE と RedHat を比較すると、スループット・平均応答時間の両方について 1 割強 SuSE のほうが成績がよいことが判明した。

尚、ここでは掲載しないが、チューニング前のディストリビューション毎のデータサイズ小・データサイズ大の比較結果も、データサイズ小・チューニング後のデータサイズ小・

データサイズ大のディストリビューション性能順位の傾向とほぼ同様であった。

#### 4.4.3.3 考察

データサイズ小のデータベースにおいては、ディストリビューション間の性能差に大きな違いはみられなかった一方で、データサイズ大のデータベースにおいては、Kernel 2.6ベースの SuSE が Kernel 2.4 ベースの OS よりも 1 割強高性能であった。これは驚きである。Kernel 2.6 の改善部分は、RedHat や Miracle へバックポーティングされていると言われていたため、実は、筆者は今回の 3 つのディストリビューション間に性能差はないと予想していた。

筆者は、RedHat と Miracle の Kernel2.6 からのバックポーティングの範囲の詳細を把握できていないため、ここでは、関連しそうな情報を示す。違いを際立たせるため、システムに負荷がもっともかかる状況のひとつである、データサイズ大・チューニング前の eu=100 の時のものを示す。最初の図が RedHat( 図 4.4-14 )、次の図が Miracle( 図 4.4-15 )、続いて図 4.4-16 が SuSE の vmstat の時系列グラフ(性能の昇順にならべてある)である。性能向上の順にしたがって、CPU の IOWAIT が安定していることがわかる。この意味するところは、それぞれの Kernel 開発者の今後の研究・考察を待ちたいところである。

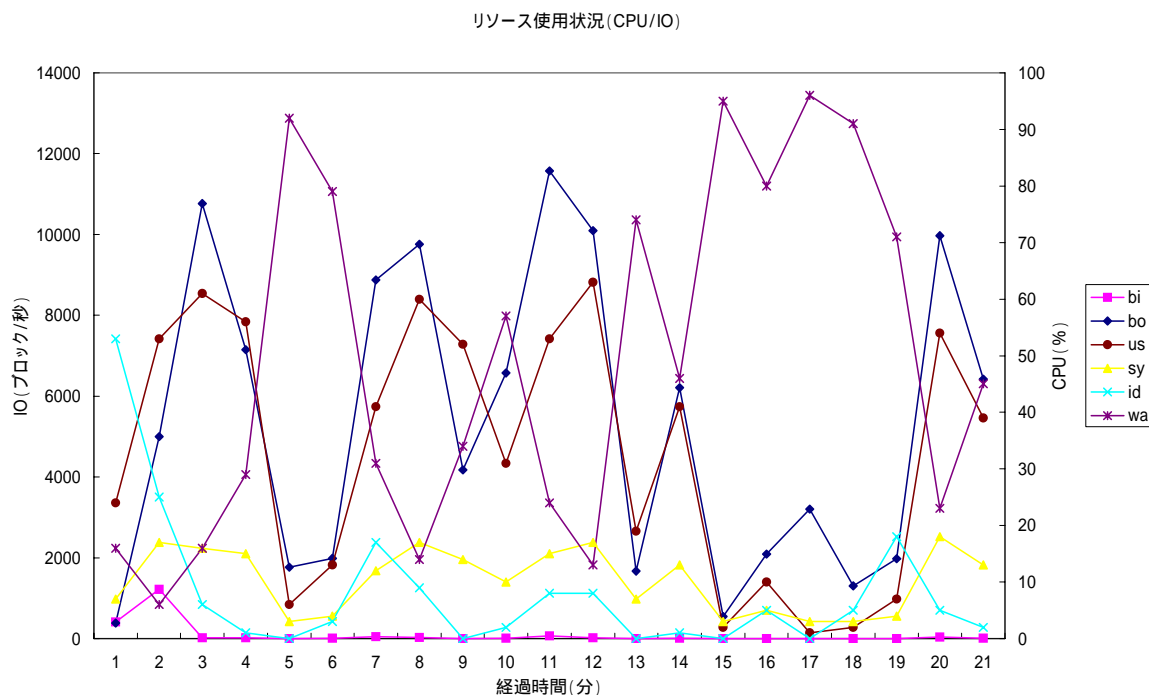


図 4.4-14 eu=100 における vmstat 情報 RedHat データベースサイズ大 チューニング前

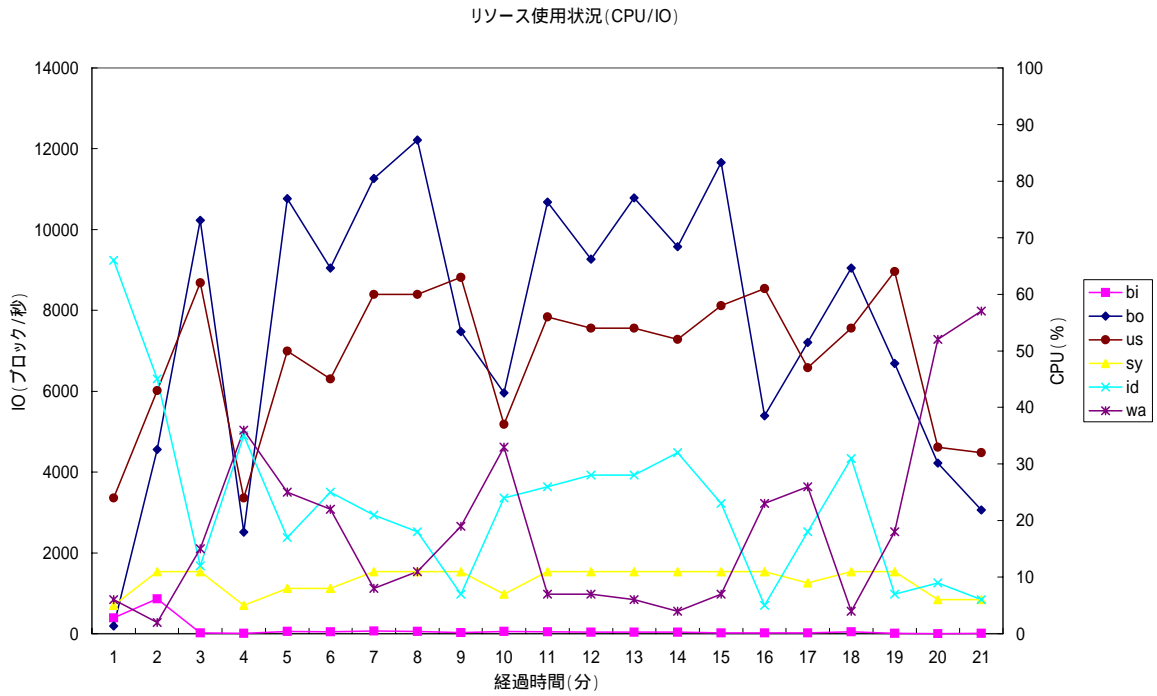


図 4.4-15 eu=100 における vmstat 情報 Miracle データベースサイズ大 チューニング前

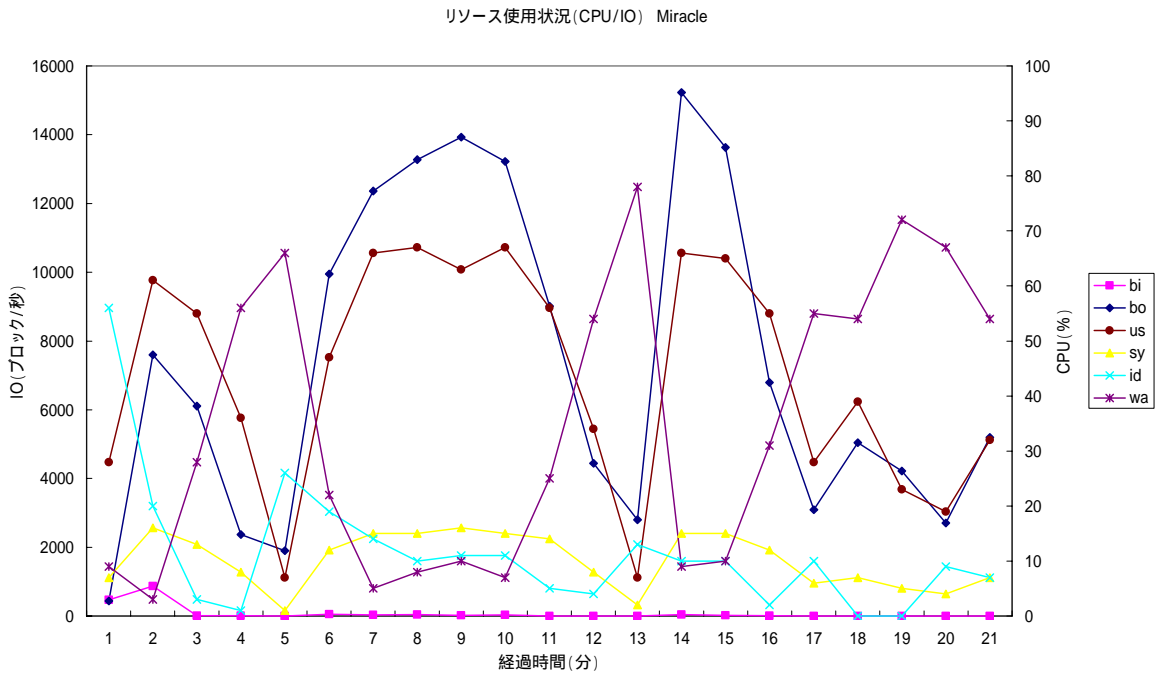


図 4.4-16 eu=100 における vmstat 情報 SuSE データベースサイズ大 チューニング前

## 4.5 まとめ

今回の測定で明らかになったことのサマリを以下に記す。

- DBT-1 によって PostgreSQL の性能測定ができるようになった  
OSDL が提供する DBT-1 v2.1 は、不具合や手順が不明瞭であることにより、そのままでは測定できない。この対策を明らかにし、性能測定が可能になった。また、本測定で用いた HW 環境では、DBT-1 により性能限界・性能特性を測定することが可能であることが確認できた。
- サイズによって大きく性能が劣化すること。  
DBT-1 はデータサイズを可変に設定できる。この機能を利用して、テキストロードデータベースで、約 3 GB ならびに約 3.5 GB のデータサイズのデータベースの性能測定を行った。今回の HW 環境では、3.5 GB において性能が著しく劣化した。ユーザの性能要件により、この性能が受け入れられるかどうかが決まるが、その参考数値（あくまで今回の HW 環境での）を示した。
- チューニングによって性能が大きく向上すること。いずれにせよサイズによる性能の劣化は避けられないこと。  
Vacuum 動作等、PostgreSQL の現実的な運用手順を踏むこと、および、いくつかの問い合わせ文実行プランの最適化により、性能が大きく向上した。その一方で、DBT-1 は一部の問い合わせ文の内容が、データサイズに大きく影響を受けるものになっており、チューニングを施しても、データサイズが大きくなれば、性能は劣化することが観測された。
- ディストリビューションによって安定性・性能に違いがあること。  
あくまで本測定環境 + DBT-1 + PostgreSQL 7.4.6 の上での結果であるが、OS ディストリビューションにより性能や挙動の安定性に違いがあった。

## 5 DBT-1 の商用 RDBMS への適用

### 5.1 概要

オープンソース RDBMS の導入に際し、導入予定者のひとつの疑問として、「これまで慣れ親しんできた商用 RDBMS と比較して、オープンソース RDBMS は機能・性能・拡張性・信頼性の点でどのような違いがあるのか」という質問が容易に予想される。本プロジェクトでは、この質問のうち、性能面、もう少し範囲を特定しておく、Web トランザクション系のアプリケーションの性能面でのオープンソース RDBMS/商用 RDBMS の比較が行えるよう、OSDL が開発した DBT-1 ベンチマークを商用 RDBMS へポータリングした。2004 年 12 月現在、商用 RDBMS は複数のものが利用可能であるが、シェア的にも代表的な Oracle 9i をポータリングターゲットとした。また、ポータリング元の RDBMS は、現時点での DBT-1 が唯一サポートする PostgreSQL 7.4.x とした。

オープンソース RDBMS と商用 RDBMS の性能比較を可能にする、という目的の他、副産物的に、ポータリング作業で得た経験から、PostgreSQL と Oracle 間のポータリングノウハウ集を得た。

以下では、5.2 で、ポータリングの内容やノウハウについての記述、5.3 以降で、実際の利用方法等、測定上実践的な内容について述べた。5.2 はその性質上、開発者向けの記述となっている。実際の測定に興味がある方はまず 5.3 を読みたい。

### 5.2 商用 RDBMS へのポータリング

#### 5.2.1 ポータリング方針

2004 年 12 月時点で DBT-1 最新版の v2.1 の、PostgreSQL7.4.x 向け部分を Oracle9i にポータリングする。PostgreSQL 版とソースを分割できるところはなるべく分割するようにした（過去 OSDL が行った、SAPDB から PostgreSQL へのポータリングと同じ方針）。ポータリング後のソースは、他の RDBMS、例えば PostgreSQL、MaxDB/SAPDB での動作を保障しないことにする。理由は以下の通りである。本プロジェクトでは、同時並行して DBT-1 の MaxDB へのポータリングを行っていること、DBT-1 の v2.1 は PostgreSQL7.4.x をサポートしているはずだが、実は不具合によりビルドができないこと、この二つにより、プロジェクト終了時点で、単一ソースセットが複数の RDBMS の同時サポートを保障することはプロジェクト運営上困難であったことによる。

尚、DBT-1 の MaxDB 版、PostgreSQL 版（オリジナル版）、Oracle 版のソース統合は、DBT-1 の TODO リストのひとつとし、本プロジェクト終了後の課題としている。

## 5.2.2 修正概要

ポーティング対象は大きく分けて3種類である。1つは、ビジネスロジックが記述されているストアードプロシージャ ( PL/PGSQL で記述されている )、二つ目は、DB 接続・ストアードプロシージャをコールする、クライアントエミュレータ/アプリケーションサーバ ( C 言語で記述されている。DB 接続には PostgreSQL のライブラリ LIBPQ が利用されている )、最後の3つ目は、DBT-1 の環境作成・実行などに関連するスクリプト類である。

まず、ストアードプロシージャであるが、PL/PGSQL で記述されているものを、Oracle の拡張 SQL 言語、PL/SQL に移植する。ポーティングのノウハウとしてはここが量的にも質的にも重要であったため、5.2.4 に PL/PGSQL-Oracle PL/SQL 移植ガイドラインとしてまとめられている。

また、DB 接続・ストアードプロシージャコール部分は、LIBPQ を利用していた部分を OCI ( Oracle Call Interface ) に移植した。OCI を選択した理由は、ライセンスへの配慮や Windows での ODBC や OLE などのミドルウェアなどが不要であることにある。

PostgreSQL 版で利用するスクリプト群は、ほぼ等価なものにポーティングしたが、大きく言って次の2点が異なる。まず、1点目は、RDBMS の統計情報収集に関してである。PostgreSQL は、RDBMS に適切な設定をあらかじめしておく、様々な実行時統計情報 ( 例えば、RDBMS 内部のキャッシュのヒット回数など ) が統計テーブルに格納される。これを取り出すスクリプトが DBT-1 PostgreSQL 版では用意されているが、この機能は、Oracle では STATSPACK という Oracle 提供の統計ツールが包含しているため、ポーティングを行わなかった。2点目は、前項で述べたオリジナル版 DBT-1 のビルドスクリプトの不具合と今後のソース統合の問題から、今回は、実行する OS ユーザー名に制約がある。具体的内容は、5.3 以降を参照されたい。

## 5.2.3 修正範囲

表 5.2-1 に修正したソース、追加したソースの一覧を示す。尚、[OraVersion]は、今回作成した DBT-1 Oracle 版のソースのルートディレクトリをあらわす。合計 109 ファイル、ステップ数は約 2 万 5 千であった。

表 5.2-1 DBT-1 Oracle 向け修正範囲

区分	ファイル・ディレクトリ名	(参考)行数
追加	[OraVersion]: Makefile	44
追加	[OraVersion]/appServer: Makefile	74
修正	[OraVersion]/appServer/appServer.c	573
修正	[OraVersion]/appServer/app_threadpool.c	320
追加	[OraVersion]/cache: Makefile	49
修正	[OraVersion]/cache/cache.c	764
追加	[OraVersion]/common: Makefile	13
修正	[OraVersion]/data_collect/analyzeBT.pl	80
修正	[OraVersion]/data_collect/dbt1_slave.pl	424
追加	[OraVersion]/data_collect: oracle	0
追加	[OraVersion]/data_collect/oracle: dbt1.1way.conf	27
追加	[OraVersion]/data_collect/oracle: dbt1_master.pl	1039
追加	[OraVersion]/data_collect/oracle: get_config.sh	85
追加	[OraVersion]/data_collect/oracle: run.config	31
修正	[OraVersion]/data_collect/sys_stats.sh	93
追加	[OraVersion]/datagen: Makefile	16
修正	[OraVersion]/datagen/main.c	1127
追加	[OraVersion]/dbdriver: Makefile	58
修正	[OraVersion]/dbdriver/eu.c	1956
修正	[OraVersion]/dbdriver/main.c	333
修正	[OraVersion]/include/app_threadpool.h	55
修正	[OraVersion]/include/db.h	47
修正	[OraVersion]/include/eu.h	103
追加	[OraVersion]/include: oci_interaction.h	48
追加	[OraVersion]/include: oci_interaction_admin_confirm.h	22
追加	[OraVersion]/include: oci_interaction_admin_request.h	21
追加	[OraVersion]/include: oci_interaction_best_sellers.h	49
追加	[OraVersion]/include: oci_interaction_buy_confirm.h	154
追加	[OraVersion]/include: oci_interaction_buy_request.h	169
追加	[OraVersion]/include: oci_interaction_debug.h	27
追加	[OraVersion]/include: oci_interaction_home.h	35
追加	[OraVersion]/include: oci_interaction_new_products.h	226
追加	[OraVersion]/include: oci_interaction_order_display.h	48
追加	[OraVersion]/include: oci_interaction_order_inquiry.h	21
追加	[OraVersion]/include: oci_interaction_product_detail.h	23
追加	[OraVersion]/include: oci_interaction_search_request.h	24
追加	[OraVersion]/include: oci_interaction_search_results.h	658
追加	[OraVersion]/include: oci_interaction_shopping_cart.h	45
追加	[OraVersion]/interfaces: Makefile	13
修正	[OraVersion]/interfaces/db.c	203
追加	[OraVersion]/interfaces: oci	0
追加	[OraVersion]/interfaces/oci: Makefile	41
追加	[OraVersion]/interfaces/oci: oci_interaction.c	138
追加	[OraVersion]/interfaces/oci: oci_interaction_admin_confirm.c	177
追加	[OraVersion]/interfaces/oci: oci_interaction_admin_request.c	100
追加	[OraVersion]/interfaces/oci: oci_interaction_best_sellers.c	1157
追加	[OraVersion]/interfaces/oci: oci_interaction_buy_confirm.c	545
追加	[OraVersion]/interfaces/oci: oci_interaction_buy_request.c	656
追加	[OraVersion]/interfaces/oci: oci_interaction_home.c	83
追加	[OraVersion]/interfaces/oci: oci_interaction_new_products.c	705
追加	[OraVersion]/interfaces/oci: oci_interaction_order_display.c	976
追加	[OraVersion]/interfaces/oci: oci_interaction_order_inquiry.c	60
追加	[OraVersion]/interfaces/oci: oci_interaction_product_detail.c	136
追加	[OraVersion]/interfaces/oci: oci_interaction_search_request.c	69
追加	[OraVersion]/interfaces/oci: oci_interaction_search_results.c	718
追加	[OraVersion]/interfaces/oci: oci_interaction_shopping_cart.c	816
追加	[OraVersion]: make.common	56

区分	ファイル・ディレクトリ名	
追加	[OraVersion]/scripts: oracle	0
追加	[OraVersion]/scripts/oracle: create dbt1_schema.sh	50
追加	[OraVersion]/scripts/oracle: create fk.sh	5
追加	[OraVersion]/scripts/oracle: create fk.sql	22
追加	[OraVersion]/scripts/oracle: create indexes.sh	5
追加	[OraVersion]/scripts/oracle: create indexes.sql	10
追加	[OraVersion]/scripts/oracle: create sequence.sh	5
追加	[OraVersion]/scripts/oracle: create tables.sh	5
追加	[OraVersion]/scripts/oracle: create tables.sql	22
追加	[OraVersion]/scripts/oracle: delete dbt1_schema.sql	51
追加	[OraVersion]/scripts/oracle: load address.ctl	13
追加	[OraVersion]/scripts/oracle: load author.ctl	12
追加	[OraVersion]/scripts/oracle: load cc_xacts.ctl	15
追加	[OraVersion]/scripts/oracle: load country.ctl	10
追加	[OraVersion]/scripts/oracle: load customer.ctl	23
追加	[OraVersion]/scripts/oracle: load db.sh	75
追加	[OraVersion]/scripts/oracle: load dbproc.sh	9
追加	[OraVersion]/scripts/oracle: load item.ctl	28
追加	[OraVersion]/scripts/oracle: load orders.ctl	17
追加	[OraVersion]/scripts/oracle: load order_line.ctl	12
追加	[OraVersion]/scripts/oracle: set_run_env.sh	17
追加	[OraVersion]/storedproc: oracle	0
追加	[OraVersion]/storedproc/oracle: addToSC.sql	89
追加	[OraVersion]/storedproc/oracle: admin_confirm.sql	81
追加	[OraVersion]/storedproc/oracle: admin_request.sql	50
追加	[OraVersion]/storedproc/oracle: best_sellers.sql	678
追加	[OraVersion]/storedproc/oracle: best_sellers_new.sql	686
追加	[OraVersion]/storedproc/oracle: buy_confirm.sql	456
追加	[OraVersion]/storedproc/oracle: buy_request.sql	687
追加	[OraVersion]/storedproc/oracle: createSC.sql	52
追加	[OraVersion]/storedproc/oracle: DigSyl.sql	68
追加	[OraVersion]/storedproc/oracle: getCustInfo.sql	101
追加	[OraVersion]/storedproc/oracle: getCustName.sql	36
追加	[OraVersion]/storedproc/oracle: getPromoImages.sql	87
追加	[OraVersion]/storedproc/oracle: getPromoImages_Id.sql	34
追加	[OraVersion]/storedproc/oracle: getPromoImages_Thumbnail.sql	34
追加	[OraVersion]/storedproc/oracle: getSCDetail.sql	379
追加	[OraVersion]/storedproc/oracle: getSCSubTotal.sql	44
追加	[OraVersion]/storedproc/oracle: home.sql	70
追加	[OraVersion]/storedproc/oracle: InsertCust.sql	111
追加	[OraVersion]/storedproc/oracle: new_products.sql	677
追加	[OraVersion]/storedproc/oracle: ora_type_pkg.sql	13
追加	[OraVersion]/storedproc/oracle: order_display.sql	516
追加	[OraVersion]/storedproc/oracle: order_inquiry.sql	35
追加	[OraVersion]/storedproc/oracle: product_detail.sql	76
追加	[OraVersion]/storedproc/oracle: refreshSC.sql	45
追加	[OraVersion]/storedproc/oracle: search_results_author.sql	695
追加	[OraVersion]/storedproc/oracle: search_results_subject.sql	684
追加	[OraVersion]/storedproc/oracle: search_results_title.sql	681
追加	[OraVersion]/storedproc/oracle: shopping_cart_prc.sql	379
追加	[OraVersion]/storedproc/oracle: updateSC.sql	87
追加	[OraVersion]/storedproc/oracle: _installprocedure.sql	39
追加	[OraVersion]/tools: Makefile	37
修正	[OraVersion]/tools/interaction_test.c	1317
追加	[OraVersion]/wgen: Makefile	61
合計	109ファイル	25021

## 5.2.4 PL/PGSQL—Oracle PL/SQL 移植ガイドライン

今回のストアードプロシージャのポーティング作業で必要となった、移植対象部分の移植方法についてガイドラインとしてまとめ、変更の方法を統一した。以下にそのガイドラインを記す。尚、ここで述べる方法が、一般的 PL/SQL PL/pgSQL 間のポーティングの際の唯一の方法ではなく、かつ、網羅的でもないことに留意されたい。繰り返すが、ポーティング元は PL/pgSQL、ポーティング先は PL/SQL である。

### 5.2.4.1 SET コマンド

- PL/pgSQL

```
¥set AUTOCOMMIT off
```

- PL/SQL

```
set AUTOCOMMIT off
```

### 5.2.4.2 CREATE OR REPLACE FUNCTION 文の引数

- PL/pgSQL  
PostgreSQL では、名前付きパラメータはない。(OUT,INOUT の概念もない)
- PL/SQL  
Oracle では、名前付きパラメータにし、IN,OUT,INOUT を指定する。

### 5.2.4.3 FROM 句のない SELECT 文(ファンクションの実行)

- PL/pgSQL  
PostgreSQL では他のファンクションを SELECT 文で実行する場合 FROM 句は指定しない。例えば、

```
SELECT getSCSubTotal(_sc_id) INTO sc_subtotal;
```

- PL/SQL

Oracle では、FROM 句がないとコンパイルでエラーとなるので、「FROM DUAL」を付加するようにする。例えば、

```
SELECT getSCSubTotal(p_sc_id) INTO sc_subtotal FROM dual;
```

### 5.2.4.4 トランザクションのアボート(RAISE EXCEPTION)の置き換え

- PL/pgSQL  
PostgreSQL では RAISE EXCEPTION("エラーメッセージ(%)",変数)を指定すると '%'が変数で置き換えられる。例えば、

```
RAISE EXCEPTION 'itemcount(%) > 100', _itemcount;
```

- PL/SQL

Oracle では、RAISE\_APPLICATION\_ERROR(ユーザ指定コード,'エラーメッセージ')となるので変数はパイプ('|')でつないで指定する。

```
RAISE_APPLICATION_ERROR (-20000,'itemcount(' || itemcount || ') > 100');
```

#### 5.2.4.5 CREATE OR REPLACE FUNCTION 文の戻り値

- PL/pgSQL  
PostgreSQL では、RETURNS numeric(10,0)等としている。
- PL/SQL  
"RETURN"とし、桁数を指定せず、RETURN numeric 等とする。

#### 5.2.4.6 CREATE OR REPLACE FUNCTION 文の開始と終了

- PL/pgSQL  
PostgreSQL では、「AS」で始まり、「LANGUAGE 'plpgsql;」で終了。
- PL/SQL  
「IS」で始まり、「/ show errors;」で終了。  
(show errors はコンパイル時のエラー表示のため付加する。)

#### 5.2.4.7 SQL 文実行後の NOT FOUND の判定

- PL/pgSQL  
PostgreSQL では、IF FOUND , IF NOT FOUND で判定している。
- PL/SQL  
(1) カーソルでフェッチ直後 refcur%FOUND を使用し、WHILE refcur%FOUND LOOP のようにする。  
(2) 単独の SELECT 文実行後 IF SQLCODE = 0 , IF SQLCODE = 100 で判定する。  
(3) 検索後、判定までの間に別の SQL 実行が行なわれる場合 FOUND,NOT FOUND の結果をフラグに移し変えておき、判定する。

#### 5.2.4.8 関数 NOW()

- PL/pgSQL  
システム日付、時刻が設定される。
- PL/SQL  
CURRENT\_TIMESTAMP に置き換える。

#### 5.2.4.9 シーケンスのインクリメントと、変数への代入

- PL/pgSQL  
PostgreSQL では、  
a :=nextval("xxxxx") のように、変数に直接代入できる。
- PL/SQL  
Oracle では、ダミーの SQL 文を作成し、以下のようにする。  
select xxxxx.NEXTVAL INTO a FROM DUAL

#### 5.2.4.10 剰余演算子

- PL/pgSQL  
PostgreSQL では、以下のように "%" を使用する。

```
d := d % base;
```

- PL/SQL

```
d := d MOD p_base;
```

又は、

```
d := MOD( d , p_base );
```

#### 5.2.4.11 型キャスト使用の SQL 文

- PL/pgSQL

PostgreSQL では、以下のように "::" を使用する。

```
SELECT _c_discount::NUMERIC(5,2),  
_sc_sub_total::NUMERIC(17,2),  
INTO rec;  
RETURN rec;"
```

- PL/SQL

型キャストの :: 以降を削除、又は、

cast(xxx as NUMERIC(5,2)) としてカーソルを設定し、rec cur1%ROWTYPE; と変数定義し、from dual を付加する。

#### 5.2.4.12 複数結果及び RECORD 型の値を返すストアドファンクション

- PL/pgSQL

PostgreSQL では、ストアドファンクションで、RECORD 型の値を返している。

- PL/SQL

Oracle では、以下のとおりとする。

ストアドファンクションからストアドプロシージャに変換する。例えば、

```
CREATE OR REPLACE PROCEDURE admin_confirm( . . . . . ) AS
```

呼ばれる側では、引数に OUT で RECORD 型で返すカラムを全て指定する。

呼び出し側では、SELECT 文の INTO に OUT で指定したカラムを指定する。

#### 5.2.4.13 テンポラリテーブルの作成およびデータの挿入

- PL/pgSQL

PostgreSQL では、テンポラリテーブルの作成およびデータの挿入として「CREATE TEMP TABLE . . . SELECT . . . 」を直接記述することができる。

- PL/SQL

Oracle ( PL/SQL ) では、CREATE 文を直接記述することができないので、EXECUTE IMMEDIATE で CREATE TABLE 文と INSERT SELECT 文に分けて記述する。( 予め実行ユーザに CREATE TABLE 権限を明示的に与える必要がある。)

```
EXECUTE IMMEDIATE (CREATE GLOBAL TEMPORARY TABLE . . . );  
EXECUTE IMMEDIATE (INSERT INTO . . . SELECT . . . );
```

#### 5.2.4.14 パラメタの配列渡し

- PL/pgSQL  
PostgreSQL では、NUMERIC[] として配列を渡している
- PL/SQL  
PACKAGE を使用して、型の宣言のみ行い、呼ぶ側と呼ばれる側で同じ型を使用する。

```
create or replace package ora_type_pkg is
TYPE typ_i_id_array IS TABLE OF NUMBER(10,0) INDEX BY BINARY_INTEGER;
TYPE typ_qty_array IS TABLE OF NUMBER(3,0) INDEX BY BINARY_INTEGER;
end;
```

#### 5.2.4.15 時間の計算

- PL/pgSQL  
PostgreSQL では、\_C\_Expire := now() + "02:00";などとして時間の加算をしている。
- PL/SQL  
Oracle では、

```
p_C_Expire := CURRENT_TIMESTAMP+' 02:00';
```

とすると、「PLS-00307: このコールに一致する '+' が複数宣言されています。」となる。よって以下のように変換する。

```
SELECT CURRENT_TIMESTAMP + 2/24 into p_C_Expire from dual;
```

#### 5.2.4.16 OCIBindByName()の設定

以降、ストアードプロシージャそのものについてはないが、移植上重要だったので掲載する。

C 言語の中から、OCI 経由でストアードプロシージャを実行する際、プロシージャの引数をバインドするために、OCIBindByName()を使用する。このとき、OUT で返されるデータの型について何を指定するのか明確にする。以下の方法を採用する。

- OUT の longlong 型  
char の LONGLONG\_MAXLEN (21)の領域に SQLT\_STR で受取り、atoll()で longlong 型の領域に設定する。
- OUT の char 型  
char の領域に SQLT\_AVC で受取り strcpy で char の領域に設定する。
- OUT の varchar2 型  
char の領域に SQLT\_STR で受取り strcpy で varchar2 の領域に設定する。
- OUT の duple 型  
double の領域に SQLT\_FLT で受取り duple の領域に設定する。
- OUT の date 型  
char の 10 バイトの領域に SQLT\_STR で受取り strcpy で char の領域に設定する。

#### 5.2.4.17 配列の入力引数が存在するストアプロシージャの場合のバインド

C 言語のソースから OCI 経由でストアプロシージャを実行する際、プロシージャの引数をバインドするために、OCIBindByName() を使用するが、引数が配列の場合は OCIBindByPos() を使用する。OCIBindByPos() の指定方法について明確にしておく。これは、[ORAVRESION]/interfaces/oci/oci\_interaction\_shopping\_cart.c で採用された方法である。

まず、配列 Type 定義を Oracle 側で行う。(ora\_type\_pkg.sql)

```
create or replace package ora_type_pkg
is
-- for shopping_cart
TYPE typ_i_id_array IS TABLE OF NUMBER(10,0) INDEX BY BINARY_INTEGER;
TYPE typ_qty_array IS TABLE OF NUMBER(3,0) INDEX BY BINARY_INTEGER;
--
end;
/
```

Shopping\_cart ストアドプロシージャは以下のような定義となっている。  
(shopping\_cart\_prc.sql)

```
CREATE OR REPLACE PROCEDURE shopping_cart_prc (
  C_ID
  p_sc_id IN NUMBER,
  p_itemcount      IN NUMBER,
  p_add_flag       IN NUMBER,
  p_i_id IN NUMBER,
  p_pp_i_id        IN NUMBER,
  p_i_id_array     IN ora_type_pkg.typ_i_id_array,      <-配列
  p_qty_array      IN ora_type_pkg.typ_qty_array,      <-配列
  t_sc_id OUT NUMBER,
  itemcount       OUT NUMBER,
  pp_i_id1 OUT NUMBER,
  :
  :
```

対応するヘッダーファイル oci\_interaction\_shopping\_cart.h の内容は以下の通りである。

```
#ifndef _OCI_INTERACTION_SHOPPING_CART_H
#define _OCI_INTERACTION_SHOPPING_CART_H

#include <oci_interaction.h>

#define STMT_SC "BEGIN shopping_cart("¥
":c_id, :p_sc_id, :p_itemcount, :p_add_flag, :p_i_id,"¥
```

```

":p_pp_i_id, :p_i_id_array, :p_qty_array,"¥
":t_sc_id, :itemcount,"¥
":pp_i_id1, :pp_i_t1, :pp_i_id2, :pp_i_t2, :pp_i_id3, :pp_i_t3, :pp_i_id4, :pp_i_t4, :pp_i_id5, :pp_i_t5,"¥
:
:

```

C 言語ソース oci\_interaction\_shopping\_cart.c からは以下のようにバインドして呼び出す。

```

#include <oci_interaction_shopping_cart.h>
int execute_shopping_cart(struct db_context_t *dbc, struct shopping_cart_t *data) {
    int i;
    static OCIStmt *stmthp;          /* Statement Handle */
    static OCIBind *bndp[140];      /* Bind Handle */
    sb2 *ind;
    ub4 retcnt1=20;
    :
    :
    if (checkerr(dbc->errhp, OCIBindByName(stmthp,
        (OCIBind **) &bndp[5],
        dbc->errhp, (OraText *) ":p_pp_i_id", strlen(":p_pp_i_id"),
        (dvoid *) &p_pp_i_id, sizeof(p_pp_i_id),
        SQLT_INT, 0, 0, 0, 0, OCI_DEFAULT), FUNC_LABEL_13) != OK) { return ERROR; }
    if (checkerr(dbc->errhp, OCIBindByPos(stmthp, (OCIBind **) &bndp[6],
        dbc->errhp, 7, (dvoid *) p_i_id_array, sizeof(p_i_id_array[0]),
        SQLT_INT, 0, 0, 0, SHOPPING_CART_ITEMS_MAX, (ub4 *) &retcnt1, OCI_DEFAULT),
        FUNC_LABEL_13) != OK) { return ERROR; }
    if (checkerr(dbc->errhp, OCIBindByPos(stmthp, (OCIBind **) &bndp[7],
        dbc->errhp, 8, (dvoid *) p_qty_array, sizeof(p_qty_array[0]),
        SQLT_INT, 0, 0, 0, SHOPPING_CART_ITEMS_MAX, (ub4 *) &retcnt1, OCI_DEFAULT),
        FUNC_LABEL_13) != OK) { return ERROR; }
    :
    :
}

```

補足

- 中括弧部分は、配列のバインドに関する部分である。
- dbc->errhp のあとの 7、8 は、配列の引数のポジション(位置)。
- SHOPPING\_CART\_ITEMS\_MAX は、配列の最大要素数。
- &retcnt1 は、実際の要素の数のポインタ。

#### 5.2.4.18 実行するストアードプロシージャの入出力引数の初期設定

ORACLE の OCI でストアードプロシージャを実行する際、引数としてバインドした変数の初期化を忘れない。

#### 5.2.4.19 整数の演算

- PL/pgSQL

```
digits := d / base;
```

として、全て整数の演算を行なった場合、digits には少数を切り捨てた値が入る。しかし、Oracle では、小数第一位を四捨五入した結果が入るため、結果が異なる。

- PL/SQL

```
digits := FLOOR( d / base );  
digits := TRUNC( d / base );
```

と変換し、小数以下を明示的に切り捨てる。

### 5.3 環境定義

#### 5.3.1 システム構成

今回の評価では、表 5.3-1 に示される環境を利用した。システム構成は、以下の通りである。

表 5.3-1 ハードウェア

製品名	Dell PowerEdge4600
プロセッサ	インテル Xeon プロセッサ 2.40GHz、2CPU
メモリ	12Gbyte(ただし grub.conf で 4GB に制限している)
ハードディスク	内蔵ドライブベイに、UltraSCSI320 72GB 8 台

Oracle では、ログの書き込み先や複数の表領域などを別のハードディスクに割り当てる機能があるが、そのような機能を持たない PostgreSQL7.4.x を念頭におき、表 5.3-2 のような単純な構成をとった。今回は特にディスクのベンチマーク測定は行っていないが、一般に RAID0 では、性能的には乱暴な目安であるが、シーケンシャル Read/Write の速度は 60MB/s \* ディスクの本数あるいは、Raid ボードの最大転送速度の低いほう、ランダム Read/Write は 5MB/s \* ディスクの本数の性能が得られる可能性がある。

表 5.3-2 ディスク構成

72GB	/	20GB
	swap	2GB
72GB	/work (ロード用データ等)	すべて
72GB*6 本 RAID0	/data	すべて

## 5.3.2 インストール

### 5.3.2.1 Linux・Oracle のインストール

本章で対象にする Oracle9i は、その枝バージョンによって、動作・インストールがサポートされる OS ディストリビューション・OS バージョンが異なり、OS インストールの手順が大きく異なる場合がある。このため、ここでは詳細の方法については記述しない。ターゲットとなる Oracle のバージョンのマニュアルと Metalink などの Oracle のサポート情報を参照し、インストールする。

尚、以下の手順では、Oracle として、Oracle 9i Database Release2 (9.2.0.4.0)を対象としている。測定環境の OS としては、RedHat EL3、Miracle Asian Linux v3.0、SuSE ES9 の3つとした。

### 5.3.2.2 DBT-1 のインストール

本章では、今回開発した dbt1-v2.1-ora-1.0 をインストール対象とする。

#### 5.3.2.2.1 前提条件

dbt1-v2.1-ora-1.0 が前提とする条件は、以下の通りである。

- ベンチマークユーザとして、Linux アカウント dbt1si を作成し、そのホームディレクトリを/home/dbt1si/とすること。
- dbt1-v2.1-ora-1.0.tar.gz の展開場所は、/home/dbt1si/の直下であること。
- Oracle Database Configuration Assistant や Oracle Net8 Configuration Assisitant などを使って、SID=dbt1 とした Oracle データベースを作成、TNS Lisnter の設定をしてあること。尚、今回のデータベースは専用データベースモードで作成した。

#### 5.3.2.2.2 ベンチマークユーザ Linux ユーザの作成

以下の方法で Linux ユーザを作成する。root で、

```
# useradd dbt1si -d /home/dbt1si -g [Oracle の管理者グループ]
```

を実行する。[Oracle の管理者グループ]は Oracle のインストール時に決めたものを利用する。また、/home/dbt1si/.bash\_profile も以下のように編集しておく。

<設定例>

```
ORACLE_BASE=/home/oracle
ORACLE_HOME=$ORACLE_BASE/product/9.2.0
ORACLE_SID=dbt1
NLS_LANG=Japanese_Japan.JA16EUCTILDE
ORA_NLS33=$ORACLE_HOME/ocommon/nls/admin/data
PATH=$PATH:$ORACLE_HOME/bin
ORACLE_DOC=$ORACLE_HOME/doc
CLASSPATH=$ORACLE_HOME/JRE:$ORACLE_HOME/jlib:$ORACLE_HOME/rdbms/jlib
```

```
CLASSPATH=$CLASSPATH:$ORACLE_HOME/network/jlib
CLASSPATH=$CLASSPATH:$ORACLE_HOME/jdbc/lib/classes12.zip
CLASSPATH=$CLASSPATH:$ORACLE_HOME/jdbc/lib/nls_charset12.zip
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/ctx/lib:$ORACLE_HOME/jdbc/lib

export PATH ORACLE_BASE ORACLE_HOME ORACLE_SID NLS_LANG ORA_NLS33
export ORACLE_DOC CLASSPATH LD_LIBRARY_PATH
```

#### 5.3.2.2.3 ベンチマーク実行 Oracle ユーザの作成

DBT-1 のインストールをする前に、ベンチマーク実行用の Oracle ユーザを作成し、dba 権限を与える。sqlplus を起動し、

```
sqlplus> connect sys/[sys のパスワード] as sysdba
sqlplus> startup;
sqlplus> create user dbt1 identified by dbt1 default tablespace [デフォルトデータ表領域名];
sqlplus> grant dba to dbt1;
```

#### 5.3.2.2.4 Oracle 向け DBT-1 パッケージのインストール

最初に dbt1si ユーザにて、/home/dbt1si/ 以下に Oracle 版 DBT-1 の tar ボール dbt1-v2.1-ora-1.0.tar.gz を展開する。tar ボールは、/tmp ディレクトリにあるものとする。

```
$ cd ~
$ tar xzf /tmp/dbt1-v2.1-ora-1.0.tar.gz
```

次に、make 環境設定ファイル(/home/dbt1si/dbt1-v2.1/make.common)の編集をする。デフォルトでは、\$ORACLE\_HOME=/home/oracle/product/9.2.0 と仮定して作成されているので、\$ORACLE\_HOME が上記設定と異なる場合は「/home/oracle/product/9.2.0」の部分を実体の\$ORACLE\_HOME に合わせて編集する。

最後に、make と make install を行う。PostgreSQL 版の DBT-1 ユーザマニュアルにあるような configure は行わない。ディレクトリ「/home/dbt1si/dbt1-v2.1」において make コマンドを実行する。(configure は行わない)

```
$ cd ~/dbt1si
$ make
$ make install
```

これで DBT-1 のインストールの完了である。

### 5.3.3 パラメータの設定

#### 5.3.3.1.1 grub の設定

今回測定の際に用いた環境では、12GBのメモリを搭載した IA32 PC サーバを利用した。現在、オープンソース RDBMS を搭載した DB サーバにこれだけのメモリを搭載する事はあまり一般的でないように思われ、これを/etc/grub.conf のブートパラメータ(mem)で 4 GB に制限した。通常はこの設定は必要ない。

#### 5.3.3.2 Oracle

すでに Oracle 上で作成した DBT1 データベースの設定を以下の通りとする。他のパラメータはデフォルトのままである。以下の設定は、本章では、チューニング前のデフォルトパラメータとして扱った。

Oracle 9i からは、sqlplus などからこれらの値を変更できるようになっているが、ここでは、Oracle8i 以前と同じ方法、すなわち、\$ORACLE\_HOME/dbs/spfiledbt1.ora を編集する。特に変更したパラメータは表 5.3-3 の通りである。

表 5.3-3 Oracle パラメータ

パラメータ名	値
max session	335
db_cache_size	25165824
sort_area_size	524288
db_cache_advice	ON
db_writer_processes	1
log_buffer	524288
log_parallelism	1
open_cursors	300
processes	150
sga_max_size	135337420
shared_pool_size	83886080
pga_aggregate_target	25165824
object_cache_optimal_size	102400

パラメータの反映のために、DBMS のリスタートを行う。Sqlplus から、

```
sqlplus> connect sys/[sys のパスワード] as sysdba
sqlplus> shutdown
sqlplus> startup
```

#### 5.3.3.3 OS

本章での測定では、OS のパラメータは以下のようにした。特に、最大ファイルディスクリプタ数(-n)は 4096 としないと、後述する同時接続ユーザ数を数千まであげた場合、エラーによって測定不可能になる場合があるため注意する。表 5.2-1 に ulimit -a の出力結果を示す。

表 5.3-4 ulimit a の出力結果

RedHat EL AS3	SuSE ES9	Miracle Asian Linux
core file size (blocks, -c) 0	core file size (blocks, -c) 0	core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited	data seg size (kbytes, -d) unlimited	data seg size (kbytes, -d) unlimited
file size (blocks, -f) unlimited	file size (blocks, -f) unlimited	file size (blocks, -f) unlimited
max locked memory (kbytes, -l) 4	max locked memory (kbytes, -l) unlimited	max locked memory (kbytes, -l) 4
max memory size (kbytes, -m) unlimited	max memory size (kbytes, -m) unlimited	max memory size (kbytes, -m) unlimited
open files (-n) 4096	open files (-n) 4096	open files (-n) 4096
pipe size (512 bytes, -p) 8	pipe size (512 bytes, -p) 8	pipe size (512 bytes, -p) 8
stack size (kbytes, -s) 10240	stack size (kbytes, -s) unlimited	stack size (kbytes, -s) 10240
cpu time (seconds, -t) unlimited	cpu time (seconds, -t) unlimited	cpu time (seconds, -t) unlimited
max user processes (-u) 7168	max user processes (-u) 30207	max user processes (-u) 7168
virtual memory (kbytes, -v) unlimited	virtual memory (kbytes, -v) unlimited	virtual memory (kbytes, -v) unlimited

また、カーネルパラメータは以下の通りであった。

- cat /proc/sys/kernel/sem の出力結果
  - RedHat EL AS3  
250 32000 100 128
  - SuSE ES9  
1250 32000 100 256
  - Miracle Asian Linux  
256 32000 100 142
- cat /proc/sys/kernel/shmmax の出力結果
  - RedHat EL AS3 ・ SuSE ES9  
2147483648
  - Miracle Asian Linux  
1717986918

## 5.4 評価手順

### 5.4.1 データベースの作成

#### 5.4.1.1 サイズの決定

DBT-1 では、データサイズを決定するファクターとして、ITEM(商品)数と、CUSTOMER(顧客)数がある。ここでは説明のため、それぞれ、10000、1000 とする(参考:この設定で総データは、テキスト換算で約 3GB)。測定者の興味(ユーザ数が多いが、商品が少ないアプリケーションを模したい、など)によりこれらの値を決定する。

#### 5.4.1.2 ロードデータの生成

第三章の 3.1 で述べた通りの手順で作成する。環境変数 \$DBT1\_RAWDATA (たとえば、dbt1si ユーザに読み書きパーミッションを与えたディレクトリ/work/medium)を設定し、ロード用テキストデータの実体の在り処、とすると、dbt1si ユーザで、次を実行する。

```
$ cd ~/dbt1-v2.1/datagen
$ ./datagen -d ORACLE -i 10000 -u 1000 -p $DBT1_RAWDATA
```

3GB のデータを作成するには、環境によっては数時間かかることを留意されたい。また、実際に生成されたデータファイルには/tmp からシンボリックリンクが張られる。データロードには、有無を言わず/tmp のパスが使用されることに注意されたい。

次に、以下の sql ファイルが datagen によって生成されたことをファイルのタイムスタンプから確認する。

```
$ ls -l /home/dbt1si/dbt1-v2.1/scripts/oracle/create_sequence.sql
```

#### 5.4.1.3 データベースの生成

いよいよ DBT-1 のデータベースの中身、すなわちスキーマを作成する。スクリプト実行時に、内部で「/home/dbt1si/dbt1-v2.1/scripts/oracle/set\_run\_env.sh」を利用する。これまでの手順に従っているのならば、このファイルを編集する必要はない。念のため、このファイルの中身を見ておこう。以下は抜粋である。

```
export ORA_USER=dbt1
export ORA_PASS=dbt1
export ORA_SID=dbt1
export ORA_LOAD_LOGPATH=/tmp
export ORA_LOAD_ERRPATH=/tmp
export ORA_LOAD_DATAPATH=/tmp
```

```
export DBT1_PERL_MODULE=/home/dbt1si/dbt1-v2.1/perlmodules
```

次に、特にデータやインデックスをどの表領域に格納するかカスタマイズしたい場合には、~/dbt1-v2.1/scripts/oracle/の下、

- create\_tables.sql
- create\_indexes.sql

の SQL 文をカスタマイズ ( Storage 句の変更 ) する。デフォルト表領域でよければ変更の必要はない。

ようやくスキーマの作成である。dbt1si ユーザにて、以下を実行し、エラーがなく終了すれば OK である。

```
$ cd /home/dbt1si/dbt1-v2.1/scripts/oracle
$ ./create_dbt1_schema.sh
```

尚、万一失敗したときのために、スキーマを削除する SQL ( delete\_dbt1\_schema.sql ) を用意してある。同一ディレクトリから、sqlplus を起動し、@delete\_dbt1\_schema を実行する。

```
$ sqlplus dbt1/dbt1@dbt1

SQL*Plus: Release 9.2.0.4.0 - Production on 月 Dec 27 19:57:10 2004

Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

Oracle9i Enterprise Edition Release 9.2.0.4.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.4.0 - Production
に接続されました。
SQL> @delete_dbt1_schema.sql
```

#### 5.4.2 DBT-1 パラメータの設定

ここではあらかじめ DBT-1 に準備される起動スクリプトを使って DBT-1 を動かすことを想定する ( すると、システム稼動統計などを裏で取得してくれる )。この場合、\$DBT1\_HOME/data\_collect/oracle/run.config を編集する。それぞれの項目の意味は PostgreSQL 版と全く同じなので、4 章を参照されたい。ただし、「bdbname」「username」「password」の項目は、Oracle の SID , ユーザ名、パスワードに設定しておく ( これまでの設定方法に従っていれば、変更の必要はない )。

表 5.4-1 DBT-1 の測定パラメータ

#	項目名	値
1	# database config	
2	items	10000
3	gcustomers	2,880,000
4	dbhost	localhost
5	bdbname	DBT1
6	username	pgsql
7	password	pgsql
8	# cache config	
9	cache	0
10	xcache_host	localhost
11	lcache_port	9999
12	mconnection	10
13	# appServer config	
14	appserver	1
15	server_host0	localhost
16	nserver_port0	9992
17	q_size0	1000
18	a_size0	1000
19	rconnection0	80
20	# dbdriver config	
21	ydriver_host0	localhost
22	vrates0	100
23	eus0	100
24	zduration0	1200
25	think_time0	7.2
26	jdriver_server_host0	localhost
27	kdriver_server_port0	9992
28	out_dir	/tmp
29	db_param	-c tcpip_socket=on
30	redirect_tmp	1
31	redirect_xlog	1

DBT-1 は第 8 章にあるように、Web 2 層モデル、Web 3 層モデルのどちらでも動作させることが可能であるが、現在世間でよく採用されている Web 3 層モデルを測定対象としている。すなわち、dbdriver ( Web クライアントエミュレータ ) <-> AppServer ( コネクションプール付 ) <-> DBMS の構成である。この構成における、今回の測定の標準設定は ( チューニングなどによりいくつかの項目を変更を適宜しているが ) 表 5.4-1 の通りである。

ランザクション作業領域の容量が不足すると、DB とは関係の無い箇所で処理待ちが生じるため、Think\_time の値にもよるが、上記のパラメータのうち、q\_size0 と z\_size0 は、eu の設定値以上の値を指定することが望ましい。

尚、同時接続ユーザ数や、appServer の DB コネクション数などの DBT-1 の実行時設定は、dbdriver や appServer を直接起動する場合は、それぞれの起動オプションで与えることができる。パラメータ「dbname」に Oracle の SID を指定すること以外は PostgreSQL 版と全く同一のオプションで動作する。よって詳しくは PostgreSQL 版の DBT-1 マニュアル

ルを参照されたい。尚、「dbname」「username」「password」のパラメータを省略すると、全て「dbt1」として値がセットされる。

<例>

```
$. /appCache --host localhost --dbname dbt1 --username dbt1 --password dbt1 --port 9999 --db_connection 10 --item_count 1000
```

```
$. /appServer --host localhost --dbname dbt1 --user dbt1 --password dbt1 --server_port 9992 --db_connection 20 --txn_q_size 100 --txn_array_size 100 --item_count 1000 --access_cache --cache_host localhost --cache_port 9999
```

```
$. /dbdriver --server_name localhost --port 9992 --item_count 1000 --customer_count 8640 --emulated_users 100 --rampup_rate 60 --think_time 1.6 --duration 900
```

### 5.4.3 DBT-1 の実行

#### 5.4.3.1 データベースサーバの起動

まず、ORACLE 用 DBT-1 データベースを起動する必要がある。起動している場合は必要ないが、Oracle 内部のキャッシュなどをクリアするには、再起動することが必要である。dbt1si ユーザで、sqlplus を起動し、

```
sqlplus> connect sys/[sys のパスワード] as sysdba
sqlplus> startup;
```

TNS リスナーがあがっていない場合は、以下の方法で起動してからデータベースを起動する。

```
$. lsnrctl start
```

#### 5.4.3.2 DBT-1 アプリサーバの起動

dbt1si ユーザで、

```
$. cd ~/dbt1-v2.1/data_collect
$. ./dbt1_slave.pl
```

を実行する。

#### 5.4.3.3 DBT-1 クライアントエミュレータの起動

DBT-1 のワークベンチをスタートする。dbt1si ユーザで、

```
$. cd ~/dbt1-v2.1/data_collect/oracle
$. ./dbt1_master.pl -f run.config
```

を実行する。尚、-f オプションでコンフィギュレーションファイルを指定できる。複数のケースについてベンチマークをする場合、コンフィギュレーションを別ファイルに保存し、適宜-f オプションで指定するとよい。

## 5.5 実行

### 5.5.1 実行結果の収集

ベンチマーク終了後、PostgreSQL 版と同様に、実行結果が run.config ファイルの out\_dir に記述した出力ディレクトリに出力される。ただし、PostgreSQL 版とは違い、DB の状態をスナップしていないので、/tmp 以下に「db\_stat」「ipcs」ディレクトリは生成されない。

主に参照されるべきは、BT というファイルの中に現れる、トランザクション種別毎の比率、平均レスポンス（秒）ならびに、BT/秒（ポゴトランザクション/sec = web 経由のリクエスト数/sec）である。この他にいくつかの情報が取得できる。代表的なものを表 5.5-1 に示す。

表 5.5-1 実行結果が記録されるファイル

ファイル名	概要	備考
BT	トランザクション種別毎の比率・平均応答時間、ポゴトランザクション	要はスループット。
config.txt	実行時のOSパラメータやCPU、メモリサイズなどのシステム環境情報、DBT-1の設定の記録。	
param.out	PostgreSQLの設定パラメータ一覧	
run.meminfo0.out	DBT-1起動時のメモリ関連情報	
run.meminfo1.out	DBT-1終了後のメモリ関連情報	
indexes.out	DBT-1起動時点でのPostgreSQLのユーザインデックスの利用統計	
run.iostat.out	システムのIO関連の情報	iostat d 60と等価。man.iostatを参照のこと。
run.vmstat.out	システムのメモリ・IO・CPUに関連する統計。	vmstat 60と等価。Man vmstatを参照のこと。
result.mix.log	トランザクション毎のトランザクション種別、要求時刻、返答時刻。	これを編集したものがBT。
ips.csv	30秒区間ごとの平均トランザクション数の推移。	入力はresult.mix.log

## 5.6 まとめ

今回開発した、DBT-1 と本章で述べた手順により、Oracle のトランザクション性能を測定できることが確認できた。ただし、適切なデータベースチューニングや HW のスペックにより、性能限界が DBT-1 の dbdriver 単体の負荷能力を超える場合も考えられる（eu を大きな値にしても性能限界に到達しないような事態）。その場合は、複数の dbdriver・appServer を用いたり、Think Time をより小さな値にするなどの対策が必要である。尚、開発の過程で、本環境・RedHat において eu=2000 までは実行でき、今回用いたデータベースの性能の限界を超えて測定できたことを付け加えておく。

## 6 DBT-3 による PostgreSQL の評価

### 6.1 概要

DBT-3 は、データを解析し意思決定支援を行うアプリケーションのための、データベースのベンチマーク・ワークロードツールである。

DBT-3 バージョン 1.5 を使用して、PostgreSQL7.4.6 と 8.0.0beta5 の評価を行う。DBT-3 については、「6.1.1 OSDL-DBT-3 の概要」を参照のこと。

今回の評価では、DBT-3 を使用して、以下について評価する。

- ・ DBT-3 ワークロードを使った測定手段を確立する
- ・ DBT-3 を使用して、PostgreSQL7.4 と PostgreSQL8.0 を比較評価する

なお、PostgreSQL のバージョンは、今回の評価を開始した時点での最新バージョンである、7.4.6 と 8.0.0beta5 採用した。

#### 6.1.1 OSDL-DBT-3 の概要

OSDL-DBT-3 (Open Source Development Labs Database Test3) は、LinuxOS やオープンソースソフトウェアのためのデータベーステストキットとして、容易に利用できるように開発された。そのため、より広い開発コミュニティで共有できる事を期待されている。このテストは TPC-H を参考にして単純化したものである。

本節では、TPC-H と DBT-3 についての概要を説明する。  
TPC-H の詳細については、以下のサイトを参照のこと。

<http://www.tpc.org/tpch/>

DBT-3 の詳細については、以下のサイトを参照のこと。

[http://www.osdl.jp/lab\\_activities/kernel\\_testing/osdl\\_database\\_test\\_suite/](http://www.osdl.jp/lab_activities/kernel_testing/osdl_database_test_suite/)  
[http://www.osdl.org/lab\\_activities/kernel\\_testing/osdl\\_database\\_test\\_suite/](http://www.osdl.org/lab_activities/kernel_testing/osdl_database_test_suite/)

#### 6.1.1.1 TPC-H とは

TPC-H ワークロードは、複雑なデータを解析し意思決定支援を行うビジネス活動をシミュレートする。

TPC-H ワークロードは 22 個の意思決定支援を行うクエリーによって行われ、以下の意思決定が導かれる。

- ・ 価格と販売促進
- ・ 需要と供給の管理
- ・ 利益と歳入の管理
- ・ 顧客満足度の調査
- ・ 市場占有率の調査
- ・ 輸送手段の管理

また、TPC-H はデータベースが定期的にアップデートされることを、2 個のリフレッシュ関数を用いてシミュレートする。

TPC-H ワークロードはロードテストとパフォーマンステストにより構成される。

ロードテストはパフォーマンステストを実行するために、データベースの構築、インデックスの作成などが行われる。

パフォーマンステストでは、パワーテストとスループットテストが行われる。

パワーテストではユーザ活動が一人の場合をシミュレートし、連続的にデータベースにデータの追加、22 個の意思決定支援を行うクエリーの問い合わせ、データベースからデータの削除を行う。

スループットテストは、どれだけ多くのクエリーを少ない時間で実行できるかどうかのテストで、スケールファクターによって 2 から 10 の同時接続数で行う。

また、スループットテストは 2 個のリフレッシュ関数と平行して 22 個の問い合わせが行われる。

TPC-H の完全なテストは、ロードテストに加え、パワーテストとスループットテストにより構成されるパフォーマンステストから成り立つ。TPC-H によって報告される性能値は Query-per-Hour(一時間あたりのクエリー実行数)で、パフォーマンステスト実行時間が最も遅い結果によって性能値を計算する。

#### 6.1.1.2 OSDL-DBT-3 とは

OSDL-DBT-3 は TPC-H Revision 1.5.0 を参考にして作成され、SAP DB と PostgreSQL 用に作られているが、できるだけ特定のデータベースに偏ることなく、中立的な立場で開発が行われている。

OSDL DBT3 はオープンソースであり、オープンソースコミュニティで共有できる。

#### 6.1.1.3 注意事項

TPC ベンチマークは性能を比較するツールとして使用されるように意図されているため、OSDL-DBT-3 ワークロードの結果は、いくつかの点において TPC-H ベンチマークの結果と比較することは不可能である。

TPC は厳密な公表に対応できるように全ての結果を公表し、競争者間の公平な比較を保証する監査規則をもっている。

TPC の規則ではさらに全体的な有効性とベンチマークに使用された全ての製品に対して、価格の開示を必要とする。

オープンソース開発プロジェクトがそれらの規則を順守することは現実的ではない。

従って、OSDL-DBT-3 テストによってもたらされた結果は「TPC-H ベンチマーク」にはなりえない。OSDL-DBT-3 ワークロードの結果は、TPC-H ベンチマークとは完全に一致しない。

#### 6.1.1.4 アーキテクチャの概要

DBT-3 は driver とデータベースが同じシステム内におかれた、いわゆる"host-based"構成のもとで、設計されている。

## 6.1.2 テーブルの概要

TPC-H で作成されるテーブルを参考にして、DBT-3 では図 6.1-1 のテーブルを作成する。

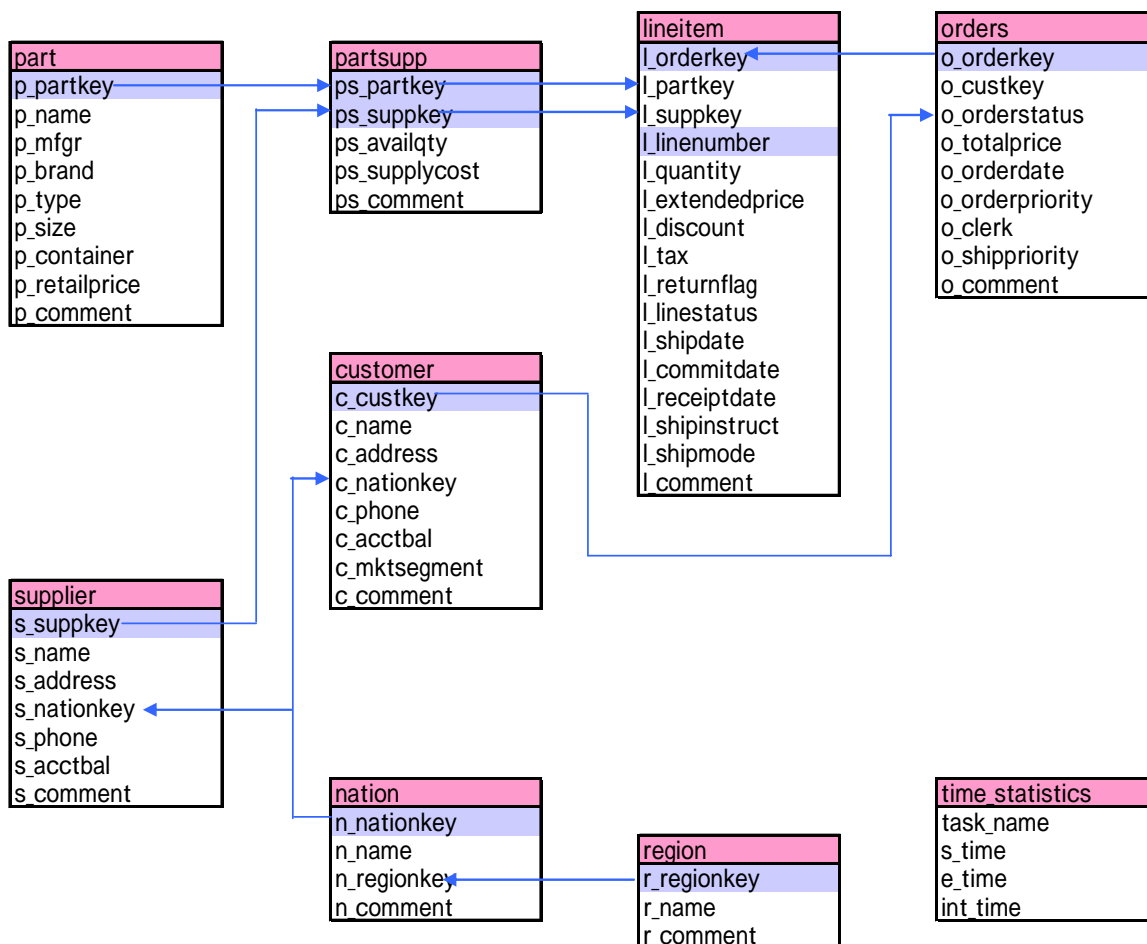


図 6.1-1 テーブル

DBT-3 ワークロードを実行する際に、スケールファクターを指定する。スケールファクターには 1 以上の整数を指定する。各テーブルに投入するデータの行数は、スケールファクターの値を元に、表 6.1-1 の通りに決定される。

表 6.1-1 テーブル一覧

テーブル名	説明	行数
part	部品	スケールファクター × 200000
supplier	発売元	スケールファクター × 10000
partsupp	部品発売元	スケールファクター × 800000

customer	顧客	スケールファクター × 150000
orders	注文	スケールファクター × 1500000
lineitem	明細	スケールファクター × 約 6000000
nation	国	25
region	地域	5
time_statistics	時間統計	0

それぞれのテーブルにおけるカラムの説明を、表 6.1-2 から表 6.1-10 までに示す。

表 6.1-2 part (部品テーブル)

カラム名	データ型	制約	説明
p_partkey	integer	primary key	部品 ID
p_name	varchar(55)		名称
p_mfgr	char(25)		製造元
p_brand	char(10)		ブランド
p_type	varchar(25)		タイプ
p_size	integer		サイズ
p_container	char(10)		コンテナ
p_retailprice	real		小売価格
p_comment	varchar(23)		コメント

表 6.1-3 supplier (発売元) テーブル

カラム名	データ型	制約	説明
s_suppkey	integer	primary key	発売元 ID
s_name	char(25)		名称
s_address	varchar(40)		住所
s_nationkey	integer		国 ID
s_phone	char(15)		電話番号
s_acctbal	real		アカウントバランス
s_comment	varchar(101)		コメント

表 6.1-4 partsupp (部品発売元) テーブル

カラム名	データ型	制約	説明
ps_partkey	integer	primary key	部品 ID
ps_suppkey	integer	primary key	発売元 ID

ps_availqty	integer		利益高
ps_supplycost	real		供給コスト
ps_comment	varchar(199)		コメント

表 6.1-5 customer (顧客) テーブル

カラム名	データ型	制約	説明
c_custkey	integer	primary key	顧客
c_name	varchar(25)		名称
c_address	varchar(40)		住所
c_nationkey	integer		国 ID
c_phone	char(15)		電話番号
c_acctbal	real		アカウントバランス
c_mktsegment	char(10)		マーケットセグメント
c_comment	varchar(117)		コメント

表 6.1-6 orders (注文) テーブル

カラム名	データ型	制約	説明
o_orderkey	integer	primary key	注文 ID
o_custkey	integer		顧客 ID
o_orderstatus	char(1)		注文ステータス
o_totalprice	real		合計金額
o_orderdate	date		注文日付
o_oderpriority	char(15)		注文プライオリティ
o_clerk	char(15)		従業員
o_shippriority	integer		出荷プライオリティ
o_comment	varchar(79)		コメント

表 6.1-7 lineitem (明細) テーブル

カラム名	データ型	制約	説明
l_orderkey	integer	primary key	注文 ID
l_partkey	integer		部品 ID
l_suppkey	integer		発売元 ID
l_linenumber	integer	primary key	明細番号
l_quantity	real		数量

l_extendedprice	real		価格
l_discount	real		割引率
l_tax	real		税金
l_returnflag	char(1)		返品フラグ
l_linestatus	char(1)		明細ステータス
l_shipdate	date		出荷日付
l_commitdate	date		到着日時
l_receiptdate	date		受取日時
l_shipinstruct	char(25)		出荷形態
l_shipmode	char(10)		出荷モード
l_comment	varchar(44)		コメント

表 6.1-8 nation (国) テーブル

カラム名	データ型	制約	説明
n_nationkey	integer	primary key	国 ID
n_name	char(25)		名称
n_regionkey	integer		地域 ID
n_comment	varchar(152)		コメント

表 6.1-9 region (地域) テーブル

カラム名	データ型	制約	説明
r_regionkey	integer	primary key	地域 ID
r_name	char(25)		名称
r_comment	varchar(152)		コメント

表 6.1-10 time\_statistics (時間統計) テーブル

カラム名	データ型	制約	説明
task_name	varchar(40)		タスク名称
s_time	timestamp		開始日時
e_time	timestamp		終了日時
int_time	integer		所要時間

### 6.1.3 クエリーの概要

TPC-H の 22 個の意志決定支援を行うクエリーを参考にして、DBT-3 には表 6.1-11 のクエリーが実装されている。

表 6.1-11 クエリー

番号	クエリー名	内容
Q1	Pricing Summary Report	明細を検索して、請求・出荷・返品のをレポートする。
Q2	Minimum Cost Supplier	ある地域で部品を注文するのに適している発売元を検索する。
Q3	Shipping Priority	ある日付において、受注して出荷前の注文を、収益の高い順で 10 件まで検索する。
Q4	Order Priority Checking	指定日から 3 ヶ月以内において、予定より遅く納品された注文について、注文プライオリティごとの注文数を検索する。
Q5	Local Supplier Volume	指定地域における指定日から 1 年間の、発売元ごとの収益総額を検索する。
Q6	Forecasting Revenue Change	指定の年・割引率・数量における、収益増加の量を予測する。
Q7	Volume Shipping	特定の国間で出荷される商品のボリュームを測定する。
Q8	National Market Share	特定の国における、特定の地域での市場占有率を検索する。
Q9	Product Type Profit Measure	国別、年度別の利益の問い合わせを行う。
Q10	Returned Item Reporting	クレームにより失った収益と顧客リストの問い合わせを行う。
Q11	Important Stock Identification	特定の国における、供給者の在庫維持コストを検索する。
Q12	Shipping Modes and Order Priority	商品の輸送方法により、顧客に商品が届けられるかどうかの問い合わせを行う。
Q13	Customer Distribution	顧客名と受注商品数の関係を検索する。
Q14	Promotion Effect	販売促進による、市場の反応を検索する。
Q15	Top Supplier	三ヶ月間にもっとも収益が多かった発売元の情報と収益を検索する。
Q16	Parts/Supplier Relationship	商品の需要に見合った発売元がいるかどうかを検索する。

Q17	Small-Quantity-Order	ごく少量の商品発注があった場合の集計を検索する。
Q18	Large Volume Customer	もっとも受注量が多かった顧客のリストの問い合わせを行う。
Q19	Discounted Revenue	特定の条件に合致する受注明細の割引率適用価格を集計する。
Q20	Potential Part Promotion	在庫をもっている発売元を検索する。
Q21	Suppliers Who Kept Orders Waiting	決められた期日に商品を出荷できなかった発売元を検索する。
Q22	Global Sales Opportunity	顧客が特定の地域にどれほどいるか検索する。

TPC-H の 2 個のリフレッシュ関数を参考にして、DBT-3 には表 6.1-12 のクエリーが実装されている。

表 6.1-12 リフレッシュ・クエリー

番号	クエリー	内容
RF1	Refresh1	データの追加を行う。
RF2	Refresh2	データの削除を行う。

## 6.2 環境定義

### 6.2.1 システム構成

今回の評価では、PostgreSQL7.4.6 用と 8.0.0beta5 用の、2つのシステムを使用して検証を行った。7.4.6 用のシステムを表 6.2-1 に、8.0.0beta5 用のシステムを表 6.2-2 に示す。

表 6.2-1 PostgreSQL7.4.6 用のシステム

製品名	HP ProLiant DL380 Generation 3		
プロセッサ	インテル Xeon プロセッサ 2.80GHz、2CPU		
メモリ	2.5GByte		
ハードディスク	Ultra SCSI 320 HDD 15000rpm 内蔵ドライブベイに、3台		
ディスク構成	36GB HDD	/boot	1GB
		swap	4GB
		/	4GB
		/usr	4GB
		/var	1GB
		/tmp	1GB
		/home	20GB
	36GB HDD	/db_xlog	すべて
72GB HDD	/dbt3_0	すべて	

すべてのパーティションは ext3 ファイルシステムでフォーマットして、ext3 のジャーナリングのモードも、デフォルトの orderd モードを使用する。  
以下のディレクトリについては、その所有者を postgres ユーザにしておく。  
/db\_xlog、/dbt3\_0

表 6.2-2 PostgreSQL8.0.0beta5 用のシステム

製品名	HP ProLiant DL380 Generation 3		
プロセッサ	インテル Xeon プロセッサ 2.80GHz、2CPU		
メモリ	2.5GByte		
ハードディスク	Ultra SCSI 320 HDD 15000rpm 内蔵ドライブベイに、6台 外部ストレージに、2台		

ディスク構成	36GB HDD	/boot	1GB
		swap	4GB
		/	4GB
		/usr	4GB
		/var	1GB
		/tmp	1GB
		/home	20GB
	36GB HDD	/db_xlog	すべて
	72GB HDD	/dbt3_0	すべて
	36GB HDD	/dbt3_1	すべて
	72GB HDD	/dbt3_2	すべて
	72GB HDD	/dbt3_3	すべて
	72GB HDD	/dbt3_4	すべて
72GB HDD	/dbt3_5	すべて	

すべてのパーティションは ext3 ファイルシステムでフォーマットして、ext3 のジャーナリングのモードも、デフォルトの orderd モードを使用する。

以下のディレクトリについては、その所有者を postgres ユーザにしておく。

/db\_xlog、/dbt3\_0、/dbt3\_1、/dbt3\_2、/dbt3\_3、/dbt3\_4、/dbt3\_4、/dbt3\_5

## 6.2.2 インストール

### 6.2.2.1 Linux のインストール

DBT-3 を実行するためには、Linux のインストール時に、開発環境を含むいくつかのパッケージをインストールする必要がある。

今回の評価では、「MIRACLE LINUX V3.0 - Asianux Inside」を使用した。MIRACLE LINUX のインストール時に選択するパッケージは、「インストールするパッケージを選択」画面で「パッケージのセットをカスタマイズ」を選択し、インストールするパッケージのグループとして、以下を選択した。

- ・ システムツール
- ・ X Window System
- ・ KDE
- ・ Web ブラウザ
- ・ 日本語サポート
- ・ 開発環境

上記の他に必要なパッケージがあり、MIRACLE LINUX インストール CD と Developer CD から、以下のパッケージをインストールした。

- ・ perl-CGI
- ・ procmail
- ・ sysstat
- ・ gnuplot

また、MIRACLE LINUX V3.0 のカーネルは、kernel-2.4.21-9.38AX にアップデートした。

さらに、MIRACLE LINUX に含まれる autoconf パッケージのバージョンは、DBT-3 が使用するバージョンと合わないので、以下のサイトより最新バージョンをダウンロードしてインストールする必要がある。今回の評価を開始した時点の最新バージョンは 2.59 であり、そのバージョンを使用した。

```
http://directory.fsf.org/autoconf.html
```

#### 6.2.2.2 PostgreSQL のインストール

今回の評価では、バージョン 7.4.6 と 8.0.0beta5 を使用した。以下の手順でインストールできるが、二つのバージョンを同時にインストールする場合は、使用するディレクトリが競合しないように、適切に変更する必要がある。

root ユーザで、PostgreSQL の所有者となる Linux のユーザとして、pgsql ユーザを作成する。

```
# useradd pgsql
```

PostgreSQL のソースコードアーカイブを展開するディレクトリと、PostgreSQL をインストールするディレクトリを作成して、ディレクトリの所有者を pgsql ユーザにする。

(a) PostgreSQL7.4.6 の場合

```
# mkdir /usr/local/src/postgresql-7.4.6
# chown pgsql /usr/local/src/postgresql-7.4.6
# mkdir /usr/local/pgsql
# chown pgsql /usr/local/pgsql
```

(b) PostgreSQL8.0.0beta5 の場合

```
# mkdir /usr/local/src/postgresql-8.0.0beta5
# chown pgsq /usr/local/src/postgresql-8.0.0beta5
# mkdir /usr/local/pgsq
# chown pgsq /usr/local/pgsq
```

pgsq ユーザで、PostgreSQL のソースコードアーカイブを展開し、展開したディレクトリに移動する。PostgreSQL のソースコードアーカイブは、/tmp ディレクトリにあるものとする。

(a) PostgreSQL7.4.6 の場合

```
# su - pgsq
$ cd /usr/local/src
$ tar xzf /tmp/postgresql-7.4.6.tar.gz
$ cd postgresql-7.4.6
```

(b) PostgreSQL8.0.0beta5 の場合

```
# su - pgsq
$ cd /usr/local/src
$ tar xzf /tmp/postgresql-8.0.0beta5.tar.gz
$ cd postgresql-8.0.0beta5
```

展開した PostgreSQL のソースコードをコンパイルし、インストールする。

(a) PostgreSQL7.4.6 の場合

```
$ ./configure --prefix=/usr/local/pgsq
$ make all
$ make install
```

(b) PostgreSQL8.0.0beta5 の場合

```
$ ./configure --prefix=/usr/local/pgsq
$ make all
$ make install
```

### 6.2.2.3 データベースクラスタの作成

DBT-3 ワークロードを実行すると、データベースクラスタが既に存在していればそれを使用し、まだ存在していなければ自動的にデータベースクラスタが作成される。しかし、自動的に作成されるデータベースクラスタの文字エンコーディングはASCIIであり、また DBT-3 ワークロードも実行してくれるので、PostgreSQL の設定ファイルを編集することが出来ない。

そこで、DBT-3 ワークロードを実行する前に、手作業でデータベースクラスタを作成することを推奨する。その際に、デフォルトの文字エンコーディングとして、日本で一般的に使用される EUC\_JP を指定する。

データベースクラスタを作成するディレクトリは、デフォルトでは PostgreSQL をインストールしたディレクトリ内であり、通常は OS のシステムファイルがあるハードディスクになる。このディレクトリを、データベースクラスタ専用のハードディスクをマウントしたディレクトリに作成することで、処理性能の向上が見込める。

今回の評価環境では、PostgreSQL をインストールしたディレクトリは空き容量が少ないので、同じディスクの別パーティションをマウントした、/home ディレクトリを使用する。評価環境については、「6.2.1 ハードウェア構成」を参照のこと。

今回の評価では、データベースクラスタのディレクトリとして、OS のシステムファイルがあるハードディスクをマウントした、/home ディレクトリと、データベースクラスタ専用のハードディスクをマウントした、/dbt3\_0 ディレクトリの両方で検証を行った。

- (a) OS のファイルシステムがあるディスクをマウントした、/home ディレクトリに作成する場合

```
$ mkdir /home/pgsql/data
$ /usr/local/pgsql/bin/initdb --encoding=EUC_JP --no-locale ¥
--pgdata=/home/pgsql/data
```

- (b) 専用のディスクをマウントした、/dbt3\_0 ディレクトリに作成する場合

```
$ mkdir /dbt3_0/pgsql
$ /usr/local/pgsql/bin/initdb --encoding=EUC_JP --no-locale ¥
--pgdata=/dbt3_0/pgsql
```

#### 6.2.2.4 DBT-3 のインストール

今回の評価では、DBT-3 のバージョン 1.5 を使用した。以下の手順でインストールすることが出来る。

##### 6.2.2.4.1 前提条件

DBT-3 のソースコードアーカイブを、以下のサイトからダウンロードして、/tmp ディレクトリに格納する。ファイル名は、「dbt3-v1.5.tar.gz」である。

```
http://sourceforge.net/projects/osdl/dbt
```

TPC-H のデータ生成ツールを、以下のサイトの「DBGEN and QGEN」からダウンロードして、/tmp ディレクトリに格納する。ファイル名は「20000511.tar.z」である。

```
http://www.tpc.org/tpch/
```

今回の評価では、DBT-3 の修正パッチを作成した。このパッチを、/tmp ディレクトリに格納する。

DBT-3 のインストールは、pgsql ユーザで行う。DBT-3 をインストールするディレクトリを「/home/pgsql/src/dbt3-v1.5」とし、環境変数「\$DBT3\_HOME」に設定する。

```
$ export DBT3_HOME=/home/pgsql/src/dbt3-v1.5
```

##### 6.2.2.4.2 インストール

- (1) はじめに、pgsql ユーザのホームディレクトリに dbt3\_data ディレクトリと、DBT-3 のソースコードを展開するディレクトリを作成する。dbt3\_data ディレクトリには、テーブル作成時に使用するテキストファイルが保存されることになる。

```
$ mkdir ~/dbt3_data
$ mkdir ~/src
$ cd ~/src
$ tar xzf /tmp/dbt3-v1.5.tar.gz
```

- (2) dbt3-v1.5 修正パッチを、dbt3-v1.5 ディレクトリに対して適用する。

```
$ patch -p0 < dbt3-v1.5.patch
```

- (3) HP DL380 の SCSI コントローラを使う場合は、「`iostat -x`」の出力な余分が改行が入るので、`gnuplot` でグラフ作成ができない。そのような場合は、このパッチを `dbt3-v1.5` ディレクトリに対して適用する。

```
$ patch -p0 < dbt3-v1.5_dl380.patch
```

- (4) TPC-H のデータ生成ツールを `dbt3-v1.5` に取り込む。

```
$ cd ~/src
$ mkdir tpc
$ cd tpc
$ tar xzf /tmp/20000511.tar.z
$ cd $DBT3_HOME/datagen
$ chmod +w dbgen/
$ cp -f ~/src/tpc/appendix/dbgen/* dbgen/
```

- (5) 次のように、PostgreSQL 用のパッチを適用する。

```
$ patch -b -p0 < osdl_dbgen.patch
patching file dbgen/Makefile
The next patch would create the file dbgen/Makefile.in,
which already exists! Assume -R? [n] n
Apply anyway? [n] n
Skippin patch.
1 out of 1 hunk ignored -- saving rejects to file dbgen/Makefile.in.rej
patching file dbgen/bm_utils.c
patching file dbgen/driver.c
patching file dbgen/print.c
patching file dbgen/tpcd.h
```

Makefile.in ファイルにパッチを適用する際に、上記のような警告が表示されるが、既に修正済みの Makefile.in が dbgen に含まれているので問題ない。

- (6) 環境変数の設定を行うために、設定ファイルを書き込み可能に変更する。

```
$ chmod +w $DBT3_HOME/scripts/pgsql/set_run_env.sh.in
```

(7) テキストエディタで、set\_run\_env.sh.in を以下のように設定する。

```
export DSS_QUERY=@TOPDIR@/datagen/@DATABASE_TO_USE@-queries
export DSS_PATH=/home/pgsql/dbt3_data
export DSS_CONFIG=@TOPDIR@/datagen/dbgen
export SID=DBT3
export DBT3_PERL_MODULE=@TOPDIR@/perlmodules
export PATH=/usr/local/pgsql/bin:$PATH
export PGDATA=/dbt3_0/pgsql
export PGUSER=pgsql
```

ここでユーザが変更すべき箇所は DSS\_PATH、PATH、PGDATA である。DSS\_PATH は、テーブル作成時に使用するテキストファイルが保存されるディレクトリ。PATH は、PostgreSQL のコマンドサーチパス。PGDATA は、データベースクラスタの格納場所を指し示す。PGUSER を変更することで、pgsql ユーザ以外でも DBT-3 を実行できそうであるが、DBT-3 のソースコードで pgsql と決め打ちしている箇所があるので、pgsql 以外に変更しても DBT-3 を実行することは出来なかった。

(8) 次の手順で、DBT-3 のインストールを行う。

```
$ autoconf
$ ./configure
$ make
$ make install
```

## 6.2.3 パラメータの設定

### 6.2.3.1 OSパラメータ

#### 6.2.3.1.1 sudo の設定

DBT-3 の実行中にシステムの root 権限が必要なプログラムを実行するので、sudo の設定を行う必要がある。sudo の設定は root ユーザで visudo を起動し、User privilege specification を次の通りに編集する。

```
# User privilege specification
root    ALL=(ALL) ALL
pgsql   ALL=(ALL) NOPASSWD:ALL
```

#### 6.2.3.1.2 grub の設定

DBT-3 ワークロードでは/proc/profile の情報も取得することができる。/proc/profile を作成するために、grub のカーネルパラメータに profile=N(N は 0 以上の数値)と設定することにより、カーネルがどこで CPU サイクルを消費しているか調べることができるようになる。今回の評価では、profile に 1 を設定した。

### 6.2.3.2 PostgreSQL パラメータ

DBT-3 ワークロードでは PostgreSQL の統計情報収集器(statistics collector) の情報を収集することができる。この機能を有効にする場合は、PostgreSQL のデータベースクラスタのディレクトリにある、設定ファイル ( postgresql.conf ) の項目を以下のように設定する必要がある。今回の評価では、以下のように設定した。

```
stats_start_collector = true
stats_command_string = true
stats_block_level = true
stats_row_level = true
stats_reset_on_server_start = true
```

## 6.3 評価手順

### 6.3.1 前提条件

#### 6.3.1.1 環境変数の設定

DBT-3 ワークロードを実行する前に、インストール時に生成された環境設定ファイルをロードして、必要な環境変数を設定する必要がある。

次の手順で、環境変数を設定することが出来る。

```
$ cd $DBT3_HOME/scripts/pgsql
$ source set_run_env.sh
```

#### 6.3.1.2 PostgreSQL7.4 の場合

DBT-3 ver1.5 は、PostgreSQL のバージョン 8.0 を前提としている。PostgreSQL7.4 の最適化では、DBT-3 ver1.5 の一部の SQL に対して適切な実行プランを作ることが出来ない。これにより、その SQL は現実的な時間内に実行を完了することが出来ない。

DBT-3 ver1.4 では、その SQL は PostgreSQL7.4 で問題なく実行できるように、TPC-H で定義されている SQL から書き換えられている。従って、PostgreSQL7.4 で DBT-3 ver1.5 を実行する場合は、該当する SQL を DBT-3 ver1.4 の物に書き換える必要がある。

そのためのパッチファイルを用意してあるので、次の手順でパッチを適用する。

```
$ cd $DBT3_HOME
$ patch -p1 < dbt3-v1.5_query.patch
```

### 6.3.2 評価環境

#### 6.3.2.1 テーブルスペースの設定

PostgreSQL8.0 では、データベースクラスタを複数のハードディスクに割り当てる、テーブルスペース機能を使用することができる。DBT-3 ワークロードで、テーブルスペースを使用するためには、DBT-3 のソースコードの修正が必要である。

DBT-3 ワークロードでは、テーブルとインデックスの作成は、それぞれ \$DBT3\_HOME/scripts/pgsql/create\_table.sql、create\_indexes.sql ファイルで行う。テーブルスペースを使用するには、この2つのファイル修正すれば良い。

例として、lineitem テーブル、order テーブル、lineitem の l\_orderkey カラムのインデックス、order の o\_orderkey プライマリキーに対してテーブルスペースを設定する方法を説明する。

- (1) テーブルスペースを格納するディレクトリとして、表 6.3-1 に示すディレクトリを pgsql ユーザで作成する。また、これらのディレクトリの中身には、ファイルやサブディレクトリを含まない空の状態である必要がある。

表 6.3-1 テーブルスペースのディレクトリ

テーブル、インデックス	ディレクトリ
lineitem テーブル	/dbt3_1/pgtbls
order テーブル	/dbt3_2/pgtbls
lineitem の l_orderkey インデックス	/dbt3_3/pgtbls
order の o_orderkey プライマリキー	/dbt3_4/pgtbls

- (2) create\_table.sql ファイルの先頭に、以下を追加する。

```
create tablespace lineitem_tbls location '/dbt3_1/pgtbls';
create tablespace order_tbls location '/dbt3_2/pgtbls';
create tablespace i_l_orderkey_tbls location '/dbt3_3/pgtbls';
create tablespace order_pkey_tbls location '/dbt3_4/pgtbls';
```

- (3) create\_table.sql ファイルの lineitem テーブル定義を、以下のように変更する。

```
create table lineitem (l_orderkey integer, l_partkey integer, l_suppkey
integer, l_linenummer integer, l_quantity real, l_extendedprice real,
l_discount real, l_tax real, l_returnflag char(1), l_linestatus char(1),
l_shipdate date, l_commitdate date, l_receiptdate date, l_shipinstruct
char(25), l_shipmode char(10), l_comment varchar(44), primary key
( l_orderkey, l_linenummer ) ) tablespace lineitem_tbls;
```

- (4) create\_table.sql ファイルの order テーブル定義を、以下のように変更する。

```
create table orders (o_orderkey integer, o_custkey integer, o_orderstatus
char(1), o_totalprice real, o_orderdate date, o_orderpriority char(15),
o_clerk char(15), o_shippriority integer, o_comment varchar(79), primary
key ( o_orderkey ) using index tablespace order_pkey_tbls ) tablespace
order_tbls;
```

- (5) create\_index.sql ファイルの i\_l\_orderkey インデックス定義を、以下のように変更する。

```
create index i_l_orderkey on lineitem (l_orderkey)
    tablespace i_l_orderkey_tbls;
```

以上の設定で、PostgreSQL8.0 のテーブルスペース機能を、DBT-3 ワークロードで使うことができる。

### 6.3.2.2 DBT-3 ワークロードの実行

DBT-3 ワークロードは、次の 4 つの処理を順番に実行する。

(1) データの生成

DBT-3 ワークロードで使用する、データベースに投入するためのテキストファイルを生成する。

(2) ロードテスト

生成したテキストファイルをデータベースに投入し、その処理時間を測定する。

(3) パワーテスト

DBT-3 ワークロードで測定する SQL 文を 1 つずつ実行し、その処理時間を測定する。

(4) スループットテスト

DBT-3 ワークロードで測定する SQL 文を、指定のストリーム数で同時に実行し、その処理時間を測定する。

DBT-3 ワークロードを実行するには、\$DBT3\_HOME/scripts/run\_workload.sh スクリプトファイルを使用する。このスクリプトファイルは、DBT-3 ワークロードを実行し、その測定結果を特定のディレクトリに保存する。

そのディレクトリは、\$DBT3\_HOME/scripts/run\_number ファイルに数値で指定する。省略時のデフォルト値は 0 である。例えば、1 を指定するには、次の通りである。

```
$ echo 1 > $DBT3_HOME/scripts/run_number
```

上記のように 1 を指定すると、DBT-3 ワークロードを実行した結果は、\$DBT3\_HOME/scripts/output/1 ディレクトリに作成される。

\$DBT3\_HOME/scripts/run\_workload.sh スクリプトファイルには、次のオプションが指定できる。

```
run_workload.sh
-f <scale_factor>
-n <num_stream>
-r <redirect_tmp>
-x <redirect_xlog>
-p <db_param>
-s <seed>
-o <USE_OPROFILE>
```

- -f <scale\_factor>  
DBT-3 ワークロードでは、データベースの規模をスケールファクターで指定する。スケールファクターの値は、データベースに投入するテキストファイルのサイズと対応し、1 以上を指定する必要がある。スケールファクターが 1 であれば、生成されるテキストファイルのサイズは 1G バイトであり、データベースに投入した場合のデータベースクラスタのサイズは 4G バイト以上となる。テキストファイルは、\$DSS\_PATH のディレクトリに生成され、スケールファクターのデフォルト値は 1 である。
- -n <num\_stream>  
スループットテストで同時に実行するトランザクション数を指定する。表 6.3-2 の通りに、scale\_factor ごとの num\_stream の値の下限が決められている。

表 6.3-2 スケールファクターごとのストリーム数の下限

scale_factor	num_stream
1	2
10	3
30	4
100	5
300	6
1000	7
3000	8
10000	10

- `-r <redirect_tmp>`  
DBT-3 ワークロードで使用される SQL 文のシンボリックリンクが格納されるディレクトリを指定する。redirect\_tmp には、0 または 1 を指定する。0 を指定した場合には /tmp が使用され、1 を指定した場合には /db\_tmp が使用される。/db\_tmp を使用する場合は、あらかじめディレクトリを作成し、所有者を postgresql ユーザにする必要がある。redirect\_tmp の省略時のデフォルト値は、0 である。
- `-x <redirect_xlog>`  
PostgreSQL のトランザクションログ (WAL ログ) を格納するディレクトリを指定する。redirect\_xlog には、0 または 1 を指定する。0 を指定した場合は、PostgreSQL のインストールディレクトリ配下の \$PGDATA/pg\_xlog ディレクトリが使用される。1 を指定した場合は、/db\_xlog ディレクトリが使用される。/db\_xlog を使用する場合は、あらかじめディレクトリを作成し、所有者を postgresql ユーザにする必要がある。redirect\_xlog の省略時のデフォルト値は、0 である。
- `-p <db_param>`  
postmaster に渡すオプションを指定する。下記のように複数のオプションを渡すことができる。

```
-p "-B 10000 -c sort_mem=262144"
```

- `-s <seed>`  
DBT-3 で実行する SQL 文のパラメータは乱数で生成する。その乱数の種を、seed で指定できる。seed の省略時のデフォルト値は、現在の時刻である。
- `-o <USE_OPROFILE>`  
oprofile のデータを取得するかを指定する。USE\_OPROFILE には、0 または 1 を指定する。0 を指定した場合は取得せず、1 を指定した場合に取得する。USE\_OPROFILE の省略時のデフォルト値は 0 である。なお、oprofile はバージョン 0.6 以上をサポートする。

DBT-3 ワークロードを一括して実行するには、以下のように実行する。この例では、スケールファクターとして 1 を、ストリーム数として 4 を指定している。

```
$ run_workload.sh -f 1 -n 4 > dbt3.out 2>&1
```

DBT-3 ワークロード実行時の出力は、測定結果のレポートを作成するときに必要なので、ファイル名を dbt3.out として保存しておく必要がある。

### 6.3.3 レポートの作成

DBT-3 ワークロードの測定結果を元に、HTML 形式のレポートを作成することができる。

#### 6.3.3.1 レポートの作成方法

DBT-3 ワークロードの測定結果を元に、HTML 形式のレポートを作成するには、次の通りに実行する。

- (1) DBT-3 ワークロードの出力ファイルを、実行した結果の出力先ディレクトリに移動する。以下は、\$DBT3\_HOME/scripts/run\_number に 1 を指定した場合の例である。

```
$ cd $DBT3_HOME/scripts
$ mv dbt3.out output/1
```

- (2) \$DBT3\_HOME/data\_collect/resulttools/wb\_dbt3\_report\_pgsql.pl スクリプトファイルを実行し、HTML 形式のレポートを作成する。また、レポートの説明ファイルをコピーする。

```
$ cd $DBT3_HOME/data_collect/resulttools
$ ./wb_dbt3_report_pgsql.pl ¥
--indir=../../scripts/output/1 ¥
--outfile=../../scripts/output/1/report.html
$ cp dbt3_explain.html ../../scripts/output/1
```

- (3) PostgreSQL のログファイルを、レポートの作成先ディレクトリにコピーして、読み込みパーミッションを与える。

```
$ cd $DBT3_HOME/run
$ cp db_logfile.txt ../../scripts/output/1
$ chmod 644 ../../scripts/output/1/db_logfile.txt
```

以上の手順で、DBT-3 ワークロードの測定結果から、HTML 形式のレポートファイルを作成できる。作成したレポートファイルを閲覧するには、Web ブラウザで report.html ファイルを開く。

### 6.3.3.2 追加レポートの作成方法

DBT-3 に付属するマニュアルには記述されていないが、DBT-3 には PostgreSQL の統計情報を元に、グラフ出力する機能が含まれている。統計情報をグラフ出力するには、次の通りに実行する。

- (1) `$DBT3_HOME/data_collect/analyzertools/pgsql/analyze_stats.pl` スクリプトファイルを使用して、PostgreSQL の統計情報を出力したファイルを元に、gnuplot でグラフを描けるように整形する。パワーテスト時の統計情報を処理するには、次の通りに実行する。

```
$ cd $DBT3_HOME/data_collect/analyzertools/pgsql
$ ./analyze_stats.pl ¥
--phase=power1 ¥
--directory=$DBT3_HOME/scripts/output/1/db_stat
```

- (2) スループット時の統計情報を処理するには、次の通りに実行する。

```
$ cd $DBT3_HOME/data_collect/analyzertools/pgsql
$ ./analyze_stats.pl ¥
--phase=throughput1 ¥
--directory=$DBT3_HOME/scripts/output/1/db_stat
```

- (3) 整形された情報を元に、パワーテスト時のインデックス情報、インデックススキャン情報、テーブル情報のグラフを描くには、次の通りに実行する。

```
$ cd $DBT3_HOME/scripts/output/1/db_stat
$ gnuplot $DBT3_HOME/data_collect/analyzertools/pgsql/power.index_info.input
$ gnuplot $DBT3_HOME/data_collect/analyzertools/pgsql/power.index_scan.input
$ gnuplot $DBT3_HOME/data_collect/analyzertools/pgsql/power.table_info.input
```

- (4) スループット時のグラフを描くには、次の通りに実行する。

```
$ cd $DBT3_HOME/scripts/output/1/db_stat
$ gnuplot $DBT3_HOME/data_collect/analyzertools/pgsql/thruput.index_info.input
$ gnuplot $DBT3_HOME/data_collect/analyzertools/pgsql/thruput.index_scan.input
$ gnuplot $DBT3_HOME/data_collect/analyzertools/pgsql/thruput.table_info.input
```

(5) Web ブラウザで、作成したグラフを表示するための HTML ファイルをコピーする。

```
$ cd $DBT3_HOME/examples/pgsql
$ cp * $DBT3_HOME/scripts/output/1/db_stat
```

以上の手順で、DBT-3 ワークロードの実行時の PostgreSQL の統計情報から、HTML 形式のレポートファイルを作成できる。作成したレポートファイルを閲覧するには、Web ブラウザで power\_db\_stat.html、throuput\_db\_stat.html ファイルを開く。

### 6.3.4 レポートの内容

Web ブラウザで report.html ファイルを開くと、図 6.3-1 の画面が表示される。

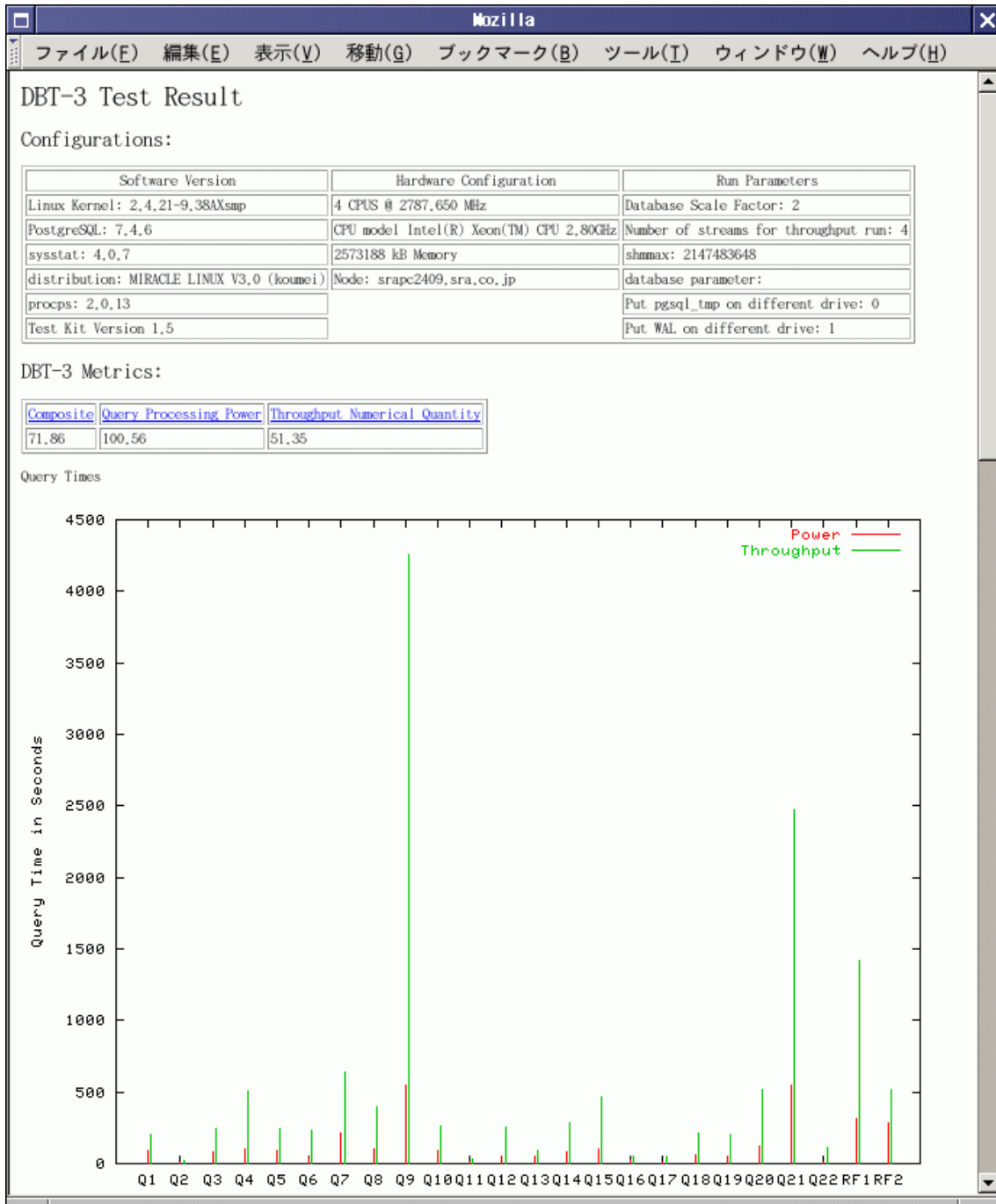


図 6.3-1 レポート

この画面に表示される各項目の詳細を、以下に説明する。

### 6.3.4.1.1 Configurations

この項目には、図 6.3-2 のように Software Version、Hardware Configurations、Run Parameters が表示される。

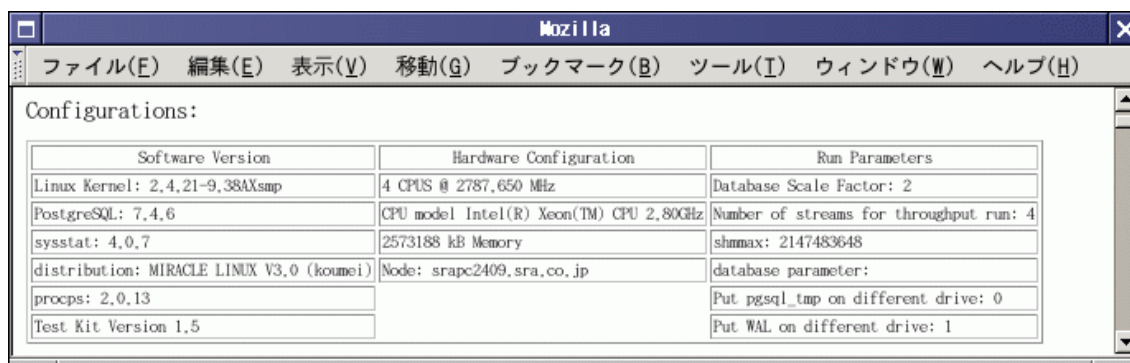


図 6.3-2 Configurations

それぞれの項目の説明を、表 6.3-3 に示す。

表 6.3-3 Configurations の項目

Software Version	OS の情報、PostgreSQL のバージョン、DBT-3 のバージョンが記載される。	
Hardware Configuration	ハードウェアの情報、ホスト名が記載される。	
Run Parameters	以下の情報が記載される。	
	DataBase Scale Factor	スケールファクター
	Number of streams for throughput run	スループットテストで、同時に SQL 文を実行するストリーム数
	shmmax	共有メモリサイズ
	database parameter	-p オプションで指定した内容
	Put postgresql_tmp on different drive	-r オプションで指定した内容
	Put WAL on different drive	-x オプションで指定した内容

#### 6.3.4.1.2 DBT-3 Metrics

この項目には、図 6.3-3 のように Composite、Query Processing Power、Through put Numerical Quantity が表示される。



Composite	Query Processing Power	Throughput Numerical Quantity
71.86	100.56	51.35

図 6.3-3 DBT-3 Metrics

それぞれの項目の説明を、表 6.3-4 に示す。

表 6.3-4 DBT-3 Metrics の項目

Composite	Query Processing Power と Throughput Numerical Quantity の相乗平均である。この数値が高ければ高いほど性能が良いといえる。
Query Processing Power	パワーテストの性能結果であり、一時間あたりに実行できるクエリー数に、スケールファクターを掛け合わせた数値である。スケールファクターを掛けるのは、TPC-H で規定されているからであり、規模を大きくしても結果の数値は一定の値となるように計算される。数値が大きければ大きいほど性能が良いといえる。
Throughput Numerical Quantity	スループットテストの性能結果であり、一時間あたりに実行できるクエリー数に、スケールファクターを掛け合わせた数値である。スケールファクターを掛けるのは、TPC-H で規定されているからであり、規模を大きくしても結果の数値は一定の値となるように計算される。数値が大きければ大きいほど性能が良いといえる。

### 6.3.4.1.3 Query Times

図 6.3-4 は、DBT-3 ワークロードで実行した、22 + 2 種類のクエリーの実行時間をグラフで描いたものである。

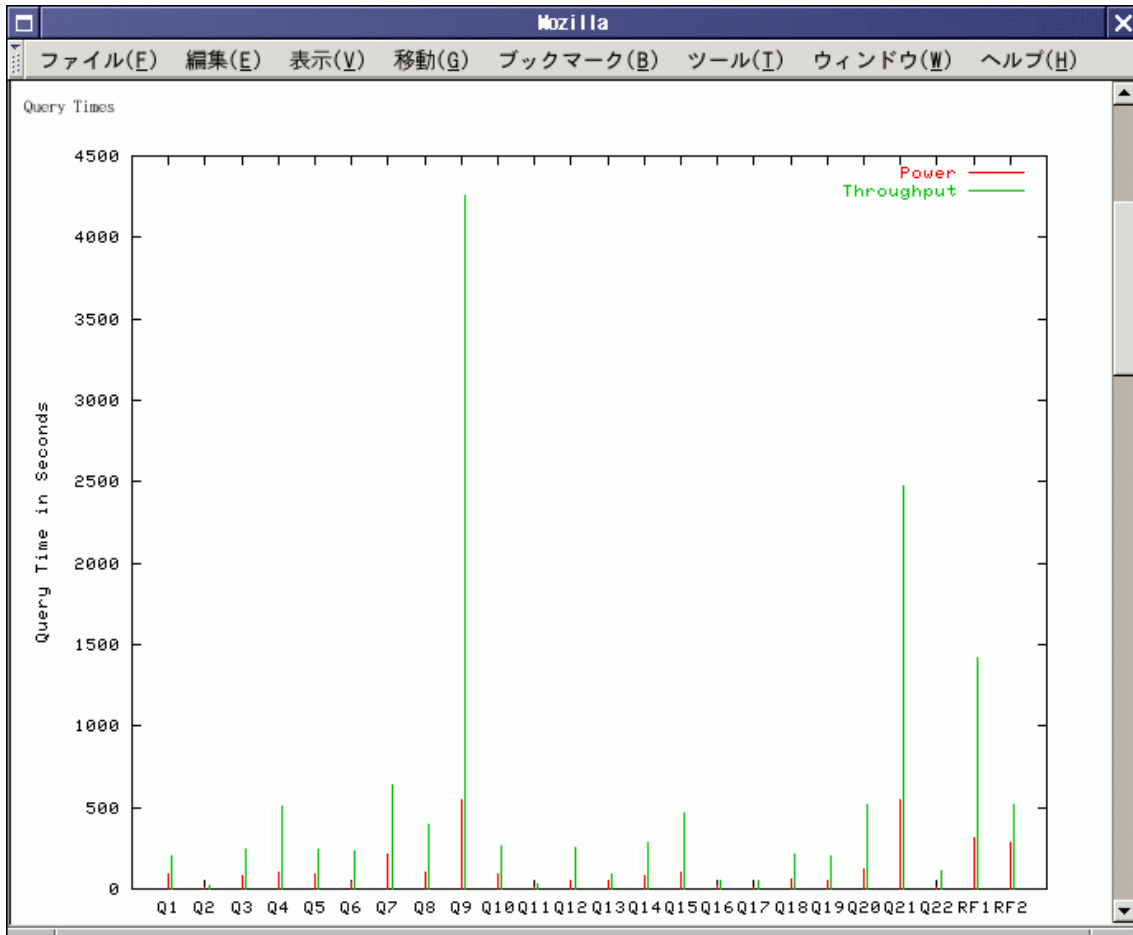


図 6.3-4 Query Times

それぞれの項目の説明を、表 6.3-5 に示す。

表 6.3-5 Query Times の項目

縦軸	実行にかかった秒数。
横軸	クエリーの番号。
赤色のグラフ	パワーテストの、各クエリーごとの実行秒数。パワーテストは、ストリーム数を 1 として、DBT-3 ワークロードを実行する。
緑色のグラフ	スループットテストの、各クエリーごとの実行秒数。スループットテストは、ストリーム数を指定の値で、DBT-3 ワークロードを実行する。指定した値は、Configurations の Number of streams for throughput run に記載される。

### 6.3.4.1.4 Task Execution Time

図 6.3-5 には、各テストの開始時間、終了時間、実行時間が記載される。

Task	Start Time	End Time	Elapsed Time
LOAD	2004-11-26 16:44:37	2004-11-26 17:25:19	00:40:42
PERF1.POWER.RF1	2004-11-26 17:25:35, 242915	2004-11-26 17:30:44, 764802	00:05:09, 521887
PERF1.POWER.QS	2004-11-26 17:30:44, 835889	2004-11-26 18:12:24, 093971	00:41:39, 258082
PERF1.POWER.RF2	2004-11-26 18:12:24, 347913	2004-11-26 18:17:09, 176649	00:04:44, 828736
PERF1.POWER	2004-11-26 17:25:35, 203696	2004-11-26 18:17:09, 21413	00:51:34, 010434
PERF1.THRUPUT.RFST1	2004-11-26 18:17:23, 031733	2004-11-26 18:46:19, 569076	00:28:56, 537343
PERF1.THRUPUT.RFST2	2004-11-26 18:46:19, 605441	2004-11-26 19:18:27, 180269	00:32:07, 574828
PERF1.THRUPUT.RFST3	2004-11-26 19:18:27, 227744	2004-11-26 19:53:44, 565944	00:35:17, 3382
PERF1.THRUPUT.RFST4	2004-11-26 19:53:44, 609955	2004-11-26 20:25:58, 360504	00:32:13, 750549
PERF1.THRUPUT.QS4.ALL	2004-11-26 18:17:25, 022524	2004-11-26 21:20:46, 375394	03:03:21, 35287
PERF1.THRUPUT.QS1.ALL	2004-11-26 18:17:25, 018205	2004-11-26 21:29:28, 23104	03:12:03, 212835
PERF1.THRUPUT.QS3.ALL	2004-11-26 18:17:25, 011893	2004-11-26 21:39:06, 281581	03:21:41, 269688
PERF1.THRUPUT.QS2.ALL	2004-11-26 18:17:24, 954059	2004-11-26 21:43:00, 768338	03:25:35, 814279
PERF1.THRUPUT	2004-11-26 18:17:22, 960767	2004-11-26 21:43:01, 181127	03:25:38, 22036
PERF1	2004-11-26 17:25:20, 935568	2004-11-26 21:43:14, 777347	04:17:53, 841779

図 6.3-5 Task Execution Time

それぞれの項目の説明を、表 6.3-6 に示す。

表 6.3-6 Task Execution Time の項目

LOAD	ロードテストの時間。
PREF1.POWER.RF1	パワーテストの、リフレッシュクエリー1の実行時間。
PREF1.POWER.QS	パワーテストの、22個のクエリーの実行時間。
PREF1.POWER.RF2	パワーテストの、リフレッシュクエリー2の実行時間。
PREF1.POWER	パワーテスト全体の実行時間。
PREF1.THRUPUT.QS[1 ~ n].ALL	スループットテストの、22のクエリーのストリームごとの実行時間。
PREF1.THRUPUT.RFST[1 ~ n]	スループットテストの、2個のリフレッシュクエリーのストリームごとの実行時間。
PREF1.THRUPUT	スループットテスト全体の実行時間。
PREF1	パワーテストとスループットテストを合わせた実行時間。

### 6.3.4.1.5 Raw data

図 6.3-6 には、DBT-3 ワークロードの実行中に採取した sar、vmstat、iostat、readprofile のログファイルへのリンクが表示される。



図 6.3-6 Raw data

### 6.3.4.1.6 gnuplot charts

図 6.3-7 には、「図 6.3-6 Row data」のログファイルを、gnuplot でグラフ表示した画像ファイルへのリンクが表示される。



図 6.3-7 gnuplot charts

### 6.3.4.1.7 Power and Throughput Test Query Results

図 6.3-8 には、DBT-3 ワークロード実行中に採取した、ログファイルへのリンクが表示される。



図 6.3-8 Power and Throughput Test Query Results

それぞれの項目の説明を、表 6.3-7 に示す。

表 6.3-7 Power and Throughput Test Query Results の項目

dbt3.out	run_workload.sh ( DBT-3 ワークロード ) が出力したログファイル
q_time.out	各クエリーごとの実行時間
power_query.result	パワーテストで実行した 22 個のクエリーのログ
power_perf1.rf1.result	パワーテストで実行したリフレッシュクエリー-1 のログ
power_perf1.rf2.result	パワーテストで実行したリフレッシュクエリー-2 のログ
thruput_qs[1 ~ n].result	スループットテストで実行した 22 個のクエリーのログ
thruput.perf1.stream[1 ~ n].rf1.result	スループットテストで実行したリフレッシュクエリー-1 のログ
thruput.perf1.stream[1 ~ n].rf2.result	スループットテストで実行したリフレッシュクエリー-2 のログ

### 6.3.4.1.8 Database Monitor Data

図 6.3-9 の「Database Monitor Data」には、データベースに関するデータへのリンクが表示される。



図 6.3-9 Database Monitor Data、Other data

それぞれの項目の説明を、表 6.3-8 に示す。

表 6.3-8 Database Monitor Data の項目

database parameters	postgresql.conf に、-p オプションで指定したオプションを反映したデータベースのパラメータ
database indexes and primary keys	index とプライマリキーの情報
database monitor files	PostgreSQL の統計情報
database log files	PostgreSQL が出力するログ

### 6.3.4.1.9 Other data

図 6.3-9 の「Other data」には、その他のデータへのリンクが表示される。それぞれの項目の説明を、表 6.3-9 に示す。

表 6.3-9 Other data の項目

ipcs data	ipcs のログ
-----------	----------

## 6.4 測定結果と考察

DBT-3 ワークロードを、いくつかのパターンで実行した結果と、その考察を報告する。

PostgreSQL のバージョンは、今回の評価を開始した時点での最新バージョンである 7.4.6 と 8.0.0beta5 を使用した。また、今回の評価を終了する直前にリリースされた、8.0.1 正式版についても、DBT-3 ワークロードの基本的なパターンで実行することで、8.0.0beta5 と比較した評価も報告する。

### 6.4.1 性能に影響を与えうるファクター

DBT-3 ワークロードは、次の要因により、実行した結果に影響を及ぼすと考えられる。

- ・ PostgreSQL バージョン (7.4.6 と 8.0.0beta5)
- ・ ディスク構成 (3 台、2 台、1 台)
- ・ スケールファクター (データベースの規模)
- ・ ストリーム数 (同時に実行するトランザクション数)
- ・ チューニング

#### 6.4.1.1 PostgreSQL バージョン

バージョン 8.0.0beta5 は、7.4.6 と比べると、機能追加以外にもオプティマイザの改良といった性能向上に貢献する改良が施されている。従って、DBT-3 ワークロードの測定結果についても、バージョン 8.0.0beta5 の方が良い結果となることが予想される。

また、8.0.0beta5 と 8.0.1 とでは主な変更点はバグフィックスなので、性能の違いはほとんど無いと予想される。

#### 6.4.1.2 ディスク構成

ディスク構成は、PostgreSQL の動作速度に影響を与える要素である。Linux のシステムファイル群を置くシステム領域、PostgreSQL のデータベースクラスタを置く領域、PostgreSQL の WAL を置く領域を、それぞれ別のディスクに配置すると効率の良いディスクアクセスが行われる。

今回の評価では、3 台のハードディスクを使い、次の 3 種類のディスク構成についてベンチマークを実施した。DBT-3 ワークロードについても、3 台構成の性能が良いのか、またどの程度の性能差があるのかを評価した。

- ・ 3 台構成  
OS のシステム領域、PostgreSQL のデータベースクラスタ、PostgreSQL の WAL を、それぞれ別のディスクに割り当てる。ディスクアクセスが分散するので、

3種類のディスク構成の中で、DBT-3 ワークロードの測定結果が一番良くなるものと予想される。

ディスク 1	OS のシステム領域
ディスク 2	PostgreSQL のデータベースクラスタ
ディスク 3	PostgreSQL の WAL

- 2 台構成

OS のシステム領域に 1 台のディスクを割り当てる。別のディスクに、PostgreSQL のデータベースクラスタと WAL の両方を割り当てる。OS のシステム領域へのアクセスは専用のディスクに行われるので、性能劣化の要因は無い。しかし、それよりもアクセス頻度が高い、PostgreSQL のデータベースクラスタと WAL へのアクセスが、同一ディスクに競合して行われる。従って、3 台構成のディスク構成よりも、ディスクアクセスの効率は低下するので、DBT-3 ワークロードの測定結果が 2 番目になるものと予想される。

ディスク 1	OS のシステム領域
ディスク 2	PostgreSQL のデータベースクラスタ、WAL

- 1 台構成

OS のシステム領域、PostgreSQL のデータベースクラスタ、PostgreSQL の WAL を、同一ディスクに配置する。ディスクアクセスが、一つのハードディスクに集中するので、3 種類のディスク構成の中で、DBT-3 ワークロードの測定結果が一番悪くなるものと考えられる。しかし、OS のシステム領域への書き込みは、それほど頻繁には発生しないので、2 台構成との性能差はほとんど無いと予想される。

ディスク 1	OS のシステム領域、 PostgreSQL のデータベースクラスタ、WAL
--------	---

なお、データベースに投入するテキストデータのファイルは、OS のシステム領域に置いて、DBT-3 ワークロードを実行した。

#### 6.4.1.3 スケールファクター

スケールファクターは、DBT-3 ワークロードで使用するデータベースに投入するテキストデータのサイズで、単位は GB (ギガバイト) である。この値を大きくすることで、データベースの規模が大きくなるので、DBT-3 ワークロードで実行する SQL の実行時間が長くなる。

DBT-3 ワークロードにおける、データベースのサイズと SQL の実行時間との関係を測定することは、PostgreSQL で意志決定支援を行うような大規模で複雑な処理が、どこまで実用に耐えられるのかを探る上で重要であると考えられる。

今回の評価では、スケールファクターとして 1、2、4、6、10 を採用して、DBT-3 ワークロードを実行した。

#### 6.4.1.4 ストリーム数

ストリーム数は、同時に DBT-3 ワークロードの SQL を実行するトランザクション数である。この値を大きくすることで、同時に SQL を実行するトランザクション数が増えるので、DBT-3 ワークロードで実行する SQL の実行時間が長くなる。しかし、DBT-3 ワークロードは、意志決定を支援するための情報を検索するのが目的であるので、同時に複数の SQL を実行することに、それほど意味は無いと考えられる。

DBT-3 ワークロードでは、ストリーム数が 1 のパワーテストと、ストリーム数を指定するスループットテストの両方を実行する。スループットテストのストリーム数の最小値は、スケールファクターの大きさに合わせて表 6.4-1 の通りに決まっている。

表 6.4-1 スケールファクターごとのストリーム数の下限

スケールファクター	ストリーム数
1	2
10	3
30	4
100	5
300	6
1000	7
3000	8
10000	10

今回の評価では、スケールファクターを 10 まで実施するので、ストリーム数として必要な最小値は 3 となるが、パワーテスト（ストリーム数は 1）と比較しやすいように、4 倍の数値の 4 をストリーム数として採用して、DBT-3 ワークロードを実施した。

#### 6.4.1.5 チューニング

DBT-3 ワークロードで実行するトランザクションの特性を解析して、PostgreSQL の設定や、実行する SQL 文などにチューニングを行った。また、PostgreSQL8.0 のテーブルスペースについても評価した。

DBT-3 ワークロードにチューニングの余地がある場合に、測定結果が良くなることが予想される。

## 6.4.2 測定結果と考察

PostgreSQL バージョン 7.4.6 と 8.0.0beta5 について、DBT-3 ワークロードを次の観点から実行し、その結果を評価して考察を記述した。

- ・ DBT-3 ワークロードの動作確認
- ・ ディスク構成による違い
- ・ スケールファクターによる違い
- ・ チューニングによる違い

また、今回の評価期間が終了する直前に、PostgreSQL8.0.1 正式版がリリースされた。代表的なパターンで 8.0.1 正式版についても評価して、8.0.0beta5 のと評価結果の比較も行った。

### 6.4.2.1 DBT-3 ワークロードの動作確認

#### 6.4.2.1.1 目的

DBT-3 ワークロードが、PostgreSQL7.4.6 と 8.0.0beta5 で問題なく動作することを確認する。

#### 6.4.2.1.2 結果

DBT-3 バージョン 1.5 で実行する SQL 文の中には、PostgreSQL7.4.6 では現実的な時間で実行を終了できない SELECT 文が含まれていた。それは、22 個ある SQL 文の一つ、Q19 の SELECT 文であり、その内容をリスト 6.4-1 に示す。

リスト 6.4-1 Q19 の SELECT 文

```
select sum(l_extendedprice* (1 - l_discount)) as revenue
from lineitem, part
where (
    p_partkey = l_partkey
    and p_brand = ':1'
    and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
    and l_quantity >= :4 and l_quantity <= :4+10
    and p_size between 1 and 5
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
) or (
    p_partkey = l_partkey
```

```

and p_brand = ':2'
and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
and l_quantity >= :5 and l_quantity <= :5+10
and p_size between 1 and 10
and l_shipmode in ('AIR', 'AIR REG')
and l_shipinstruct = 'DELIVER IN PERSON'
) or (
p_partkey = l_partkey and p_brand = ':3'
and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
and l_quantity >= :6 and l_quantity <= :6+10
and p_size between 1 and 15
and l_shipmode in ('AIR', 'AIR REG')
and l_shipinstruct = 'DELIVER IN PERSON'
);

```

この SELECT 文は、PostgreSQL7.4.6 ではリスト 6.4-2 に示す実行プランが作成された。cost の数値の異常な大きさから、実用的な時間で実行を完了することは出来ないことが分かる。実際には、3 日かけても終了しなかったので、強制中断せざるを得なかった。

実行プランについては、PostgreSQL のマニュアル「第 13 章 性能に関するヒント」を参照のこと。

<http://www.postgresql.jp/document/pg746doc/html/performance-tips.html>

#### リスト 6.4-2 PostgreSQL7.4.6 での実行プラン

```

Aggregate (cost=680256343229.03..680256343229.04 rows=1 width=8)
-> Nested Loop (cost=8222.00..680256343228.95 rows=33 width=8)
      Join Filter: (((("outer".l_shipmode = 'AIR'::bpchar) AND
("inner".p_container = 'SM CASE'::bpchar) AND ("inner".p_partkey =
"outer".l_partkey) AND ("inner".p_brand = 'Brand#51'::bpchar) AND
("outer".l_quantity >= 2::double precision) AND ("outer".l_quantity <= 12::double
precision) AND ("inner".p_size >= 1) AND ("inner".p_size <= 5) AND
("outer".l_shipinstruct = 'DELIVER IN PERSON'::bpchar)) OR (("outer".l_shipmode =
'AIR REG'::bpchar) AND ("inner".p_container = 'SM CASE'::bpchar) AND
("inner".p_partkey = "outer".l_partkey) AND ("inner".p_brand = 'Brand#51'::bpchar)
AND ("outer".l_quantity >= 2::double precision) AND ("outer".l_quantity <=
12::double precision) AND ("inner".p_size >= 1) AND ("inner".p_size <= 5) AND
("outer".l_shipinstruct = 'DELIVER IN PERSON'::bpchar)) OR

```

．．．中略．．．

```
((“outer”.l_shipmode = 'AIR'::bpchar) AND (“inner”.p_container = 'LG PKG'::bpchar)
AND (“inner”.p_partkey = “outer”.l_partkey) AND (“inner”.p_brand =
'Brand#45'::bpchar) AND (“outer”.l_quantity >= 21::double precision) AND
(“outer”.l_quantity <= 31::double precision) AND (“inner”.p_size >= 1) AND
(“inner”.p_size <= 15) AND (“outer”.l_shipinstruct = 'DELIVER IN PERSON'::bpchar))
OR ((“outer”.l_shipmode = 'AIR REG'::bpchar) AND (“inner”.p_container = 'LG
PKG'::bpchar) AND (“inner”.p_partkey = “outer”.l_partkey) AND (“inner”.p_brand =
'Brand#45'::bpchar) AND (“outer”.l_quantity >= 21::double precision) AND
(“outer”.l_quantity <= 31::double precision) AND (“inner”.p_size >= 1) AND
(“inner”.p_size <= 15) AND (“outer”.l_shipinstruct = 'DELIVER IN PERSON'::bpchar))
-> Seq Scan on lineitem (cost=0.00..187831.95 rows=5995295 width=59)
-> Materialize (cost=8222.00..11687.00 rows=200000 width=36)
-> Seq Scan on part (cost=0.00..6757.00 rows=200000 width=36)
```

そこで、この SELECT 文をリスト 6.4-3 に示す通りに変更した。この変更は、重複している検索条件をまとめることで、cost を下げることが目的としている。検索結果が変わるような変更では無い。

### リスト 6.4-3 変更した SELECT 文

```
select sum(l_extendedprice* (1 - l_discount)) as revenue
from lineitem, part
where
    p_partkey = l_partkey
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
    and
(
    (
        p_brand = ':1'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= :4 and l_quantity <= :4+10
        and p_size between 1 and 5
    ) or (
        p_brand = ':2'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
```

```

        and l_quantity >= :5 and l_quantity <= :5+10
        and p_size between 1 and 10
    ) or (
        p_brand = ' :3'
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= :6 and l_quantity <= :6+10
        and p_size between 1 and 15
    )
);

```

この SELECT 文は、PostgreSQL7.4.6 ではリスト 6.4-4 に示す実行プランが作成された。

#### リスト 6.4-4 変更後の実行プラン

```

Aggregate (cost=271799.95..271799.95 rows=1 width=8)
-> Merge Join (cost=257694.05..271799.87 rows=32 width=8)
    Merge Cond: ("outer".p_partkey = "inner".l_partkey)
    Join Filter: (((("outer".p_brand = 'Brand#51'::bpchar) AND
("outer".p_container = 'SM CASE'::bpchar) OR ("outer".p_container = 'SM
BOX'::bpchar) OR ("outer".p_container = 'SM PACK'::bpchar) OR ("outer".p_container
= 'SM PKG'::bpchar)) AND ("inner".l_quantity >= 2::double precision) AND
("inner".l_quantity <= 12::double precision) AND ("outer".p_size >= 1) AND
("outer".p_size <= 5)) OR (((("outer".p_brand = 'Brand#21'::bpchar) AND
(("outer".p_container = 'MED BAG'::bpchar) OR ("outer".p_container = 'MED
BOX'::bpchar) OR ("outer".p_container = 'MED PKG'::bpchar) OR ("outer".p_container
= 'MED PACK'::bpchar)) AND ("inner".l_quantity >= 19::double precision) AND
("inner".l_quantity <= 29::double precision) AND ("outer".p_size >= 1) AND
("outer".p_size <= 10)) OR (((("outer".p_brand = 'Brand#45'::bpchar) AND
(("outer".p_container = 'LG CASE'::bpchar) OR ("outer".p_container = 'LG
BOX'::bpchar) OR ("outer".p_container = 'LG PACK'::bpchar) OR ("outer".p_container
= 'LG PKG'::bpchar)) AND ("inner".l_quantity >= 21::double precision) AND
("inner".l_quantity <= 31::double precision) AND ("outer".p_size >= 1) AND
("outer".p_size <= 15)))
    -> Index Scan using part_pkey on part (cost=0.00..7569.00 rows=200000
width=36)
    -> Sort (cost=257694.05..258285.50 rows=236583 width=16)
        Sort Key: lineitem.l_partkey
        -> Seq Scan on lineitem (cost=0.00..232796.66 rows=236583

```

```
width=16)
```

```
Filter: (((l_shipmode = 'AIR'::bpchar) OR (l_shipmode = 'AIR  
REG'::bpchar)) AND (l_shipinstruct = 'DELIVER IN PERSON'::bpchar))
```

cost が、「680256343229.03..680256343229.04」から「271799.95..271799.95」に下がる事が確認できた。これで、PostgreSQL7.4.6でも実用的な時間でDBT-3ワークロードを実行できるようになった。

今回の評価では、このQ19のSELECT文をPostgreSQL7.4.6用に変更するためのパッチを作成した。そのパッチの適用方法は、「6.3.1.2 PostgreSQL7.4.6の場合」に記述した。

なお、PostgreSQL8.0.0beta5では、オプティマイザが改良されており、SELECT文を変更しなくても、リスト6.4-5に示す実行プランが作成された。Q19のクエリーを変更しなくても、現実的な時間でDBT-3ワークロードを実施することが出来た。

#### リスト 6.4-5 PostgreSQL8.0.0beta5での実行プラン

```
Aggregate (cost=282579.51..282579.51 rows=1 width=8)
-> Merge Join (cost=258221.08..282579.30 rows=81 width=8)
    Merge Cond: ("outer".p_partkey = "inner".l_partkey)
    Join Filter: (((("outer".p_brand = 'Brand#51'::bpchar) AND
(("outer".p_container = 'SM CASE'::bpchar) OR ("outer".p_container = 'SM
BOX'::bpchar) OR ("outer".p_container = 'SM PACK'::bpchar) OR ("outer".p_container
= 'SM PKG'::bpchar)) AND ("inner".l_quantity >= 2::double precision) AND
("inner".l_quantity <= 12::double precision) AND ("outer".p_size <= 5)) OR
(("outer".p_brand = 'Brand#21'::bpchar) AND (("outer".p_container = 'MED
BAG'::bpchar) OR ("outer".p_container = 'MED BOX'::bpchar) OR ("outer".p_container
= 'MED PKG'::bpchar) OR ("outer".p_container = 'MED PACK'::bpchar)) AND
("inner".l_quantity >= 19::double precision) AND ("inner".l_quantity <= 29::double
precision) AND ("outer".p_size <= 10)) OR (((("outer".p_brand = 'Brand#45'::bpchar)
AND (("outer".p_container = 'LG CASE'::bpchar) OR ("outer".p_container = 'LG
BOX'::bpchar) OR ("outer".p_container = 'LG PACK'::bpchar) OR ("outer".p_container
= 'LG PKG'::bpchar)) AND ("inner".l_quantity >= 21::double precision) AND
("inner".l_quantity <= 31::double precision) AND ("outer".p_size <= 15)))
-> Index Scan using part_pkey on part (cost=0.00..8175.00 rows=199980
width=36)
    Filter: (p_size >= 1)
-> Sort (cost=258221.08..258743.93 rows=209140 width=16)
    Sort Key: lineitem.l_partkey
```

```
    -> Seq Scan on lineitem (cost=0.00..235619.26 rows=209140
width=16)
        Filter: (((l_shipmode = 'AIR'::bpchar) OR (l_shipmode = 'AIR
REG'::bpchar)) AND (l_shipinstruct = 'DELIVER IN PERSON'::bpchar))
```

#### 6.4.2.1.3      考察

PostgreSQL8.0.0beta5 では問題なく、7.4.6 では Q19 のクエリーを変更することで、DBT-3 ワークロードが実行できたことが確認できた。

また、PostgreSQL7.4 から 8.0 へのバージョンアップで実行プラン作成が改良されたことを、SQL 文の実行時間が劇的に短縮されたことで確認ができた。

DBT-3 ワークロードの Q19 のような SELECT 文では、PostgreSQL8.0 を使用した方が良いだろう。

## 6.4.2.2 ディスク構成による違い

### 6.4.2.2.1 目的

ディスク構成を変えて DBT-3 ワークロードを実行することで、ディスク構成による性能への影響を評価する。

PostgreSQL では、データと WAL をシステムログが書かれるディスクとは別々に配置する 3 台構成が性能が良いとされている。また、今回の評価では 1 台構成と 2 台構成ともに、PostgreSQL のデータと WAL を分離しないので、大量にシステムログが書かれることの無い通常の運用では、性能に差は無いと予想される。

ここでは、DBT-3 ワークロードにおいても、上記が成り立つのかを検証し、実際にどの程度の性能差があるのかを評価する。なお、スケールファクターは 1 を採用した。

### 6.4.2.2.2 結果

PostgreSQL7.4.6 での測定結果を表 6.4-2 に、8.0.0beta5 での測定結果を表 6.4-3 に示す。

表 6.4-2 PostgreSQL7.4.6

ディスク構成	ロードテスト	パワーテスト		スループットテスト	
	所要時間	所要時間	トランザクション数	所要時間	トランザクション数
3 台	00:17:08	00:08:57	324.72	00:23:56	220.61
2 台	00:17:45	00:08:42	308.45	00:26:37	198.37
1 台	00:18:00	00:08:55	327.30	00:27:36	191.42

表 6.4-3 PostgreSQL8.0.0beta5

ディスク構成	ロードテスト	パワーテスト		スループットテスト	
	所要時間	所要時間	トランザクション数	所要時間	トランザクション数
3 台	00:17:37	00:09:08	336.99	00:23:30	224.68
2 台	00:17:24	00:09:04	329.89	00:27:26	192.47
1 台	00:17:45	00:09:18	337.10	00:28:04	188.24

所要時間の単位は、「時：分：秒」である。

トランザクション数は、「1 時間あたりに実行できるトランザクション数 × スケールファクター数 (ここでは 1)」である。

(1) ロードテスト

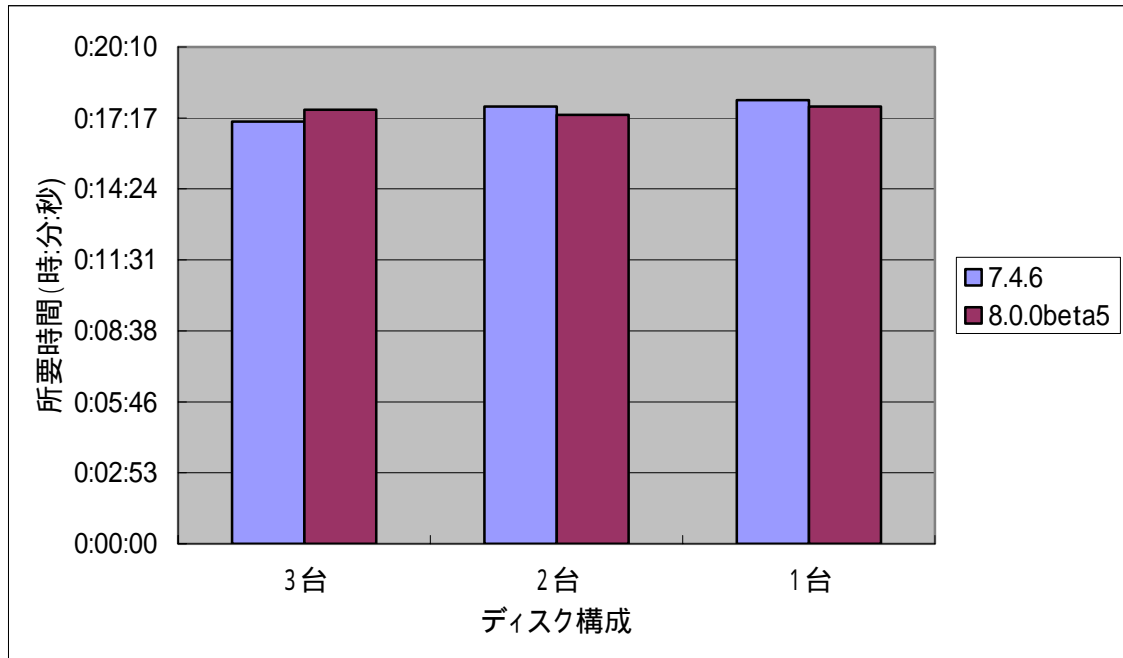


図 6.4-1 ロード時間

データのロードのみという単純な処理では、PostgreSQL のバージョンや、ディスク構成の違いによる差は、ほとんど無いという結果となった。

## (2) パワーテスト

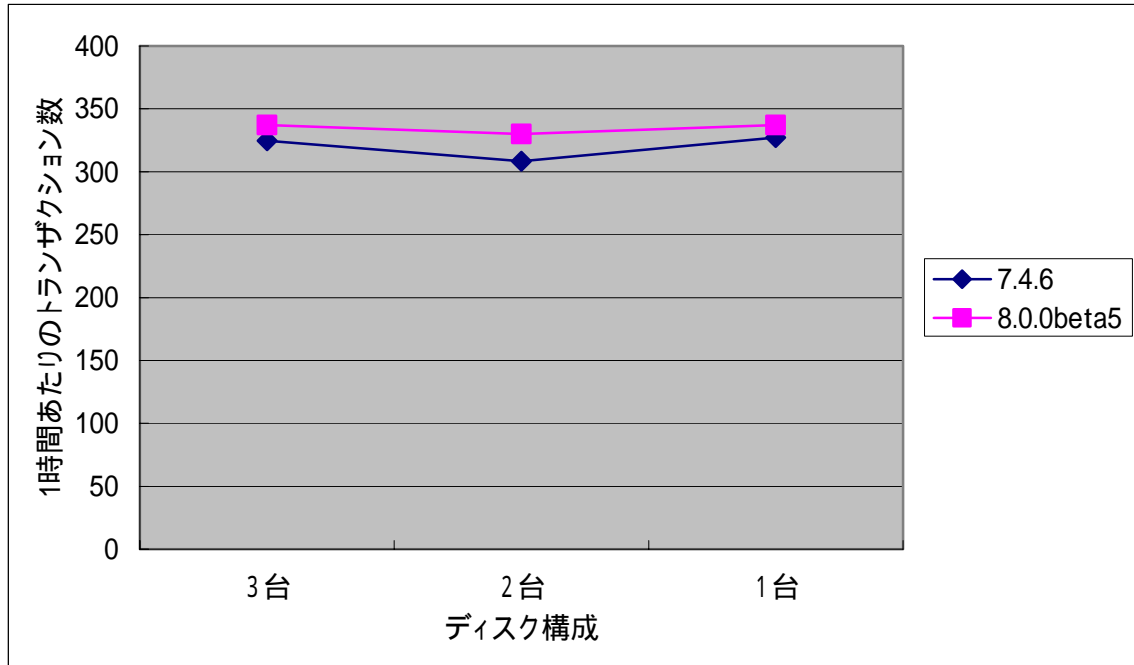


図 6.4-2 パワーテスト

PostgreSQL のバージョンによる違いとして、7.4.6 よりも 8.0.0beta5 の方が約 5%ほど性能が良いという結果となった。これは、7.4.6 では型違いのインデックスは使われなかったが、8.0.0beta5 では使うように改良されたので、その効果であることも考えられる。

ディスク構成から予想される結果は、3 台構成の性能が良く、続いて 2 台構成と 1 台構成が同等の性能であった。実際に実行したところ、3 台構成と 1 台構成がほぼ同じ性能で、2 台構成の性能が落ち込んだ。

これは、OS のディスクキャッシュ、ファイルシステムのフラグメント等も理由として考えられるが、それ以上に DBT-3 ワークロードの各クエリーを実行する際の検索条件のパラメータがランダムであることが原因と考えられる。

「6.4.2.3 誤差率の測定」にて、検索条件のパラメータを同じにして DBT-3 ワークロードを 10 回実行した結果を報告する。

### (3) スループットテスト

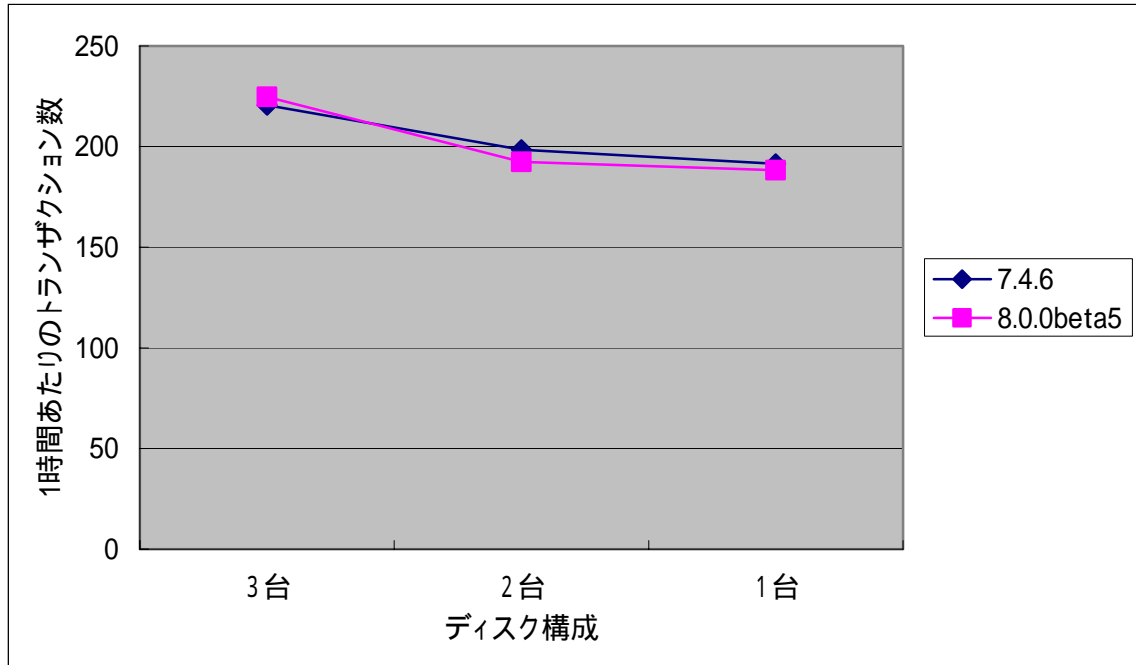


図 6.4-3 スループットテスト

ディスク構成から予想される結果は、3台構成の性能が良く、続いて2台構成と1台構成が同等の性能であり、そのことを確認する結果となった。特に3台構成の性能は良く、PostgreSQL8.0.0beta5において2台構成と比べると約16%の性能差が認められた。

パワーテストは同時に実行するトランザクション数が1だが、スループットテストは同時に実行するトランザクション数は任意に選ぶことができ、今回の評価では4を指定した。そのため、検索条件のパラメータがランダムであっても、4つのトランザクションの測定結果を合計するので、ディスク構成の予想通りの結果となったと考えられる。

### 6.4.2.2.3 考察

ディスク構成および、PostgreSQL のバージョン 7.4.6 と 8.0.0beta5 の違いによる、DBT-3 ワークロードの測定結果の評価を表 6.4-4 にまとめた。

表 6.4-4 ディスク構成の違い

ロードテスト	PostgreSQL のバージョンによる性能差はほとんど無い。 ディスク構成の違いによる性能差もほとんど無い。
パワーテスト	PostgreSQL8.0.0beta5 は、7.4.6 より約 5%ほど性能が良い。 ディスク構成は予想と異なり、2 台構成が 1 台構成より性能が低下した。クエリーの検索条件パラメータがランダムであることが原因であると考えられる。
スループットテスト	PostgreSQL のバージョンによる性能差はほとんど無い。 ディスク構成は予想通りに、3 台構成の性能が良かったが、約 16%という大きな違いとなる場合があった。

同時に実行するトランザクション数が 1 のパワーテストでは、PostgreSQL8.0.0beta5 は 7.4.6 より約 5%の性能向上が見られた。大規模データベースにおいて複雑なクエリーを一人のユーザが実行するような用途では、7.4.6 よりも 8.0.0beta5 の方が効果があるだろう。

ディスク構成による性能差は、同時に実行するトランザクション数が 4 のスループットテストにおいて、予想通りに 3 台構成の性能が良いことが確認できた。それに対して、パワーテストでは予想とは異なる結果となった。この理由として、同時に実行するトランザクション数が 1 なので、ディスクアクセスを分散する効果が少ないこともあるが、検索条件のパラメータがランダムであることが大きいと考えられる。

そこで、次の「6.4.2.3 誤差率の測定」では、検索条件のパラメータを固定したうえで、DBT-3 ワークロードを 10 回実行することで、誤差率の評価を行う。

### 6.4.2.3 誤差率の測定

#### 6.4.2.3.1 目的

DBT-3 ワークロードの測定結果の誤差率を求めるために、DBT-3 ワークロードを以下のように実行する。

- ・ DBT-3 ワークロードの、22 個のクエリーの検索条件のパラメータを決める乱数の元となる seed 値を同じ値（今回の評価では、1967000000）に固定して、同じパラメータで実行する。
- ・ 上記の条件で、DBT-3 ワークロードを 10 回実行して、その結果を評価する。

また、以下の環境で実行した。

- ・ PostgreSQL のバージョンは、7.4.6 と 8.0.0beta5。
- ・ ディスク構成は、3 台構成のみ
- ・ スケールファクターは、1

### 6.4.2.3.2 結果

PostgreSQL7.4.6での測定結果を表 6.4-5 に、8.0.0beta5での測定結果を表 6.4-6 に示す。

表 6.4-5 PostgreSQL7.4.6

実行番号	ロードテスト	パワーテスト		スループットテスト	
	所要時間	所要時間	トランザクション数	所要時間	トランザクション数
1	00:17:10	00:09:08	314.06	00:24:07	218.94
2	00:17:03	00:09:06	283.94	00:24:16	217.73
3	00:17:07	00:09:16	308.67	00:24:29	215.80
4	00:17:01	00:09:13	279.22	00:24:20	217.14
5	00:17:00	00:09:23	266.03	00:24:28	215.95
6	00:17:15	00:09:03	304.25	00:24:37	214.63
7	00:17:02	00:09:06	295.85	00:24:18	217.43
8	00:16:56	00:08:52	312.95	00:24:37	214.49
9	00:17:11	00:09:30	265.97	00:24:01	219.85
10	00:17:00	00:09:05	310.25	00:24:59	211.34

表 6.4-6 PostgreSQL8.0.0beta5

実行番号	ロードテスト	パワーテスト		スループットテスト	
	所要時間	所要時間	トランザクション数	所要時間	トランザクション数
1	00:17:23	00:09:11	336.36	00:23:41	222.94
2	00:17:25	00:09:03	308.91	00:25:15	209.11
3	00:17:23	00:09:00	336.41	00:24:41	214.05
4	00:17:27	00:09:02	337.82	00:24:48	213.05
5	00:17:27	00:09:14	327.10	00:24:19	217.28
6	00:17:25	00:09:01	309.07	00:24:24	216.54
7	00:17:25	00:09:05	337.61	00:24:26	216.10
8	00:17:38	00:08:58	337.45	00:23:57	220.61
9	00:17:36	00:08:51	341.11	00:24:02	219.85
10	00:17:29	00:09:18	294.49	00:24:49	212.90

所要時間の単位は、「時：分：秒」である。

トランザクション数は、「1時間あたりに実行できるトランザクション数 × スケールファクター数（ここでは1）」である。

(1) ロードテスト

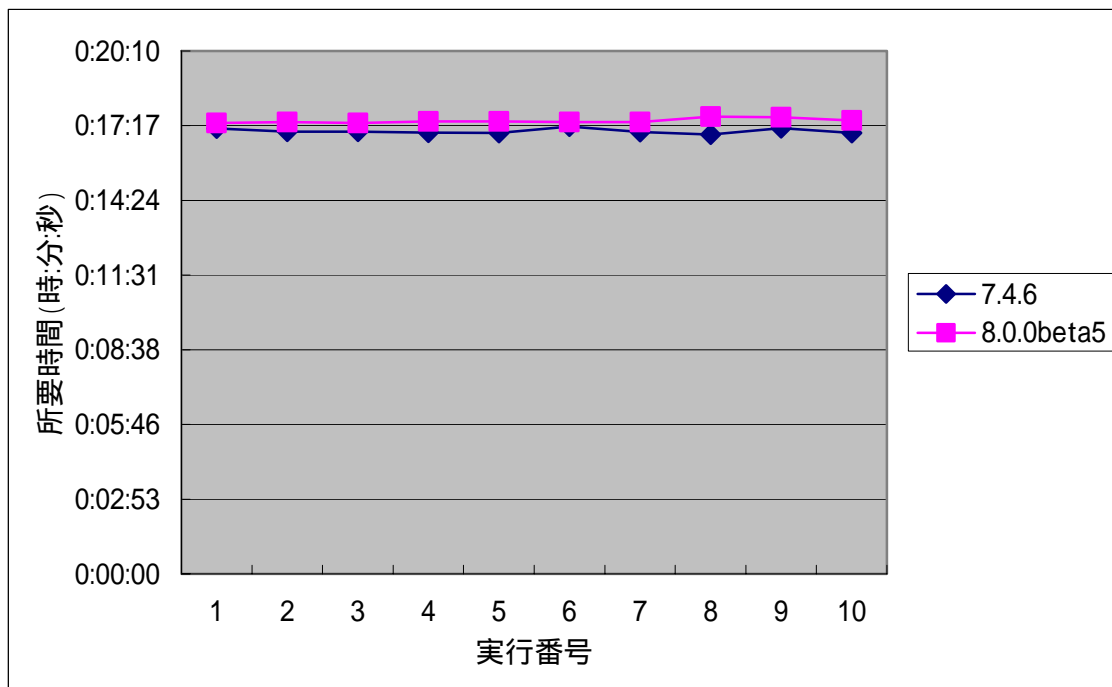


図 6.4-4 ロード時間

ロードテストを 10 回実行した結果の集計を、表 6.4-7 に示す。

表 6.4-7 ロード時間の集計

	平均	最小	最大	最大 - 最小	誤差率
7.4.6	00:17:04	00:16:56	00:17:15	00:00:19	約 1.84%
8.0.0beta5	00:17:28	00:17:23	00:17:38	00:00:15	約 1.42%

ロード時間については、10 回の実行による結果の違いは 2%未満と少なく、何回か実行しても安定した結果が得られた。

(2) パワーテスト

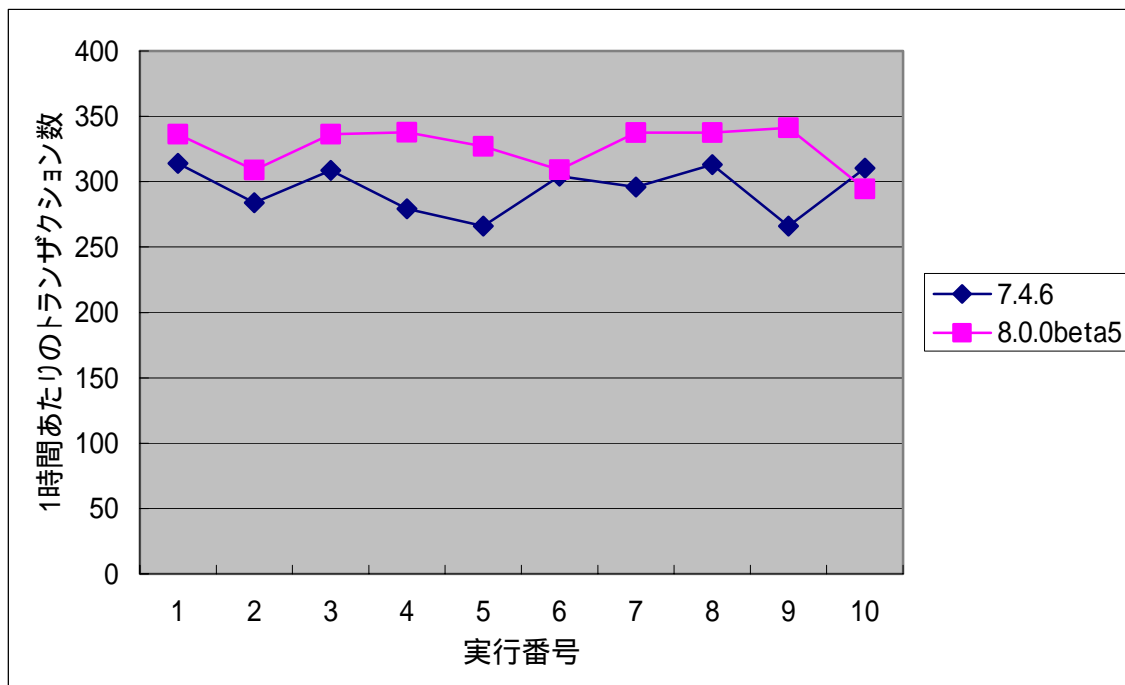


図 6.4-5 パワーテスト

パワーテストを 10 回実行した結果の集計を、表 6.4-8 に示す。

表 6.4-8 パワーテストの集計

	平均	最小	最大	最大 - 最小	誤差率
<b>7.4.6</b>	294.119	265.97	314.06	48.09	約 15.31%
<b>8.0.0beta5</b>	326.633	294.49	341.11	46.62	約 13.67%

平均を見ると、8.0.0beta5 は 7.4.6 よりも約 10%ほど性能が良いことが確認できた。しかし、DBT-3 ワークロードの実行条件を同じにしても、10 回のパワーテストの結果の違いは、7.4.6 で約 15%、8.0.0beta5 で約 14%という大きな結果となった。

そこで、PostgreSQL8.0.0beta5 で特に性能差の大きい実行番号 9 と実行番号 10 の測定結果について、各クエリーごとの所要時間が記録されたファイル (DBT-3 ワークロードの測定結果レポートの、q\_time.out のリンクから参照できる) を比較したところ、Q17 のクエリーの実行時間に大きな違いがあった。実行番号 9 は 1.296596 秒で、実行番号 10 は 10.213652 秒であり、その差は 8.91056 秒であった。他のクエリーの実行時間は、ほとんど同じなのに対して、Q17 のクエリーだけは大きな違いがあった。

この実行時間の違いの原因を調べるために、DBT-3 ワークロードのソースコードを変更して、Q17 のクエリーを実行する際に、SQL に「EXPLAIN ANALYZE」を付与した。これで、実際に実行したときの実行プランを取得できる。何度か実行した結果、速い場合と遅い場合で、異なる実行プランが使われていることが確認できた。速い場合をリスト 6.4-6 に、遅い場合をリスト 6.4-7 に示す。

#### リスト 6.4-6 速い場合

```

EXPLAIN ANALYZE select sum(l_extendedprice) / 7.0 as avg_yearly from lineitem,
part where p_partkey = l_partkey and p_brand = 'Brand#21' and p_container = 'SM
DRUM' and l_quantity < ( select 0.2 * avg(l_quantity) from lineitem where
l_partkey = p_partkey );

```

QUERY PLAN

---

```

Aggregate          (cost=927160.33..927160.33  rows=1  width=4)  (actual
time=1435.679..1435.679 rows=1 loops=1)
  ->  Nested Loop  (cost=0.00..927155.27  rows=2021  width=4)  (actual
time=21.383..1434.533 rows=616 loops=1)
    Join Filter: ("inner".l_quantity < (subplan))
    ->  Seq Scan on part  (cost=0.00..7863.18  rows=201  width=4)  (actual
time=2.053..212.654 rows=221 loops=1)
        Filter: ((p_brand = 'Brand#21'::bpchar) AND (p_container = 'SM
DRUM'::bpchar))
    ->  Index Scan using i_l_partkey on lineitem  (cost=0.00..134.42
rows=33 width=12)  (actual time=1.375..1.444 rows=30 loops=221)
        Index Cond: ("outer".p_partkey = lineitem.l_partkey)
    SubPlan
    ->  Aggregate  (cost=134.50..134.51  rows=1  width=4)  (actual
time=0.131..0.132 rows=1 loops=6646)
        ->  Index Scan using i_l_partkey on lineitem
(cost=0.00..134.42  rows=33  width=4)  (actual time=0.008..0.086  rows=31
loops=6646)
            Index Cond: (l_partkey = $0)
Total runtime: 1435.910 ms
(12 rows)

```

## リスト 6.4-7 遅い場合

```

EXPLAIN ANALYZE select sum(l_extendedprice) / 7.0 as avg_yearly from lineitem,
part where p_partkey = l_partkey and p_brand = 'Brand#21' and p_container = 'SM
DRUM' and l_quantity < ( select 0.2 * avg(l_quantity) from lineitem where
l_partkey = p_partkey );

```

QUERY

PLAN

---

```

Aggregate          (cost=1133019.30..1133019.30  rows=1  width=4)  (actual
time=17789.518..17789.518 rows=1 loops=1)
  -> Hash Join      (cost=7863.80..1133014.30  rows=2001 width=4)  (actual
time=344.823..17787.054 rows=616 loops=1)
    Hash Cond: ("outer".l_partkey = "inner".p_partkey)
    Join Filter: ("outer".l_quantity < (subplan))
      -> Seq Scan on lineitem  (cost=0.00..191565.79  rows=6031679
width=12) (actual time=12.690..8033.753 rows=6000935 loops=1)
        -> Hash          (cost=7863.30..7863.30  rows=199  width=4)  (actual
time=210.952..210.952 rows=0 loops=1)
          -> Seq Scan on part  (cost=0.00..7863.30  rows=199 width=4)
(actual time=2.147..210.297 rows=221 loops=1)
            Filter: ((p_brand = 'Brand#21'::bpchar) AND (p_container
= 'SM DRUM'::bpchar))
              SubPlan
                -> Aggregate  (cost=150.53..150.53  rows=1  width=4)  (actual
time=0.666..0.667 rows=1 loops=6646)
                  -> Index Scan using i_l_partkey on lineitem
(cost=0.00..150.44  rows=37  width=4)  (actual time=0.105..0.602  rows=31
loops=6646)
                    Index Cond: (l_partkey = $0)
Total runtime: 17789.742 ms
(13 rows)

```

速い場合は Nested Loop (入れ子ループ結合) を、遅い場合は Hash Join を使用していることが分かった。

データベースの内容は同じ、クエリーの検索条件のパラメータも同じという、同一と考えられる条件で、異なる実行プランが選択された理由については、今回の評価の期間内では原因を特定するまでには至らなかった。しかし、常に Nested Loop を使用するよ

うに PostgreSQL に指示することは出来る。その方法は、Q17 のクエリーを実行する直前に、Hash Join を使用しない（つまり Nested Loop を使う）ように、問い合わせの制御スイッチを設定すれば良い。具体的には、以下の SQL を実行する。

```
SET enable_hashjoin TO off;
```

DBT-3 ワークロードのソースコードを上記のように変更して、DBT-3 ワークロードを何度か実行したところ、常に Nested Loop が使われて、実行時間も約 1 秒と速く実行できることを確認した。

なお、今回の評価では上記の変更を行わずに、オリジナルのクエリーのまま、DBT-3 ワークロードの評価を行った。

(3) スループットテスト

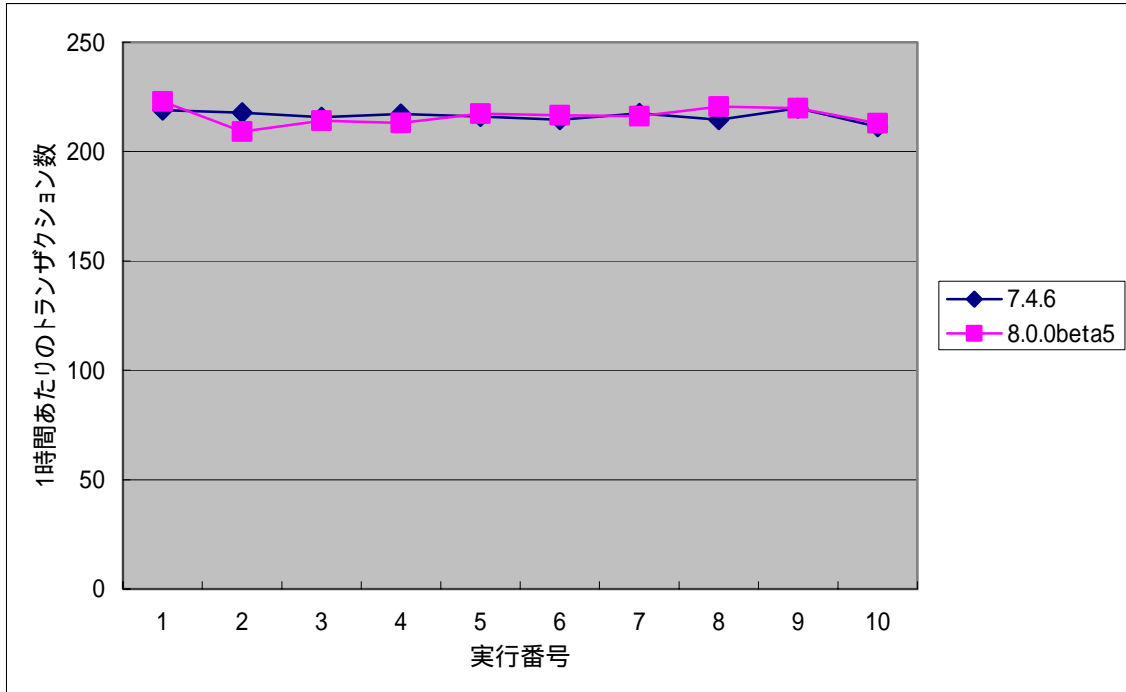


図 6.4-6 スループットテスト

スループットテストを 10 回実行した結果の集計を、表 6.4-9 に示す。

表 6.4-9 スループットテストの集計

	平均	最小	最大	最大 - 最小	誤差率
<b>7.4.6</b>	216.33	211.34	219.85	8.51	約 3.87%
<b>8.0.0beta5</b>	216.24	209.11	222.94	13.83	約 6.20%

スループットテストについては、10 回の実行による結果の違いは、7.4 で約 4%、8.0 で約 6%となった。この誤差率は、パワーテストと同様の理由で、17 番のクエリーが原因であるが、同時に実行するトランザクション数が 4 なので、その分だけ平均化されている。

### 6.4.2.3.3 考察

DBT-3 ワークロードの測定結果に、どの程度の誤差が出るかを確認するために、10 回実行して分かったことを、表 6.4-10 にまとめる。

表 6.4-10 誤差率の測定

ロードテスト	PostgreSQL7.4.6 と 8.0.0beta5 は、ほとんど同じ性能。 誤差は少ない。
パワーテスト	PostgreSQL7.4.6 と 8.0.0beta5 は、8.0 が 10%ほど性能が良い。 誤差は約 15%と大きく、DBT-3 ワークロードの Q17 のクエリーの実行プランが、遅い物を選択する可能性があるのが原因。問い合わせの制御スイッチで遅いプランを選択させないことで回避できた。
スループットテスト	PostgreSQL7.4.6 と 8.0.0beta5 は、ほとんど同じ性能。 誤差は約 5%であり、パワーテストと同様に 17 番クエリーが原因。

10 回実行した結果の誤差率は、ロードテストは僅かであったが、パワーテストで約 15%と大きかった。スループットテストは約 5%であるが、同時に実行するトランザクション数が 1 のパワーテストと違い、4 つのトランザクションが同時に実行されるので、平均化されたものと考えられる。

この誤差の原因は、DBT-3 ワークロードの Q17 のクエリーにあり、データベースの内容が同一で、実行するクエリーの順序が同じであっても、異なる実行プランが選択される場合があった。これは、PostgreSQL のバージョン 7.4.6 と 8.0.0beta5 および 8.0.1 で確認した。PostgreSQL のオプティマイザには、まだ改善の余地があると考えられる。

PostgreSQL7.4.6 と 8.0.0beta5 を比較すると、同時に実行するトランザクション数が 1 のパワーテストでは、8.0.0beta5 は 7.4.6 より約 10%の性能向上が見られた。大規模データベースにおいて複雑なクエリーを一人のユーザが実行するような用途では、7.4.6 よりも 8.0.0beta5 の方が効果があるだろう。

#### 6.4.2.4 スケールファクターによる違い

##### 6.4.2.4.1 目的

スケールファクターを変えたパターンで、DBT-3 ワークロードを実行し、データベースの規模による影響を測定した。ディスク構成は、3 台のハードディスクを使用した構成を採用した。

##### 6.4.2.4.2 結果

PostgreSQL7.4.6 での測定結果を表 6.4-11 に、8.0.0beta5 での測定結果を表 6.4-12 に示す。

表 6.4-11 PostgreSQL7.4.6

スケール ファク タ ー	ロードテスト	パワーテスト		スループットテスト	
	所要時間	所要時間	トランザク ション数	所要時間	トランザク ション数
1	00:17:18	00:08:57	324.72	00:23:56	220.61
2	00:40:42	00:51:34	100.56	03:25:38	51.35
4	01:39:31	04:05:55	53.86	15:17:04	23.03
6	02:37:01	07:53:45	48.31	31:00:40	17.02
10	04:58:27	19:18:29	45.53	64:54:10	13.55

表 6.4-12 PostgreSQL8.0.0beta5

スケール ファク タ ー	ロードテスト	パワーテスト		スループットテスト	
	所要時間	所要時間	トランザク ション数	所要時間	トランザク ション数
1	00:17:37	00:09:08	336.99	00:23:30	224.68
2	00:39:52	01:08:45	94.86	03:49:34	46.00
4	01:28:13	04:45:23	52.61	18:24:50	19.12
6	02:17:27	08:44:48	53.81	29:26:10	17.93
10	03:57:04	17:42:21	48.14	68:39:50	12.81

所要時間の単位は、「時：分：秒」である。

トランザクション数は、「1 時間あたりに実行できるトランザクション数 × スケールファクター数」である。

(1) ロードテスト

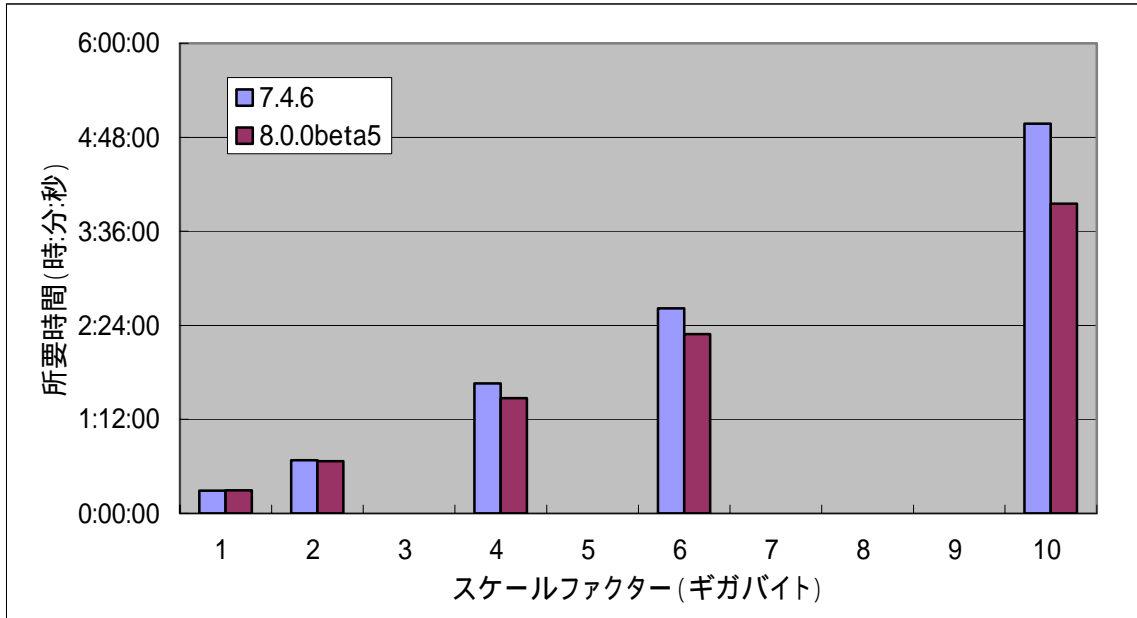


図 6.4-7 ロード時間

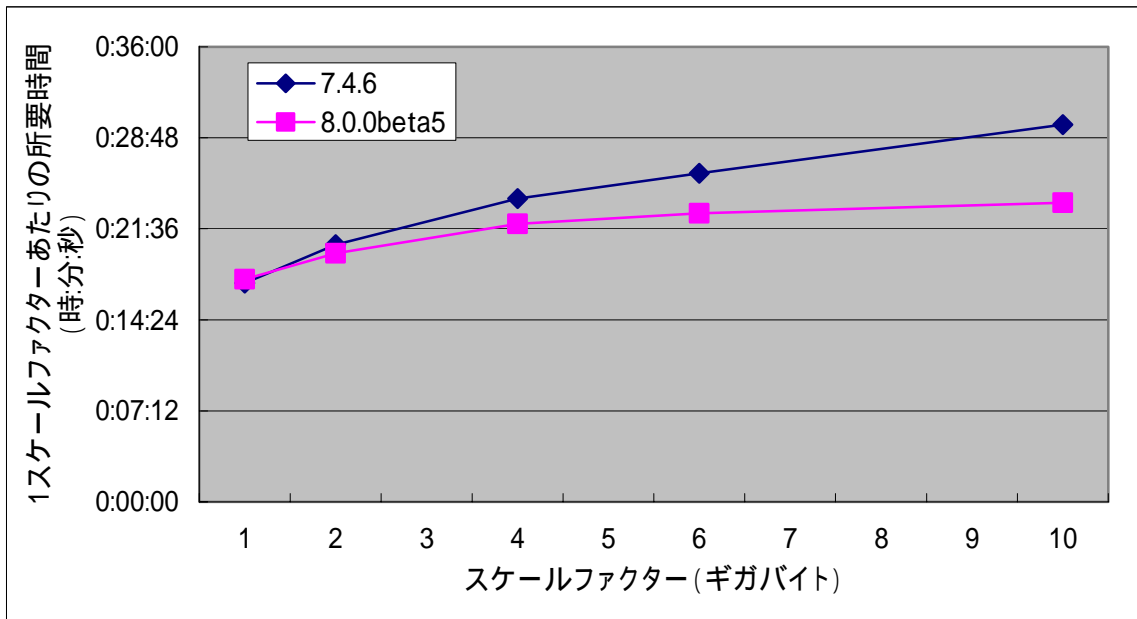


図 6.4-8 1スケールファクターあたりのロード時間

PostgreSQL7.4.6 と 8.0.0beta5 では、スケールファクターが 1 では 7.4.6 の方が若干ながら速くロードできるが、スケールファクターが大きくなればなるほど 8.0.0beta5 の方が速くロードできるという結果となった。

## (2) パワーテスト

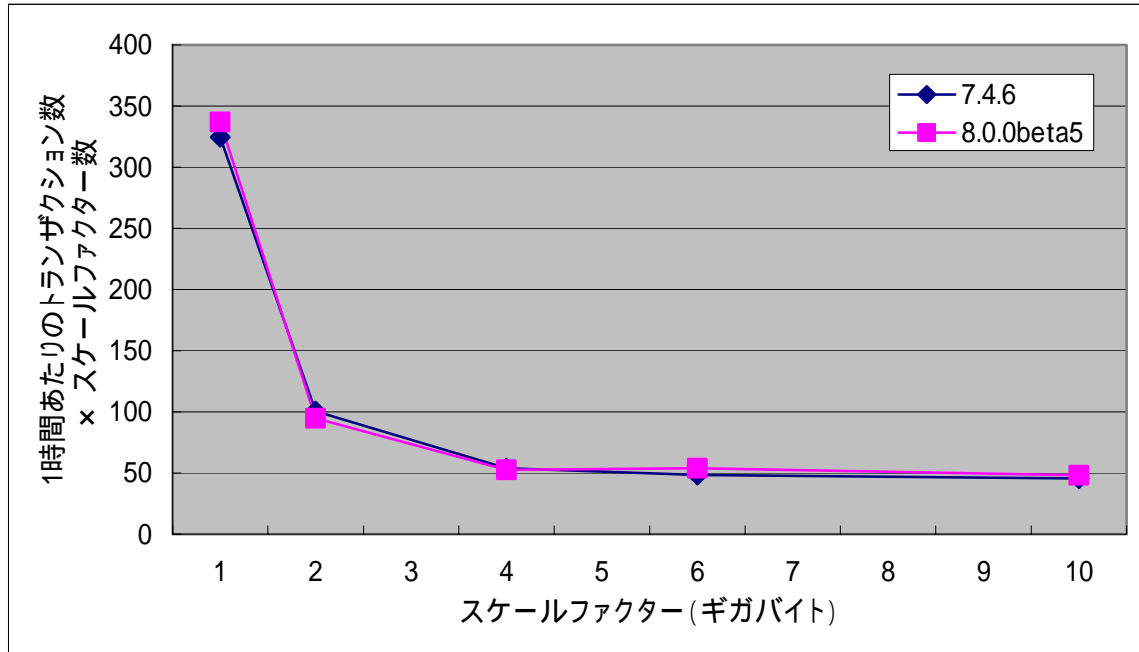


図 6.4-9 パワーテスト

パワーテストの結果は、「1時間あたりに実行できるトランザクション数 × スケールファクター数」である。スケールファクターを掛けるのは、TPC-Hで規定されているからであり、規模を大きくしても結果の数値は一定の値となるように計算される。

グラフでは、スケールファクター4と10では性能が変わっていないように見えるが、トランザクション数はスケールファクター数に反比例しているので、数値の読み方に注意する必要がある。

スケールファクター1~4では、実行可能なトランザクション数が減少していることが分かる。データベースの規模が大きくなるに従い、キャッシュメモリにデータが乗らなくなっていることが考えられる。

スケールファクター4~10では、データベースの規模とトランザクション数が反比例した。キャッシュメモリにデータが乗らないので、キャッシュがヒットしなくなることによって性能が一定になったものと考えられる。おおよそ、1時間あたりに実行可能なトランザクション数は「50 / スケールファクター数」で計算できることになる。

### (3) スループットテスト

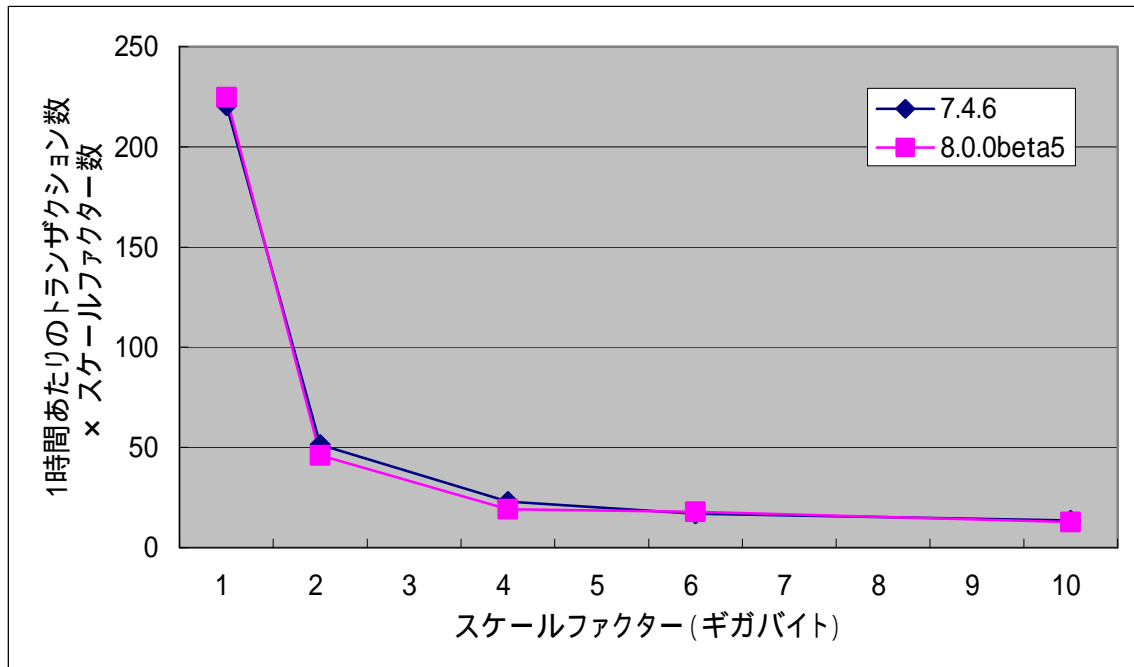


図 6.4-10 スループットテスト

スループットテストの結果は、パワーテストと同様に「1 時間あたりに実行できるトランザクション数 × スケールファクター数」であり、数値の読み方に注意する必要がある。

スケールファクター1~4では、実行可能なトランザクション数が減少していることが分かる。データベースの規模が大きくなるに従い、キャッシュメモリにデータが乗らなくなっていることが考えられる。

スケールファクター4~10では、データベースの規模とトランザクション数は反比例に近い。キャッシュメモリにデータが乗らないので、キャッシュがヒットしなくなることで性能が一定に近くなったものと考えられる。しかし、パワーテストとは異なり、性能は規模に応じて若干の劣化傾向が見られた。

#### 6.4.2.4.3 考察

スケールファクターを変えたパターンで、DBT-3 ワークロードを実行し、データベースの規模による影響を測定した結果を、表 6.4-13 にまとめる。

表 6.4-13 スケールファクターによる違い

ロードテスト	PostgreSQL7.4.6 と 8.0.0beta5 は、スケールファクターを大きくするほど 8.0.0beta5 の性能が良くなった。
パワーテスト	PostgreSQL7.4.6 と 8.0.0beta5 は、ほとんど同じ性能。 スケールファクター1~4 は、規模が小さい方が性能が良く、キャッシュメモリの効果と考えられる。スケールファクター4~10 は、規模に反比例した性能を維持した。
スループットテスト	PostgreSQL7.4.6 と 8.0.0beta5 は、ほとんど同じ性能。 スケールファクター1~4 は、規模が小さい方が性能が良く、キャッシュメモリの効果と考えられる。スケールファクター4~10 は、規模に反比例した性能だが、若干の劣化傾向があった。

ファイルからデータを投入するロードでは、規模を大きくするほど 7.4.6 よりも 8.0.0beta5 の方が性能が良かった。大規模なデータのロードでは、8.0.0beta5 の方が効果があるだろう。

スケールファクター1~4 では、キャッシュメモリのヒット率が性能を左右していると考えられる。今回の評価では実施しなかったが、PostgreSQL の共有メモリ、OS のディスクキャッシュなどを変更すると、結果が違う可能性がある。

スケールファクター4~10 では、規模が大きすぎてキャッシュに乗らなかったと考えられる。そのため、キャッシュの影響が少ないことで、性能が一定に近くなったのだと考えられる。

なお、今回の評価では実施しなかったが、スケールファクターを 11 以上にした場合に、どのような性能になるのかを測定することは十分に価値のあることであろう。

## 6.4.2.5 個別のチューニング

### 6.4.2.5.1 目的

DBT-3 ワークロードで実行する 22 個のクエリーに対して、トランザクションの特性を解析してチューニングを行い、その効果を測定して評価する。

ここでは、DBT-3 ワークロードとしてではなく、一つずつのクエリーとして個別にチューニングを行って、その効果を評価する。

なお、「6.4.2.7 チューニングによる違い」の節にて、ここで効果を確認したチューニングに基づいて、DBT-3 ワークロードを実行して、個別のクエリーではなくワークロード全体としての効果を確認する。

### 6.4.2.5.2 結果

性能改善のために、以下のインデックスを追加した。

```
create index i_p_size on part(p_size);
create index i_l_shipdate_discount_quantity on
  lineitem(l_shipdate, l_discount, l_quantity);
create index i_p_type on part(p_type);
create index i_p_brand_container on part(p_brand, p_container);
create index i_l_shipmode on lineitem(l_shipmode);
create index i_p_name on part(p_name);
create index i_substr_phone_index on customer((substr(c_phone, 1, 2)));
```

また、各クエリーごとにソートメモリを増やす、random\_page\_cost を変更するといった個別のチューニングを行った。具体的なチューニングの内容と結果については、付録資料の「DBT-3 トランザクション特性の解析」を参照のこと。

これらのチューニングを行って、22 個のクエリーを個別に実行した結果を、表 6.4-14 と表 6.4-15 に示す。所要時間は、チューニング前と比べて、チューニング後はどのくらいの時間で実行できたかの割合である。

表 6.4-14 PostgreSQL7.4.6

問い合わせ番号	主な改善内容	所要時間
Q2	インデックスの設定	79%
Q3	ソートメモリの設定	76%
Q8	ソートメモリ, インデックスの設定	54%
Q9	ソートメモリ, インデックスの設定	35%
Q14	インデックスの設定	37%
Q16	ソートメモリの設定	81%

Q17	インデックスの設定	61%
Q19	インデックスの設定	70%
Q20	インデックスの設定	91%
Q22	インデックスの設定	50%

表 6.4-15 PostgreSQL8.0.0beta5

問い合わせ番号	主な改善内容	所要時間
Q2	インデックスの設定	73%
Q3	ソートメモリの設定	70%
Q6	インデックスの設定	15%
Q8	ソートメモリ, インデックスの設定	63%
Q10	ソートメモリの設定	79%
Q14	インデックスの設定	35%
Q16	ソートメモリの設定	70%
Q17	インデックスの設定	54%
Q20	インデックスの設定	88%
Q22	インデックスの設定	51%

#### 6.4.2.5.3 考察

主にインデックスを追加した効果で、性能が向上したことが確認できた。DBT-3 ワークロードのような大規模なデータベースで、複雑なクエリーを実行するような場合でも、適切なインデックスの設定は、より良いパフォーマンスを得るためには重要であろう。

## 6.4.2.6 個別のテーブルスペース

### 6.4.2.6.1 目的

DBT-3 ワークロードで実行する 22 個のクエリーの中の一つに対して、PostgreSQL8.0.0beta5 のテーブルスペース機能を使用するように設定して、その効果を測定して評価する。

なお、「6.4.2.7 チューニングによる違い」の節にて、ここで効果を確認したテーブルスペースを使用して、DBT-3 ワークロードを実行して、個別のクエリーではなくワークロード全体としての効果を確認する。

### 6.4.2.6.2 結果

ここでは、問い合わせ番号 Q3 について、テーブルスペース機能を使用する。このクエリーは、22 個のクエリーの中でも比較的大きな 3 つのテーブル (lineitem、orders、customer) に対して検索を行うので、テーブルスペースの効果が検証しやすいと考えられる。これら 3 つのテーブルを、テーブルスペース機能を使用して表 6.4-16 のように、別々のディスクに配置した。具体的なテーブルスペースの割り当て方については、付録資料の「DBT-3 トランザクション特性の解析」を参照のこと。

表 6.4-16 テーブルスペースのディスク割り当て

ディスク	マウントポイント	用途	
36GB	/db_xlog	PostgreSQL の WAL 領域	
72GB	/dbt3_0	PostgreSQL のデータ領域	
36GB	/dbt3_1	PostgreSQL の テーブルスペース	lineitem テーブル
72GB	/dbt3_2		orders テーブル
72GB	/dbt3_3		customer テーブル

表 6.4-17 にテーブルスペース機能を使用して問い合わせ番号 Q3 を実行した結果を示す。所要時間は、チューニング前と比べて、チューニング後はどのくらいの時間で実行できたかの割合である。

表 6.4-17 PostgreSQL8.0.0beta5

問い合わせ番号	主な改善内容	所要時間
Q3	テーブルスペースの使用	77.3%

#### 6.4.2.6.3      考察

問い合わせ計画コストは同様であるにもかかわらず、テーブルスペースを使用した場合の実行時間は短縮されたことが確認できた。

これは、実行にあたりテーブルスペースに割り当てたディスクを、アンマウント/マウントすることで、OSのバッファキャッシュをクリアした後に測定した結果である。アンマウント/マウントしなかった場合は、これほどの効果は無かった。

従って、OSのバッファキャッシュの効果が薄れるような、負荷の高い実環境や、より大規模のデータベースにおいては、テーブルスペースを利用するメリットがあると考えられる。

## 6.4.2.7 チューニングによる違い

### 6.4.2.7.1 目的

「6.4.2.5 個別のチューニング」と「6.4.2.6 個別のテーブルスペース」を、DBT-3ワークロード全体に適用して10回実行する。実行する条件は以下の通りで、「6.4.2.3 誤差率の測定」の結果と比較する。

- PostgreSQLのバージョンは、8.0.0beta5
- ディスクは、3台構成
- スケールファクターは、1

適用するチューニング内容は、主にインデックスを使い検索性能の向上を目的としているので、パワーテストには効果があると予想される。それに対して、ロードテストではインデックスの構築時間が増えることになる。また、スループットテストでは、更新系のリフレッシュクエリーが検索系のクエリーと同時に実行されるので、その影響により全体としての性能の低下が予想される。

### 6.4.2.7.2 結果

チューニング後の測定結果を表 6.4-18 以下に示す。チューニング前の測定結果は表 6.4-6 を参照のこと。

表 6.4-18 PostgreSQL8.0.0beta5

実行番号	ロードテスト	パワーテスト		スループットテスト	
	所要時間	所要時間	トランザクション数	所要時間	トランザクション数
1	00:20:02	00:09:26	362.48	00:26:08	202.17
2	00:19:06	00:08:51	373.85	00:26:16	201.14
3	00:19:13	00:09:01	391.99	00:25:51	204.26
4	00:19:17	00:10:00	348.23	00:26:55	196.16
5	00:19:01	00:08:47	369.79	00:26:36	198.62
6	00:18:57	00:09:24	383.24	00:26:14	201.27
7	00:19:10	00:09:20	356.96	00:26:26	199.75
8	00:18:55	00:10:18	358.16	00:26:38	198.37
9	00:19:01	00:09:14	378.99	00:26:30	199.37
10	00:18:57	00:08:55	389.03	00:27:03	195.31

所要時間の単位は、「時：分：秒」である。

トランザクション数は、「1時間あたりに実行できるトランザクション数 × スケールファクター数（ここでは1）」である。

(1) ロードテスト

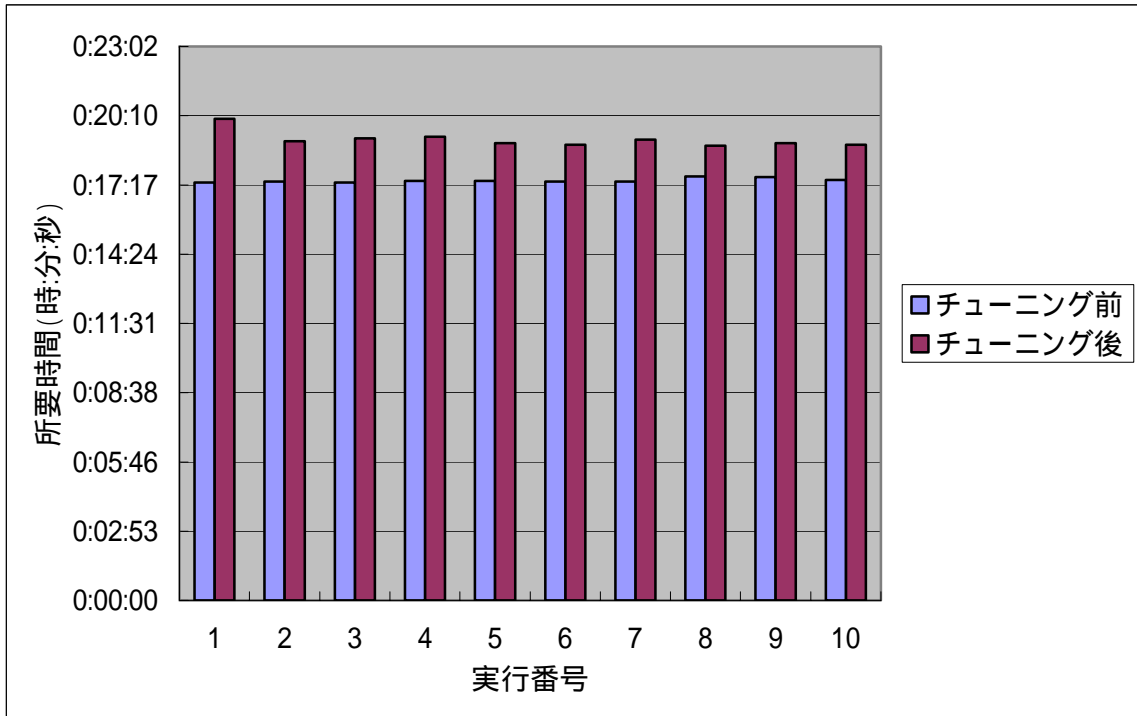


図 6.4-11 ロード時間

ロードテストを 10 回実行した結果の集計を、表 6.4-19 に示す。

表 6.4-19 ロード時間の集計

	平均	最小	最大	最大 - 最小	誤差率
チューニング前	00:17:28	00:17:23	00:17:38	00:00:15	約 1.42%
チューニング後	00:19:10	00:18:55	00:20:02	00:01:07	約 5.57%

ロード時間については、約 10%ほど遅くなった。

チューニングでは、検索速度を向上させるためのインデックスを追加した。それにより、更新系のロード時にインデックスを生成する個数が増えたことが、遅くなった原因である。

(2) パワーテスト

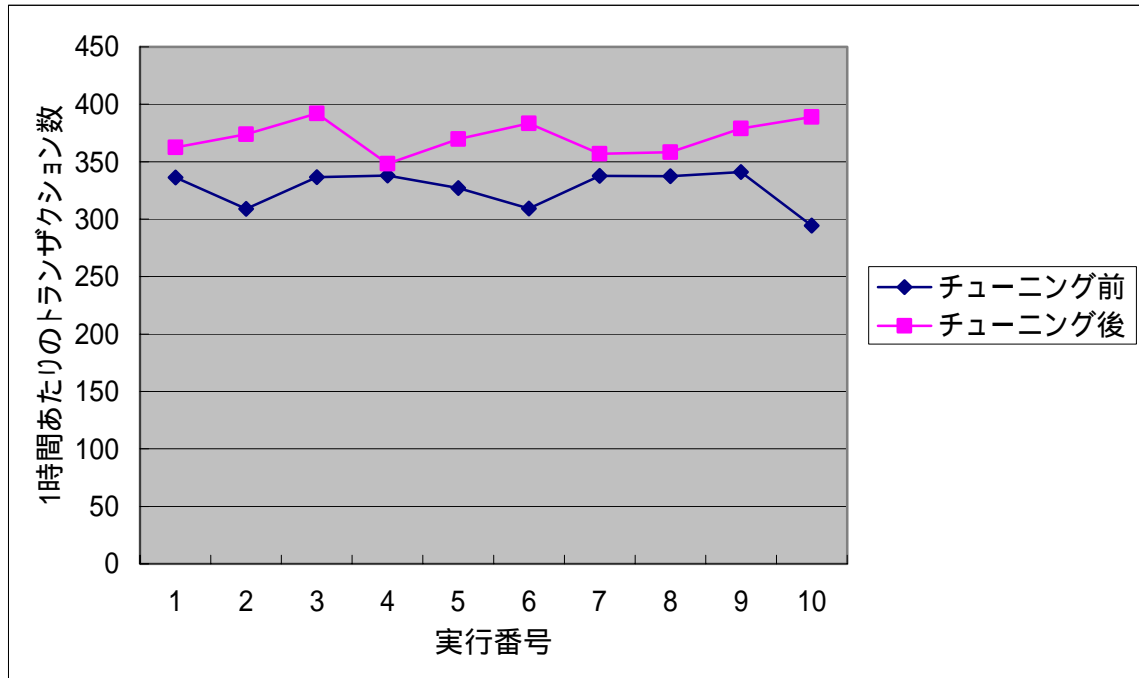


図 6.4-12 パワーテスト

パワーテストを 10 回実行した結果の集計を、表 6.4-20 に示す。

表 6.4-20 パワーテストの集計

	平均	最小	最大	最大 - 最小	誤差率
チューニング前	326.633	294.39	341.11	46.62	約 13.67%
チューニング後	361.272	348.23	391.99	43.76	約 11.16%

パワーテストについては、約 10%ほど速くなった。

チューニングでは、検索速度を向上させるためのインデックスの追加と、大量のデータを持つ 3 つのテーブルについてテーブルスペース機能を使用して専用のディスクを割り当てた。これらが効果的に使われたため、このように性能が向上したのであろう。

### (3) スループットテスト

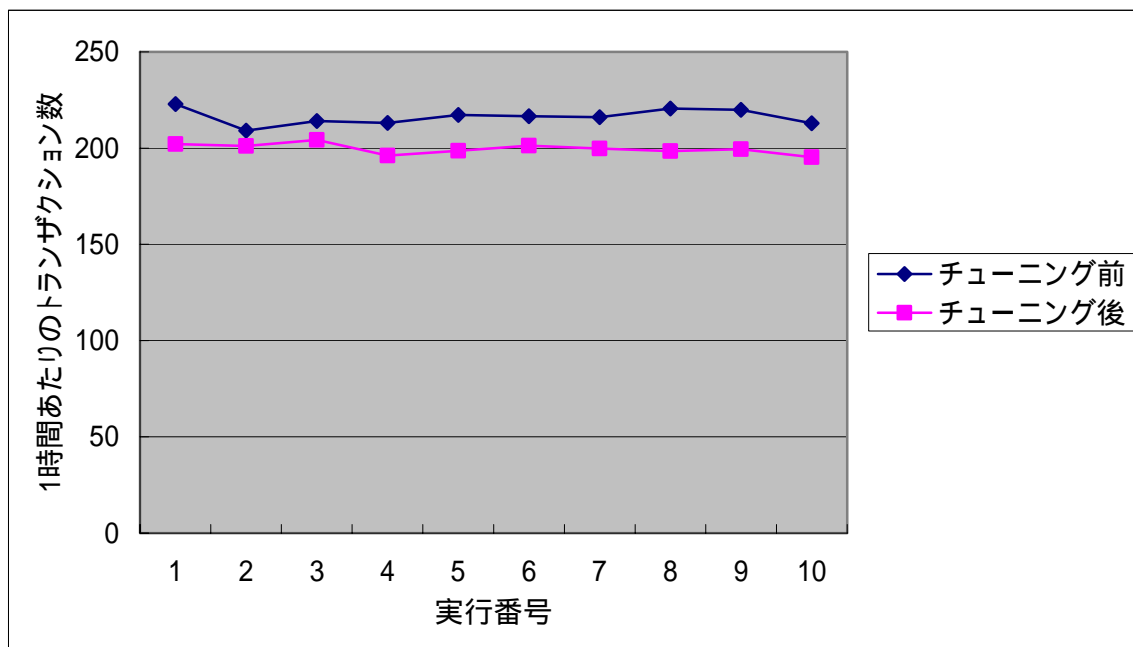


図 6.4-13 スループットテスト

スループットテストを 10 回実行した結果の集計を、表 6.4-21 に示す。

表 6.4-21 スループットテストの集計

	平均	最小	最大	最大 - 最小	誤差率
チューニング前	216.243	209.11	222.94	13.83	約 6.20%
チューニング後	199.642	195.31	204.26	8.95	約 4.38%

スループットテストでは、約 9%ほど遅くなった。

チューニングでは、検索速度を向上させるためのインデックスを追加した。DBT-3 ワークロードのスループットテストでは、22 個の検索系クエリーの他に、挿入と削除を行う 2 つのリフレッシュクエリーも実行する。この更新系の 2 つのクエリーを、それぞれ同時接続数の 4 回実行する。

これにより、インデックスの効果により検索系の処理は速くなるが、それ以上に更新系のリフレッシュクエリーに要する時間が増えたので、全体として約 9%遅くなるという結果となったと考えられる。

### 6.4.2.7.3 考察

「6.4.2.5 個別のチューニング」と「6.4.2.6 個別のテーブルスペース」を、DBT-3 ワークロード全体に適用して 10 回実行し、「6.4.2.3 誤差率の測定」の結果と比較した結果を、表 6.4-22 にまとめる。

表 6.4-22 チューニングによる違い

ロードテスト	約 10%ほど遅くなった。 インデックスを追加したため、その構築時間が増えたのが原因。
パワーテスト	約 10%ほど速くなった。 インデックスの追加とテーブルスペースの効果による。
スループット テスト	約 9%ほど遅くなった。 同時 4 接続で更新系クエリーが平行実行されるため、インデックス追加による検索系クエリーの性能向上よりも、インデックス更新による性能低下の割合が大きかったと考えられる。

チューニングで特に効果があるのはインデックスの作成だが、それは検索系のクエリーには効果的だが、更新系のクエリーでは逆効果となる。

同時に実行するトランザクション数が 1 のパワーテストでは、検索系と更新系のクエリーが同時に実行されることは無いので、インデックスの効果があった。それに対して、同時に実行するトランザクション数が 4 のスループットテストでは、検索系と更新系のクエリーが同時に実行されるので、全体としての性能が低下したのだと考えられる。

今回の評価では、スループットテスト時のトランザクション数に 4 を指定したが、これを 2 に指定した場合は性能が向上した可能性もある。また、4 よりも大きな値を指定した場合は性能が更に低下することが予想される。

チューニングにおいては、個別にクエリーをチューニングすることも重要だが、同時に実行されるトランザクション全体で測定することも必要である。これらのトレードオフを考慮して、それにあったチューニングを施すことが重要であろう。

## 6.4.2.8 PostgreSQL8.0.1 正式版と比較

### 6.4.2.8.1 目的

今回の評価を開始した時点では、PostgreSQL8.0の最新版はbeta5であったので、これまでの測定は8.0.0beta5で行った。今回の評価を終了する直前に、正式版の8.0.1がリリースされたので、8.0.0beta5と比較する目的で、DBT-3ワークロードを実行する。

DBT-3ワークロードは10回実行して、測定誤差の範囲も評価する。なお、DBT-3ワークロードを実行する条件は以下の通りとして、「6.4.2.3 誤差率の測定」の結果と比較する。

- PostgreSQLのバージョンは、8.0.1
- ディスクは、3台構成
- スケールファクターは、1

### 6.4.2.8.2 結果

PostgreSQL8.0.1での測定結果を表 6.4-23に示す。7.4.6での測定結果は表 6.4-5を、8.0.0beta5での測定結果は表 6.4-6を参照のこと。

表 6.4-23 PostgreSQL8.0.1

実行番号	ロードテスト	パワーテスト		スループットテスト	
	所要時間	所要時間	トランザクション数	所要時間	トランザクション数
1	00:17:51	00:08:59	340.93	00:24:33	215.07
2	00:17:38	00:09:13	302.66	00:24:55	211.91
3	00:17:37	00:09:05	333.61	00:23:54	221.07
4	00:17:29	00:08:59	336.60	00:23:30	224.68
5	00:17:30	00:09:09	323.66	00:24:53	212.33
6	00:17:33	00:09:02	335.02	00:23:35	223.89
7	00:17:37	00:09:01	307.26	00:24:35	214.93
8	00:17:34	00:08:53	337.64	00:24:06	219.24
9	00:18:02	00:09:00	307.90	00:24:42	213.91
10	00:17:42	00:09:05	333.08	00:24:25	216.39

所要時間の単位は、「時：分：秒」である。

トランザクション数は、「1時間あたりに実行できるトランザクション数 × スケールファクター数（ここでは1）」である。

(1) ロードテスト

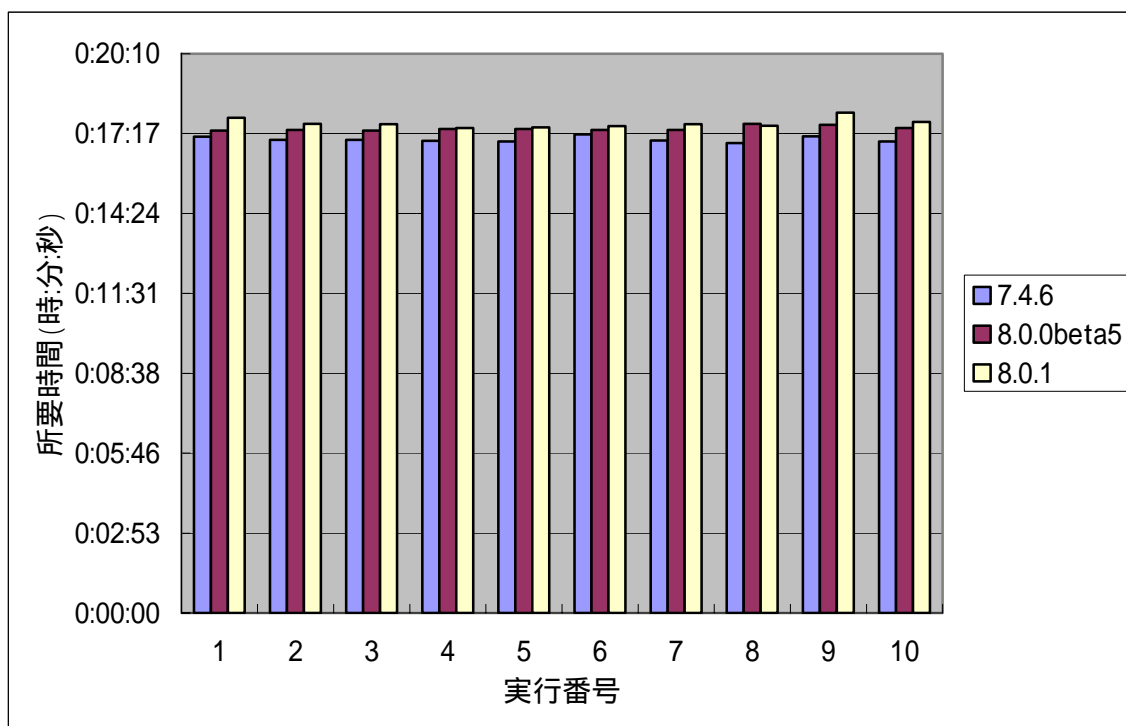


図 6.4-14 ロード時間

ロードテストを 10 回実行した結果の集計を、表 6.4-24 に示す。

表 6.4-24 ロード時間の集計

	平均	最小	最大	最大 - 最小	誤差率
<b>7.4.6</b>	00:17:04	00:16:56	00:17:15	00:00:19	約 1.84%
<b>8.0.0beta5</b>	00:17:28	00:17:23	00:17:38	00:00:15	約 1.42%
<b>8.0.1</b>	00:17:39	00:17:29	00:18:02	00:00:33	約 3.05%

8.0.0beta5 と、8.0.1 との違いは、わずかに 8.0.1 の方が遅いという結果になったが、その差は少ない。

(2) パワーテスト

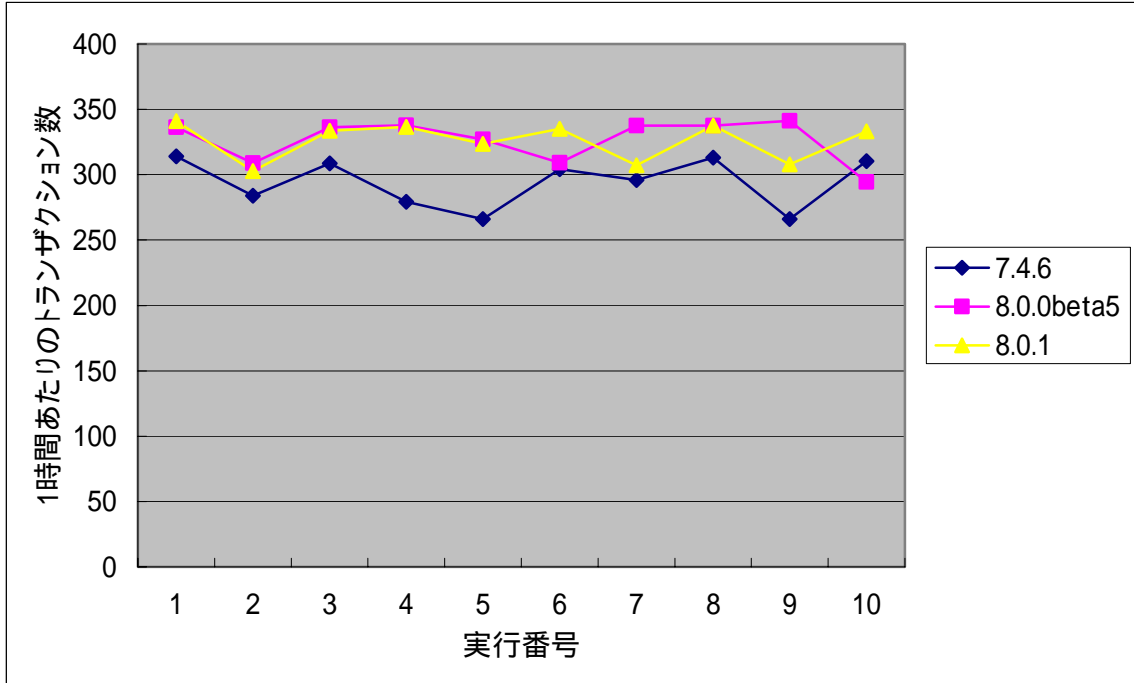


図 6.4-15 パワーテスト

パワーテストを 10 回実行した結果の集計を、表 6.4-25 に示す。

表 6.4-25 パワーテスト

	平均	最小	最大	最大 - 最小	誤差率
<b>7.4.6</b>	294.119	265.97	314.06	48.09	約 15.31%
<b>8.0.0beta5</b>	326.633	294.49	341.11	46.62	約 13.67%
<b>8.0.1</b>	325.836	302.66	340.93	38.27	約 11.23%

8.0.0beta5 と 8.0.1 を比較すると、ほとんど違いは無いという結果になった。

(3) スループットテスト

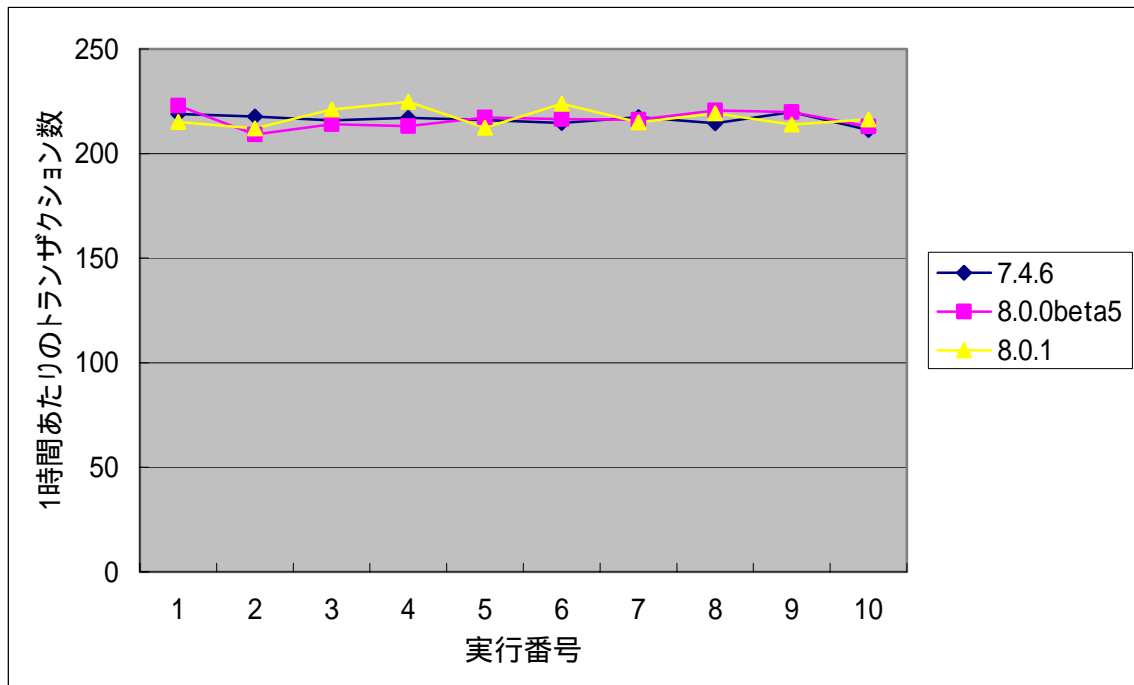


図 6.4-16 スループットテスト

スループットテストを 10 回実行した結果の集計を、表 6.4-26 に示す。

表 6.4-26 スループットテストの集計

	平均	最小	最大	最大 - 最小	誤差率
<b>7.4.6</b>	216.33	211.34	219.85	8.51	約 3.87%
<b>8.0.0beta5</b>	216.24	209.11	222.94	13.83	約 6.20%
<b>8.0.1</b>	217.348	211.97	224.68	12.71	約 5.66%

8.0.0beta5 と 8.0.1 を比較すると、ほとんど違いは無いという結果になった。

### 6.4.2.8.3 考察

PostgreSQL8.0 の最新版は、今回の評価開始時点では beta5 であったので、そのバージョンで評価してきた。今回の評価を終了する直前に、正式版の 8.0.1 がリリースされたので、そのバージョンで DBT-3 ワークロードを実行した。「6.4.2.3 誤差率の測定」で測定した 8.0.0beta5 と比較した結果を、表 6.4-27 にまとめる。

表 6.4-27 PostgreSQL8.0.1 正式版と比較

ロードテスト	8.0.1 は、8.0.0beta5 よりわずかに遅いが、その差は少ない。
パワーテスト	8.0.1 と 8.0.0beta5 は、ほとんど同じ性能。
スループットテスト	8.0.1 と 8.0.0beta5 は、ほとんど同じ性能。

8.0.0beta5 と 8.0.1 では、性能差はほとんど無いという結果となった。

これまでに、8.0.0beta5 で行った評価結果については、正式版の 8.0.1 でも同様の評価結果になると考えて問題ないだろう。

### 6.4.3 まとめ

DBT-3 バージョン 1.5 を使用して、PostgreSQL7.4.6 と 8.0.0beta5 の評価を行った。DBT-3 は、大規模データベースにおける意志決定支援のための情報検索を行うベンチマークである。DBT-3 のワークロードでは、大規模なデータを投入する「ロードテスト」、同時に一つのトランザクションで性能測定する「パワーテスト」、同時に指定数(今回の評価では 4) のトランザクションで性能測定する「スループットテスト」を実行する。

何種類かの条件で、DBT-3 ワークロードを実行した結果、以下のことが分かった。

- PostgreSQL8.0.0beta5 のオプティマイザは、7.4.6 より改良されていることが確認できた。DBT-3 のクエリーの中には、7.4.6 では 3 日待っても終わらない物があったが、8.0.0beta5 では数秒で実行できた。
- ディスク構成は、PostgreSQL のデータと WAL を専用のディスクに割り振る 3 台構成が、データと WAL を共存した構成と比べて約 16%ほど良い場合があった。
- パワーテストでは、10 回実行した平均を見ると、PostgreSQL8.0.0beta5 は 7.4.6 より約 10%ほど性能が良かった。7.4.6 では型違いのインデックスは使われなかったが、8.0.0beta5 では使うように改良されたので、その効果であることも考えられる。
- DBT-3 のクエリーの中には、同一条件であっても PostgreSQL の実行プランが変わってしまい、1 秒で完了する場合と 10 秒ほど掛かる場合があり、それが測定結果の性能差として現れた。問い合わせの制御スイッチを使うことで、常に 1 秒で完了できることも確認した。
- PostgreSQL8.0.0beta5 でのデータのロードは、データの規模が大きくなるほど、7.4.6 よりも速くロードできた。
- データベースの規模を、スケールファクターを 4 以上にすると、規模に応じた性能が一定に近くなった。スケールファクターが 4 未満では、キャッシュメモリの効果により性能が良い傾向があった。なお、今回の評価では、スケールファクターを 10 まで測定した。
- PostgreSQL8.0.0beta5 のテーブルスペースは、ディスクをアンマウント/マウントすることでバッファキャッシュをクリアした場合に効果が確認できた。バッファキャッシュの効果が薄れる、負荷の高い実環境や、より大規模なデータベースではテーブルスペースを利用するメリットがあると考えられる。
- インデックスを使って、DBT-3 のクエリーを個別にチューニングした場合は、パワーテストでは効果があるが、スループットテストでは逆効果であった。チューニングでは、クエリーの個別チューニングも大切だが、同時に実行されるクエリーを考慮することも重要である。
- PostgreSQL8.0.1 正式版で DBT-3 を実行した結果は、8.0.0beta5 とほぼ同じであった。今回、8.0.0beta5 で評価した結果は、8.0.1 でも同様の評価結果になると考えて問題ないだろう。

今回の評価では、PostgreSQL を利用した、DBT-3 ワークロードの測定手順を確立することが出来た。今回使用した、DBT-3 バージョン 1.5 に付属するマニュアルは、バージョン 1.4 の記述と混在しており、付属マニュアルの通りでは DBT-3 ワークロードを実行することはできなかった。今回、確立した手順により、容易に DBT-3 ワークロードを実行して、性能を測定できることが出来るようになった。

また、今回の評価を行った時点では、実際に利用しているユーザが少ない PostgreSQL8.0 においても、大規模データベースの性能測定を行う DBT-3 ワークロードを実際に動かして評価することができた。また、PostgreSQL7.4 と比較して評価することで、PostgreSQL8.0 は着実に成長していることが確認できた。

DBT-3 は、大規模データベースにおける複雑なクエリーの実行において、PostgreSQL の性能を測定するのに、非常に有効なツールであることを、今回の評価を通して確信できた。

#### 今後の課題

DBT-3 に限らないが、オープンソースプロダクトは開発元へのフィードバックは不可欠である。今回の評価では、多くのバグ修正を開発元へフィードバックできた。DBT-3 は、今回限りではなく継続的に利用するのが望ましく、開発元にフィードバックすることが強く望まれる。

今後は大規模データベースにも PostgreSQL などのオープンソースデータベースが採用されていけよう。大規模になればなるほど物理メモリを利用した OS のバッファキャッシュの効果が望めなくなる。今回の評価では OS のバッファキャッシュをクリアして測定して効果を確認したが、OS のバッファキャッシュを含めた、性能への影響を調べるための方法は確立していない。この点も今後の必須課題の一つとなるだろう。

## 7 大規模 DB 性能(運用性)の評価

### 7.1 概要

大規模データベース環境における性能評価として、以下について評価する。

- ・ バックアップとリカバリに要する時間の性能評価
- ・ 大容量データのロードに要する時間の性能評価
- ・ インデックスの再構築に要する時間の性能評価
- ・ PostgreSQL8.0 の Point In Time Recovery(PITR)を使用し、クラッシュリカバリに要する時間の性能評価

評価にあたって、PostgreSQL に付属するベンチマークツールである `pgbench` を使用して大規模なデータベースを生成する。

`pgbench` とは、TPC-B に似たベンチマークテストを行うプログラムであり、データベースへのテーブルの作成およびデータを投入する機能と、`select`、`update`、`insert` を含むトランザクションを実行して 1 秒あたりに実行できたトランザクション数 (tps) を求める機能がある。

今回の評価では、大容量データのロードに `pgbench` を使用し、それ以外の評価については PostgreSQL の標準のコマンドを使用して、性能評価を行った。

なお、PostgreSQL のバージョンについては、今回の評価を始めた時点での最新バージョンである、7.4.6 と 8.0.0beta5 を使用した。

### 7.1.1 テーブル構成

pgbench では、図 7.1-1 のテーブルを作成して、データを投入することが出来る。

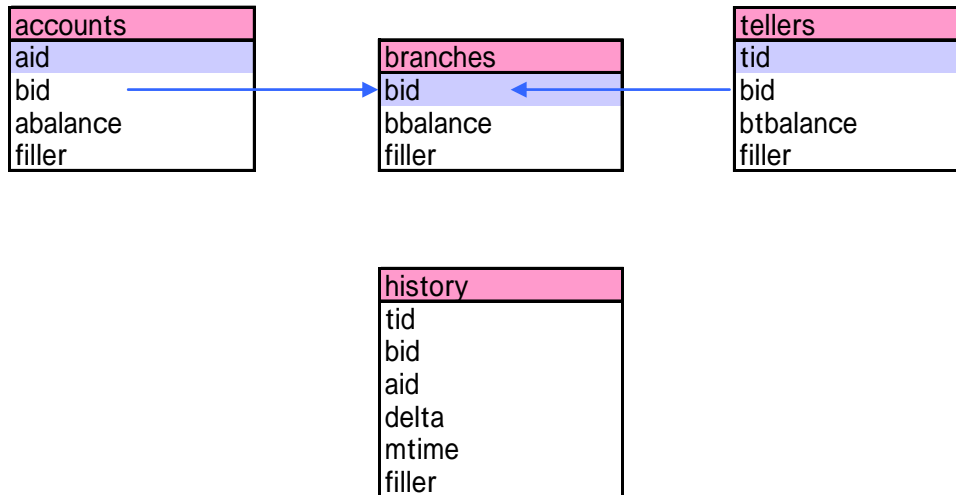


図 7.1-1 テーブル

pgbench を実行する際に、スケールファクターを指定する。スケールファクターには 1 以上の整数を指定する。各テーブルに投入するデータの行数は、スケールファクターの値を元に、表 7.1-1 の通り決定される。

表 7.1-1 テーブル一覧

テーブル名	説明	行数
branches	支店	スケールファクター × 1
tellers	窓口	スケールファクター × 10
accounts	口座	スケールファクター × 100000
history	履歴	0

それぞれのテーブルにおけるカラムの説明を、表 7.1-2 から表 7.1-5 までに示す。

表 7.1-2 branches (支店) テーブル

カラム名	データ型	制約	説明
bid	integer	primary key	支店 ID
bbalance	integer		残高
filler	character(88)		空き領域

表 7.1-3 tellers (窓口) テーブル

カラム名	データ型	制約	説明
tid	integer	primary key	窓口 ID
bid	integer		支店 ID
tbalance	integer		残高
filler	character(84)		空き領域

表 7.1-4 accounts (口座) テーブル

カラム名	データ型	制約	説明
aid	integer	primary key	口座 ID
bid	integer		支店 ID
abalance	integer		残高
filler	character(84)		空き領域

表 7.1-5 history (履歴) テーブル

カラム名	データ型	制約	説明
tid	integer		窓口 ID
bid	integer		支店 ID
aid	integer		口座 ID
delta	integer		デルタ
mtime	timestamp without time zone		日時
filler	character(22)		空き領域

history テーブルは、pgbench のベンチマーク機能を実行する場合に使用する。今回の評価では、pgbench はデータ投入に使用するので、history テーブルを使うことは無い。

## 7.2 環境定義

### 7.2.1 システム構成

今回の評価では、表 7.2-1 と表 7.2-2 で示すシステム構成で検証を行った。

表 7.2-1 ハードウェア構成

製品名	HP ProLiant DL380 Generation 3
プロセッサ	インテル Xeon プロセッサ 2.80GHz、2CPU
メモリ	2.5GByte
ハードディスク	Ultra SCSI 320 HDD 15000rpm 内蔵ドライブベイに、6 台 外部ストレージに、2 台

表 7.2-2 ディスク構成

ディスク容量	パーティション割り当て	マウントポイント	用途
36GB	1GB	/boot	
	4GB	/swap	
	4GB	/	
	4GB	/usr	
	1GB	/var	
	1GB	/tmp	
	残りすべて	/home	
36GB	すべて	/db_xlog	PostgreSQL の WAL 領域
72GB	すべて	/dbt3_0	PostgreSQL のデータ領域
36GB	すべて	/dbt3_1	accounts テーブルのインデックス (8.0 のテーブルスペース用)
72GB	すべて	/dbt3_2	accounts テーブル (8.0 のテーブルスペース用)
72GB	すべて	/dbt3_3	アーカイブログ (8.0 のクラッシュリカバリ用)
72GB	すべて	/dbt3_4	バックアップ先
72GB	すべて	/dbt3_5	未使用

すべてのパーティションは ext3 ファイルシステムでフォーマットして、ext3 のジャーナリングのモードも、デフォルトの orderd モードを使用する。  
以下のディレクトリについては、その所有者を postgres ユーザにしておくこと。  
/db\_xlog、/dbt3\_0、/dbt3\_1、/dbt3\_2、/dbt3\_4

## 7.2.2 インストール

### 7.2.2.1 Linux のインストール

Linux のインストールでは、PostgreSQL をコンパイルするために開発環境のパッケージを加える必要がある。

今回の評価では、MIRACLE LINUX V3.0 - Asianux Inside をインストールした。MIRACLE LINUX のインストール時に選択するパッケージは、「インストールするパッケージを選択」画面で「パッケージのセットをカスタマイズ」を選択し、以下の項目を選択した。

- ・ システムツール
- ・ X Window System
- ・ KDE
- ・ Web ブラウザ
- ・ 日本語サポート
- ・ 開発環境

また、MIRACLE LINUX V3.0 のカーネルは、kernel-2.4.21-9.38AX にアップデートした。

### 7.2.2.2 PostgreSQL のインストール

今回の評価では、バージョン 7.4.6 と 8.0beta5 を使用した。以下の手順でインストールできるが、二つのバージョンを同時にインストールする場合は、使用するディレクトリが競合しないように、適切に変更する必要がある。

root ユーザで、PostgreSQL の所有者となる Linux ユーザとして、postgres ユーザを作成する。

```
# useradd postgres
```

PostgreSQL のソースコードアーカイブを展開するディレクトリと、PostgreSQL をインストールするディレクトリを作成して、ディレクトリの所有者を postgres ユーザにする。

(a) PostgreSQL7.4.6 の場合

```
# mkdir /usr/local/src/postgresql-7.4.6
# chown postgresql /usr/local/src/postgresql-7.4.6
# mkdir /usr/local/pgsql
# chown postgresql /usr/local/pgsql
```

(b) PostgreSQL8.0beta5 の場合

```
# mkdir /usr/local/src/postgresql-8.0.0beta5
# chown postgresql /usr/local/src/postgresql-8.0.0beta5
# mkdir /usr/local/pgsql
# chown postgresql /usr/local/pgsql
```

pgsql ユーザで、PostgreSQL のソースコードアーカイブを展開し、展開したディレクトリに移動する。PostgreSQL のソースコードアーカイブは、/tmp ディレクトリにあるものとする。

(a) PostgreSQL7.4.6 の場合

```
# su - postgresql
$ cd /usr/local/src
$ tar xzf /tmp/postgresql-7.4.6.tar.gz
$ cd postgresql-7.4.6
```

(b) PostgreSQL8.0.0beta5 の場合

```
# su - postgresql
$ cd /usr/local/src
$ tar xzf /tmp/postgresql-8.0.0beta5.tar.gz
$ cd postgresql-8.0.0beta5
```

PostgreSQL8.0.0beta5 の場合は、pg\_dump コマンドにインデックスに対するテーブルスペースを正しくバックアップできないというバグがあり、そのパッチを作成した。以下の手順でパッチを適用して修正する。

```
$ cd src/bin/pg_dump
$ patch -p0 < /tmp/pg_dump.patch
```

```
$ cd ../../..
```

PostgreSQL8.0.0 正式版にも同じバグがあり、それは PostgreSQL 開発コミュニティにバグレポートを報告した。

<http://archives.postgresql.org/pgsql-bugs/2005-01/msg00211.php>

展開した PostgreSQL のソースコードをコンパイルし、インストールする。

(a) PostgreSQL7.4.6 の場合

```
$ ./configure --prefix=/usr/local/pgsql
$ make all
$ make install
```

(b) PostgreSQL8.0beta5 の場合

```
$ ./configure --prefix=/usr/local/pgsql
$ make all
$ make install
```

PostgreSQL のインストールが完了したので、コマンドサーチパスに PostgreSQL の bin ディレクトリを追加する。また、PGDATA 環境変数にデータベースクラスタのディレクトリを指定する。

```
$ export PATH=/usr/local/pgsql/bin:$PATH
$ export PGDATA=/dbt3_0/pgsql
```

PostgreSQL のデータベースクラスタを、データベースクラスタ専用のディスクをマウントしたディレクトリに作成する。これにより、処理性能の向上が見込める。また、デフォルトの文字エンコーディングは、日本で一般的に使用される EUC\_JP を指定する。今回の検証では、データベースクラスタを/dbt3\_0 ディレクトリ以下に作成した。

```
$ mkdir /dbt3_0/pgsql
$ initdb --encoding=EUC_JP --no-locale
```

PostgreSQL のデータベースクラスタ内にある WAL のディレクトリを、WAL 専用のディスクをマウントしたディレクトリに移動して、シンボリックリンクを設定する。これにより、処理性能の向上が見込める。

今回の検証では、WAL を/db\_xlog ディレクトリに移動した。

```
$ cd /dbt3_0/pgsql
$ mv pg_xlog /db_xlog
$ ln -s /db_xlog/pg_xlog .
```

### 7.2.2.3 *pgbench* のインストール

pgsql ユーザになり、PostgreSQL のソースコードを展開したディレクトリ以下にある、*pgbench* のディレクトリに移動する。

(a) PostgreSQL7.4.6 の場合

```
# su - pgsq1
$ cd /usr/local/src/postgresql-7.4.6/contrib/pgbench
```

(b) PostgreSQL8.0.0beta5 の場合

```
# su - pgsq1
$ cd /usr/local/src/postgresql-8.0.0beta5/contrib/pgbench
```

PostgreSQL8.0 のテーブルスペース機能を使用して評価を行う場合は、*pgbench* のソースコードを変更する必要がある。今回の評価では、その変更を行うパッチを作成した。以下の手順でパッチを適用することができる。

```
$ patch -p0 < /tmp/pgbench8.0-tblspace.patch
```

このパッチの詳細は、「7.3.2.1 テーブルスペースの設定」を参照のこと。

必要なパッチを適用したら、*pgbench* のコンパイルとインストールを行う。

```
$ make
$ make install
```

以上で、*pgbench* のインストールは完了である。

## 7.2.3 パラメータの設定

### 7.2.3.1 PostgreSQL パラメータ(ログ)

PostgreSQL の設定ファイル ( /dbt3\_0/pgsql/postgresql.conf ) に、ログの記録時にその日時も出力するように設定する。

(a) PostgreSQL7.4.6 の場合

```
log_timestamp = true
```

(b) PostgreSQL8.0 の場合

```
log_line_prefix = '<%'>
```

### 7.2.3.2 PostgreSQL パラメータ(PITR)

PostgreSQL8.0 でクラッシュリカバリを使用するためには、PostgreSQL の設定ファイルで Point In Time Recovery(PITR)の設定を行う必要がある。PITR の設定を行うと、全体的な性能が低下する傾向があるので、PITR を使用した評価を行う場合にのみ、以下の設定を行うこと。

PostgreSQL の設定ファイル ( /dbt3\_0/data/postgresql.conf ) をテキストエディタで編集する。

```
#archive_command = '' # command to use to archive a logfile segment
```

上記の行を編集して、アーカイブコマンドを設定する。今回の評価では、以下のよう  
にアーカイブログ専用のディレクトリにコピーするように設定した。

```
archive_command = ' cp %p /dbt3_3/archivedir/%f'
```

上記で設定したアーカイブログのディレクトリを、所有者を postgres ユーザとして作成してから、PostgreSQL を起動する。

```
$ mkdir /dbt3_3/archivedir
```

## 7.3 評価手順

### 7.3.1 前提条件

#### 7.3.1.1 PostgreSQL の起動

評価を行う前に、pgsql ユーザで PostgreSQL を起動しておく必要がある。

```
$ pg_ctl -w start
```

### 7.3.2 評価環境

#### 7.3.2.1 テーブルスペースの設定

PostgreSQL8.0 では、データベースクラスタを複数のハードディスクに割り当てる、テーブルスペース機能を使用することができる。pgbench で、テーブルスペースを使用するためには、pgbench のソースコードの変更が必要である。

今回の評価では、pgbench で作成する accounts テーブルと、そのプライマリキーであるインデックスの二つを、テーブルスペース機能で以下のように別々のハードディスクに割り当てた。

/dbt3_0	標準のデータベースクラスタ
/dbt3_1/pgtbls	accounts テーブルのインデックス (プライマリキー)
/dbt3_2/pgtbls	accounts テーブル

この変更を行うパッチを作成した。パッチの適用方法は、「7.2.2.3 pgbench のインストール」で説明した。

#### 7.3.2.2 PITR の設定

PostgreSQL8.0 でクラッシュリカバリを使用するためには、PostgreSQL の設定ファイルで Point In Time Recovery(PITR)の設定を行う必要がある。

今回の評価では、アーカイブログを格納する専用のハードディスクを使用した。

/dbt3_3	アーカイブログ (8.0 のクラッシュリカバリ用)
---------	---------------------------

設定方法は、「7.2.3.2 PostgreSQL パラメータ (PITR)」で説明した。

### 7.3.3 大量データのロード

以下の手順で実行する。

#### (1) データベースの作成

データをロードする空のデータベースを作成する。既にデータベースが存在する場合には、削除して作成し直す必要がある。

```
$ dropdb dbname  
$ createdb dbname
```

PostgreSQL8.0 の PITR を設定した場合は、アーカイブログを削除する。アーカイブログは増加する一方なので、これまでに実行したトランザクションがログとして記録されている。大量データのロードでのアーカイブログの増加率を測定するには、ここで削除しておく必要がある。

```
$ rm -f /dbt3_3/archivedir/*
```

通常の運用では、データベースクラスタの物理バックアップが成功した場合に、バックアップ前に生成されたアーカイブログを削除するのが良いだろう。しかし、今回の評価では利便性のために、それぞれの測定を始める前の段階でアーカイブログを削除する。

#### (2) データのロード

pgbench を使用して、大量のデータをロードする。<scale>には、スケールファクターの数値を指定する。この際に、time コマンドでロードに要した時間を測定する。ロード時間は、スケールファクターに応じて長くなる。

```
$ time pgbench -i -s <scale> dbname
```

#### (3) ディスク使用量の取得

データベースクラスタのディスク使用量を取得する。

```
$ du -m -c /dbt3_0
```

PostgreSQL8.0 のテーブルスペース機能を使用する場合は、テーブルスペースのディレクトリを含めたディスク使用量を取得する。

```
$ du -m -c /dbt3_0 /dbt3_1 /dbt3_2
```

PostgreSQL8.0 の PITR を設定した場合は、アーカイブログのディスク使用量も取得する。

```
$ du -m -c /dbt3_3
```

以上で、指定したスケールファクターにおける、以下の情報を測定することが出来る。

- ・ ロード時間
- ・ データベースクラスタのディスク使用量(WAL は除く、テーブルスペースを含む)
- ・ アーカイブログの増加量 ( PostgreSQL8.0 で PITR を設定した場合 )

### 7.3.4 バックアップとリカバリ

以下の手順で実行する。

#### (1) バックアップを実行する

PostgreSQL8.0 の PITR を設定した場合は、アーカイブログを削除しておく。

```
$ rm -f /dbt3_3/archivedir/*
```

pg\_dump を使用して、「7.3.3 大量データのロード」でロードしたデータベースの内容をバックアップする。データベースのサイズが大きいため、1GB ごとにファイルを分けてバックアップを行う。その際に、time コマンドを使用して、バックアップに要した時間を測定する。

今回の評価では、バックアップ時間の短縮のために、専用のディスクを使用したパーティションに対してバックアップを行った。

```
$ time pg_dump -b -F c dbname | split -b 1000m - /dbt3_4/dbname.dump.
```

PostgreSQL8.0 の PITR を設定した場合は、アーカイブログのディスク使用量も取得する。

```
$ du -m -c /dbt3_3
```

バックアップファイルのサイズを取得する。

```
$ du -m -c /dbt3_4/dbname.dump.*
```

(2) リカバリを実行する

バックアップファイルを元にリカバリを行うために、データベースを作り直す。

```
$ dropdb dbname  
$ createdb dbname
```

PostgreSQL8.0 の PITR を設定した場合は、アーカイブログを削除しておく。

```
$ rm -f /dbt3_3/archivedir/*
```

pg\_restore を使用して、バックアップファイルを元にリカバリを実行する。その際に、time コマンドを使用して、リカバリに要した時間を測定する。

```
$ time cat /dbt3_4/dbname.dump.* | pg_restore -d dbname
```

PostgreSQL8.0 の PITR を設定した場合は、アーカイブログのディスク使用量も取得する。

```
$ du -m -c /dbt3_3
```

以上で、指定したスケールファクターにおける、以下の情報を測定することが出来る。

- ・ バックアップ時間
- ・ バックアップファイルからのリカバリ時間
- ・ アーカイブログの増加量 ( PostgreSQL8.0 で PITR を設定した場合 )

### 7.3.5 インデックス再構築

以下の手順で実行する。

(1) インデックス再構築を実行する

PostgreSQL8.0 の PITR を設定した場合は、アーカイブログを削除しておく。

```
$ rm -f /dbt3_3/archivedir/*
```

「7.3.3 大量データのロード」でロードしたデータベースに対して、reindex を使用して accounts テーブルのインデックス (プライマリー) を再構築する。その際に、time コマンドを使用して、インデックスの再構築に要した時間を測定する。

```
$ time psql -d dbname -c "reindex table accounts force"
```

PostgreSQL8.0 の PITR を設定した場合は、アーカイブログのディスク使用量も取得する。

```
$ du -m -c /dbt3_3
```

以上で、指定したスケールファクターにおける、以下の情報を測定することが出来る。

- ・ インデックス再構築時間
- ・ アーカイブログの増加量 ( PostgreSQL8.0 で PITR を設定した場合 )

なお、インデックス再構築の対象とした accounts の行数は、「7.1.1 テーブル構成」で説明したように、「スケールファクター数 × 100000」である。

### 7.3.6 クラッシュリカバリ (PostgreSQL8.0 のみ)

以下の手順で実行する。

#### (1) アーカイブコマンドの設定

PostgreSQL8.0 でクラッシュリカバリを使用するためには、PostgreSQL の設定ファイルで Point In Time Recovery(PITR)の設定を行う必要がある。設定方法は、「7.2.3.2 PostgreSQL パラメータ ( PITR )」で説明した。

#### (2) データベースの作成

データをロードする空のデータベースを作成する。既にデータベースが存在する場合には、削除して作成し直す必要がある。

```
$ dropdb dbname  
$ createdb dbname
```

また、アーカイブログも削除しておく。

```
$ rm -f /dbt3_3/archivedir/*
```

(3) 物理イメージでバックアップする

psql で pg\_start\_backup() という関数を使い、バックアップの開始を宣言する。引数はこのバックアップを識別するラベルであり、任意の文字列を指定する。

```
$ psql dbname  
dbname=# SELECT pg_start_backup('mybackup');
```

tar コマンドを使って、データベースクラスタを物理イメージでバックアップする。データベースにある程度のデータがあるならば、time コマンドでバックアップに要した時間を取得しても良いのだが、今回の検証では作成したばかりの空のデータベースなので、バックアップは即座に完了する。従って、time コマンドで所要時間を測定する必要は無い。

なお、バックアップ中に、データベースへの更新処理を行っても問題ないが、今回の検証では、検索および更新の処理は行わなかった。

```
$ cd /dbt3_0  
$ tar cfz /dbt3_4/pgsql.bak postgresql
```

tar コマンドのバックアップが終了したら、psql で pg\_stop\_backup() という関数を使い、バックアップの終了を宣言する必要がある。

この関数は、バックアップが終了したらなるべく早く実行するのが良いが、多少は遅れても問題ない。

```
dbname=# SELECT pg_stop_backup();
```

(4) データのロード

pgbench を使用して、大量のデータをロードする。<scale>には、スケールファクターの数値を指定する。

```
$ pgbench -i -s <scale> dbname
```

(5) データベースの停止

以上で、クラッシュリカバリの準備が出来たので、postgres ユーザでデータベースを停止する。

```
$ pg_ctl stop
```

これまでの手順で、データベースを作成した後の任意の時点において、データベースクラスタの物理イメージによるバックアップを取得した。今回の評価では、データベース作成直後に行ったが、ある程度のデータが挿入された時点でも問題ない。

その際には、`pg_start_backup()`と`pg_stop_backup()`の二つの関数を実行する間で、物理イメージでバックアップを行う必要があった。そして、バックアップを取得した後に`pgbench`によりデータを挿入してから、データベースを停止した。

(6) トランザクションログの待避

後ほど、データベースクラスタを削除することにより破壊するので、その前にデータベースクラスタ内にあるトランザクションログを待避しておく必要がある。これは、クラッシュリカバリを行う際に、アーカイブログに記録された以降の直近のデータとして、トランザクションログが必要だからである。

しかし、今回の評価では、トランザクションログをデータベースクラスタとは別のディレクトリに設定してあるので、待避する必要は無い。

(7) データベースクラスタの破壊

データベースクラスタを削除する。

```
$ cd /dbt3_0
$ rm -rf pgsql
```

(8) 物理バックアップの復元

(3)でバックアップした物理バックアップから、データベースクラスタを復元する。

```
$ tar xzf /dbt3_4/pgsql.bak
```

このままでは、(4)でロードしたデータが含まれていない。これより、このデータを復元していく。

(9) トランザクションログの復元

(6)で待避したトランザクションログを、(8)で復元したデータベースクラスタ内に復元する。

今回の評価では、トランザクションログはデータベースクラスタとは別のディレクトリにあるので、復元の必要は無い。

(10) `recover.conf` の作成

クラッシュリカバリには、`recover.conf` ファイルが必要なので、雛形のファイルをコピーする。

```
$ cd /dbt3_0/pgsql
$ cp /usr/local/pgsql/share/recovery.conf.sample recovery.conf
```

コピーした recovery.conf ファイルをテキストエディタで編集する。

```
#restore_command = 'cp /mnt/server/archivedir/%f %p'
```

上記の行を、以下のように変更する。

```
restore_command = 'cp /dbt3_3/archivedir/%f %p'
```

ここまでで、クラッシュリカバリの設定は完了である。

#### (11) クラッシュリカバリの実行

PostgreSQL を起動すれば、クラッシュリカバリが実行される。

```
$ pg_ctl start
```

起動時にクラッシュリカバリの実行ログが画面に表示される。クラッシュリカバリの開始時は、ログに以下のように記録される。

```
<2005-02-09 21:27:17 JST>LOG: starting archive recovery
```

クラッシュリカバリの終了時は、ログに以下のように記録される。

```
<2005-02-09 21:46:18 JST>LOG: archive recovery complete
```

この二つの時間の差が、クラッシュリカバリに要した所要時間である。この際に「time pg\_ctl -w start」で所要時間を測定することは出来ない。-w オプションは1分間でタイムアウトするためである。

なお、クラッシュリカバリが正常に実行された場合は、recovery.conf ファイルは、recovery.done にファイル名が変更される。

- ・ 以上で、指定したスケールファクターにおける、クラッシュリカバリの所要時間を測定することができる。

## 7.4 測定結果と考察

大規模データベース環境における性能評価として、以下について評価した結果を報告する。

- ・ バックアップとリカバリに要する時間の性能評価
- ・ 大容量データのロードに要する時間の性能評価
- ・ インデックスの再構築に要する時間の性能評価
- ・ PostgreSQL8.0 の Point In Time Recovery(PITR)を使用し、クラッシュリカバリに要する時間の性能評価

PostgreSQL のバージョンは、今回の評価を開始した時点での最新バージョンである、7.4.6 と 8.0.0beta5 について評価した。

## 7.4.1 測定結果と考察

### 7.4.1.1 スケールファクターを変えて測定

#### 7.4.1.1.1 目的

PostgreSQL7.4.6 と 8.0.0beta5 について、スケールファクターを変えたパターンで、大規模データベースにおける性能を測定する。

#### 7.4.1.1.2 結果

PostgreSQL7.4.6 での測定結果を表 7.4-1 に、8.0.0beta5 での測定結果を表 7.4-2 に示す。

表 7.4-1 PostgreSQL7.4.6

		スケールファクター			
		1000	2000	3000	4000
大規模データのロード	時間	0:50:58	1:42:37	2:34:06	3:26:23
	サイズ	14547M	29080M	43613M	58145M
バックアップ	時間	0:19:45	0:40:27	1:00:10	1:23:45
	サイズ	273M	541M	810M	1079M
リカバリ	時間	0:58:10	1:57:27	2:56:25	3:55:06
インデックス再構築	時間	0:11:51	0:23:47	0:35:37	(a)

表 7.4-2 PostgreSQL8.0.0beta5 (PITR なし、テーブルスペースなし)

		スケールファクター			
		1000	2000	3000	4000
大規模データのロード	時間	0:47:42	1:56:51	2:57:26	3:48:13
	サイズ	14761M	29508M	44255M	52152M
バックアップ	時間	0:19:58	0:40:17	1:00:20	1:21:36
	サイズ	273M	541M	810M	1079M
リカバリ	時間	0:57:20	1:55:28	2:54:11	(b)
インデックス再構築	時間	0:10:49	0:21:38	0:32:30	(a)

(a)、(b)ともに、インデックスの構築時にディスク容量不足の「No space left on drive」エラーが発生した。

(b)では、ディスク使用率 90%でエラーが発生した。

(1) 大規模データのロード

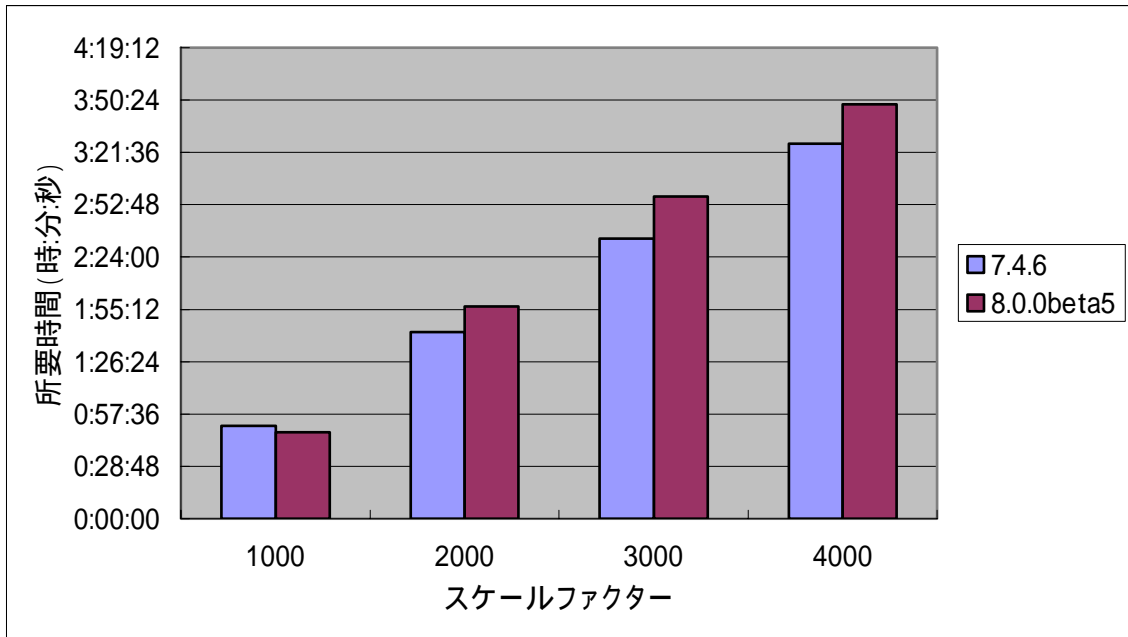


図 7.4-1 大規模データのロード時間

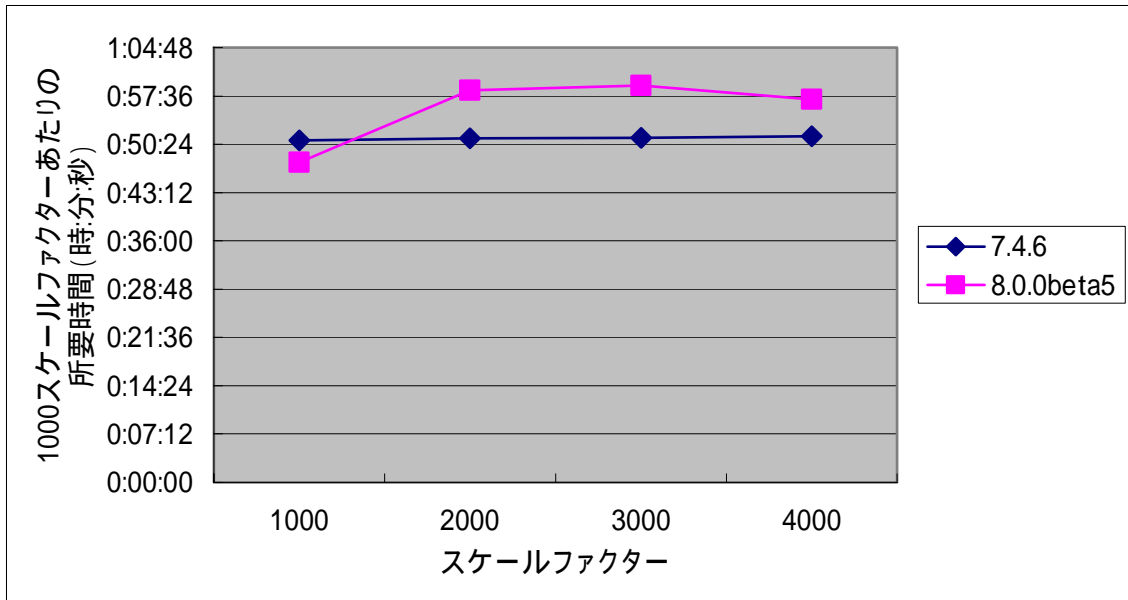


図 7.4-2 1000 スケールファクターあたりのロード時間

7.4.6 では規模あたりのロード時間は、ほぼ一定であるが、8.0.0beta5 では変動があることが確認された。今回の評価では実施しなかったが、スケールファクター1000 未満では 8.0.0beta5 の方がロード時間が短い可能性はある。

(2) バックアップ

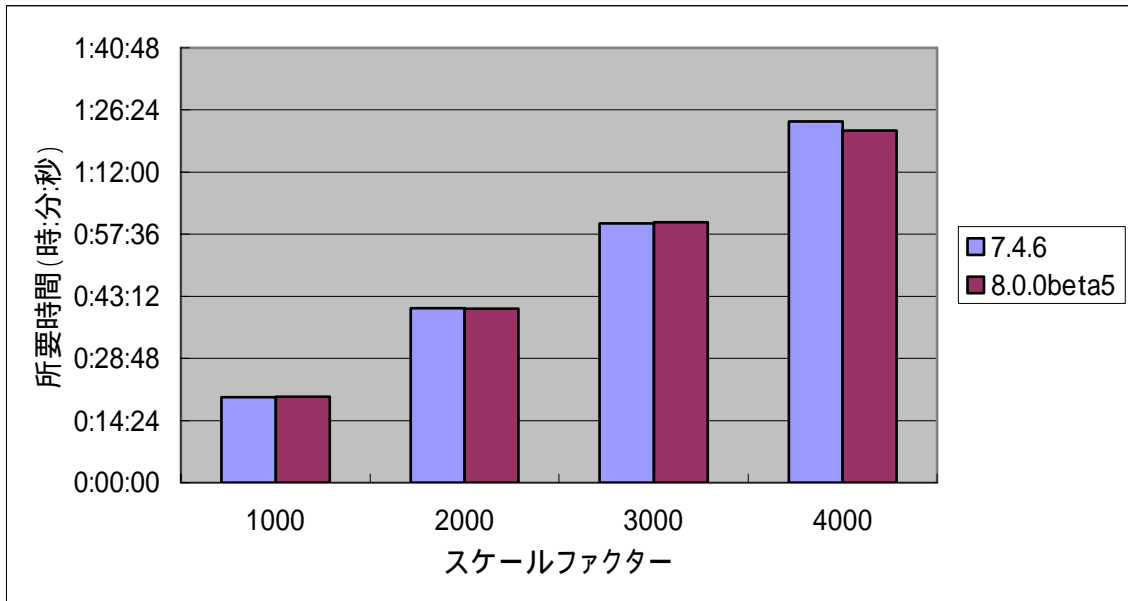


図 7.4-3 バックアップ時間

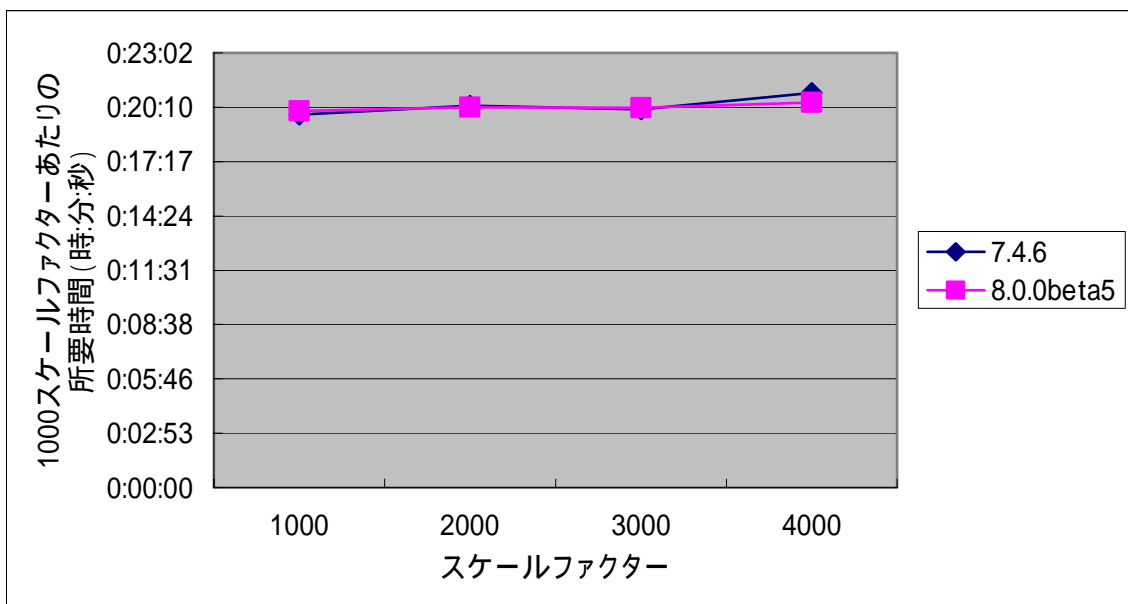


図 7.4-4 1000 スケールファクターあたりのバックアップ時間

8.0.0beta5 では、規模あたりのバックアップ時間は一定であった。7.4.6 では、若干の増加傾向が見られた。

(3) バックアップファイルからのリカバリ

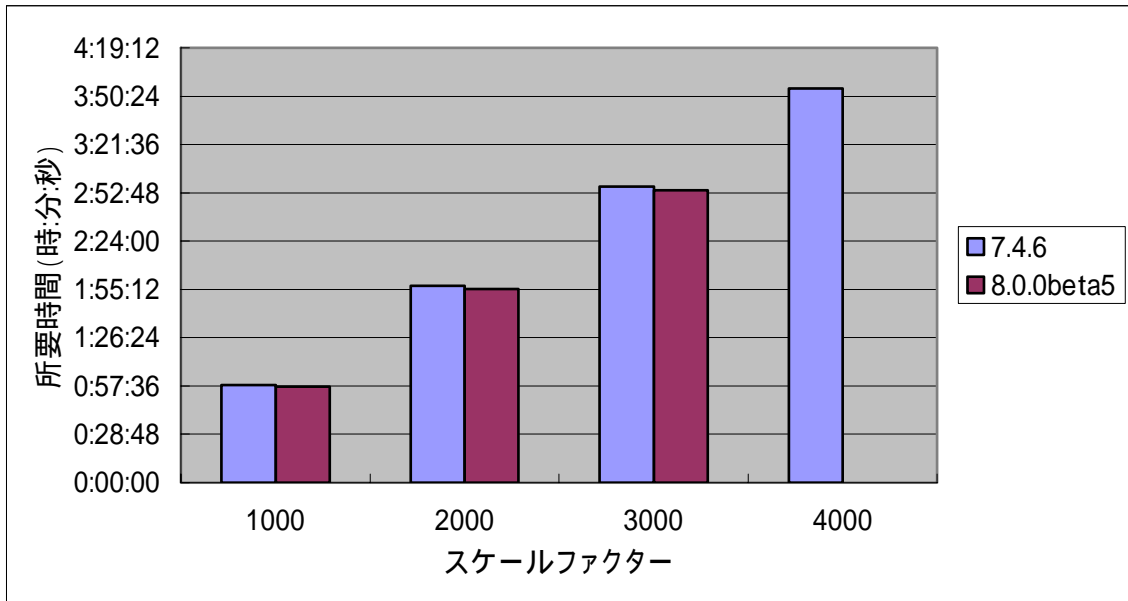


図 7.4-5 リカバリ時間

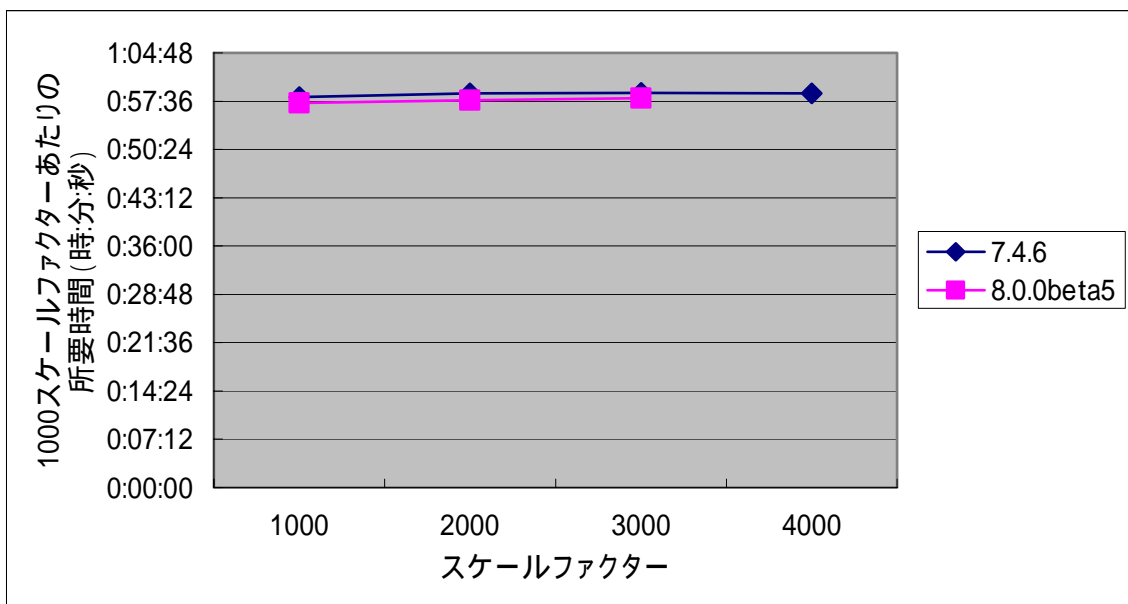


図 7.4-6 1000 スケールファクターあたりのリカバリ時間

規模あたりのリカバリ時間は、ほぼ一定であった。

スケールファクター4000での8.0.0beta5の測定は、データをロードしてから、インデックスを構築中にディスク容量不足のエラーとなった。エラー直前のディスク使用率は90%で、エラー発生後はインデックスがロールバックされて81%となった。今回の評価では、その原因を突き止めるまでには至らなかった。

#### (4) インデックス再構築

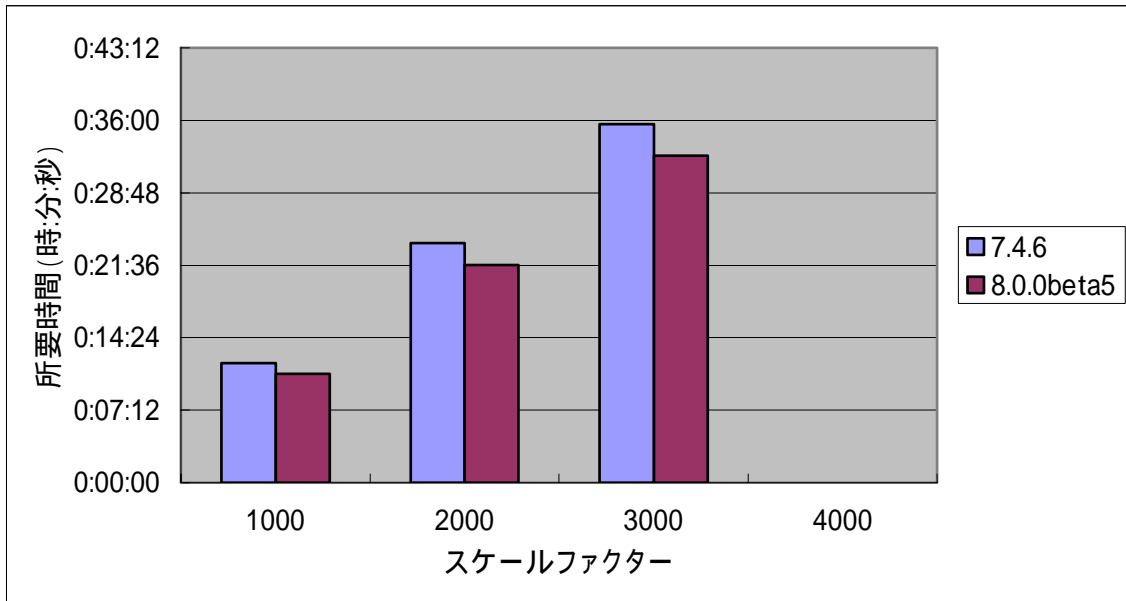


図 7.4-7 インデックス再構築時間

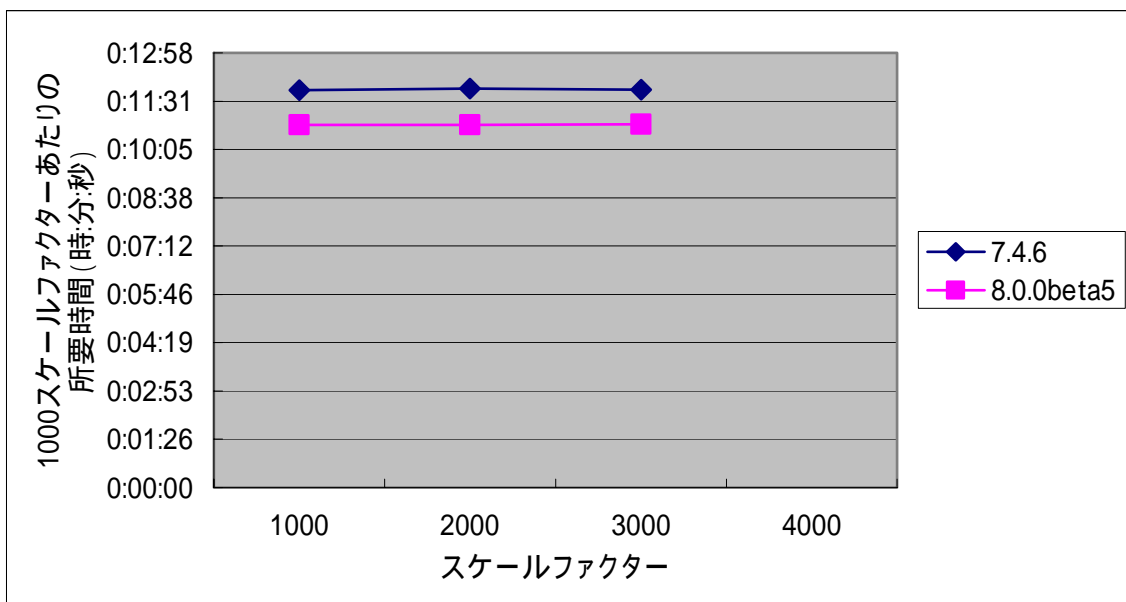


図 7.4-8 1000 スケールファクターあたりのインデックス再構築時間

8.0.0beta5の方が7.4.6よりも約10%ほど性能が良いという結果となった。規模あたりのインデックス再構築時間は一定であった。

スケールファクター4000では、ディスク容量不足となって実行できなかった。

### 7.4.1.1.3 考察

PostgreSQL7.4.6 と 8.0.0beta5 について、スケールファクターを変えたパターンで、大規模データベースにおける性能を比較した結果を表 7.4-3 にまとめる。

表 7.4-3 スケールファクターを変えて測定

大規模データのロード	7.4.6 は、規模あたりの性能はほぼ一定。 8.0.0beta は、規模あたりの性能は変動し、スケールファクター1000 では 7.4.6 よりも速いが、2000/3000/4000 では 7.4.6 よりも遅くなった。
バックアップ	7.4.6 と 8.0.0beta5 はほとんど同じ性能。 規模あたりの性能は、8.0.0 は一定だが、7.4.6 では若干の増加傾向。
リカバリ	7.4.6 と 8.0.0beta5 はほとんど同じ性能。 規模あたりの性能も一定。 8.0.0beta5 のスケールファクター4000 では、インデックス構築中にディスク容量不足のエラーとなったが、7.4.6 では問題なかった。
インデックス再構築	8.0.0beta5 が 7.4.6 よりも約 10%ほど性能が良い。 規模あたりの性能は一定。

ロードは、7.4.6 は規模あたりの性能はほぼ一定だが、8.0.0beta5 は規模により変動した。ロードは pgbench で行ったが、これはファイルからロードするのではなく、pgbench がデータを動的に生成してロードする。この点が、このような結果となった理由の一つと考えられるが、今回の評価では原因を突き止めるまでには至らなかった。

バックアップ、リカバリ、インデックス再構築は、規模あたりの性能は、ほぼ一定であった。インデックス再構築では、8.0.0beta5 が 7.4.6 よりも約 10%ほど性能が良かった。

8.0.0beta5 でのリカバリでは、スケールファクター4000 において、インデックス構築中にディスク容量不足のエラーが発生したが、7.4.6 では問題なく実行できた。PostgreSQL8.0.0beta5 のソースコードを調査したが、今回の評価では原因となる箇所を突き止めるまでには至らなかった。

pgbench によるロードとリカバリでは、8.0.0beta5 で予想外の結果となったが、それ以外の測定では 7.4.6 と同じかそれ以上の性能となった。

#### 7.4.1.2 テーブルスペースでの測定

##### 7.4.1.2.1 目的

PostgreSQL8.0 のテーブルスペース機能を使用する場合と、使用しない場合とで、大規模データベースにおける性能を測定した。

##### 7.4.1.2.2 結果

測定結果を表 7.4-4 に示す。なお、テーブルスペースを使用しない場合の測定結果は、表 7.4-2 を参照のこと。

表 7.4-4 PostgreSQL8.0.0beta5 (PITR なし、テーブルスペースあり)

		スケールファクター			
		1000	2000	3000	4000
大規模 データの ロード	時間	0:46:29	1:55:18	2:54:02	3:14:58
	テーブルサ イズ	13034M	26068M	39101M	52135M
	インデック スサイズ	1713M	3425M	5137M	6850M
	クラスタサ イズ	15M	16M	17M	17M
バック アップ	時間	0:20:32	0:40:43	1:00:37	1:20:51
	サイズ	273M	541M	810M	1079M
リカバ リ	時間	0:54:36	1:49:19	2:44:32	3:39:33
インデ ックス 再構築	時間	0:10:24	0:20:50	0:31:17	0:42:04

(1) 大規模データのロード

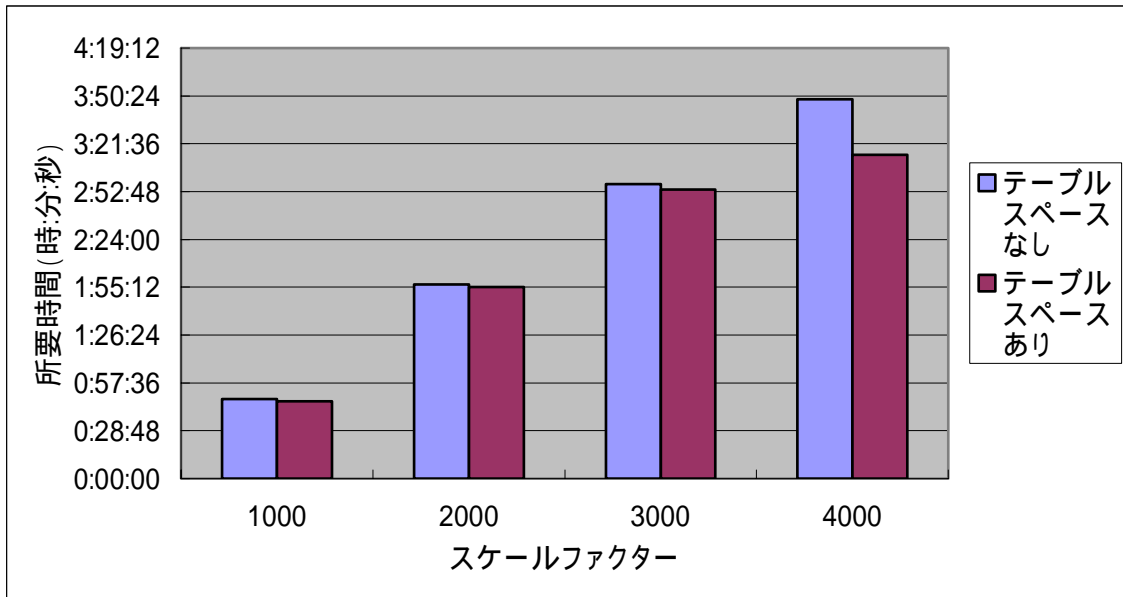


図 7.4-9 大規模データのロード時間

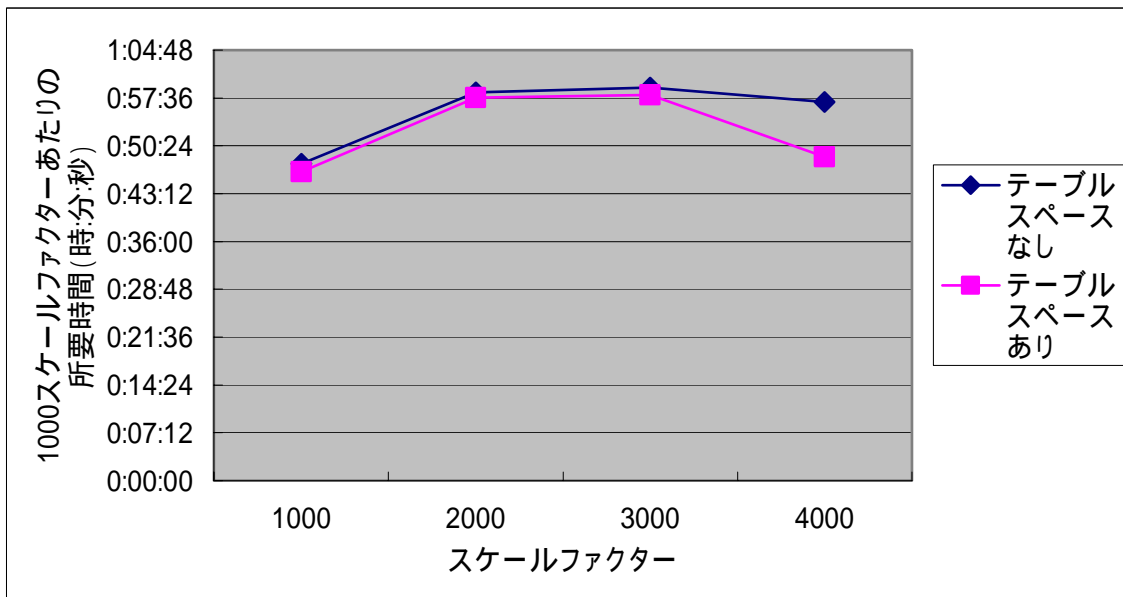


図 7.4-10 1000 スケールファクターあたりのロード時間

全体的にテーブルスペースを使用した方が性能が良いが、スケールファクターが 4000 の場合に特に効果があった。

(2) バックアップ

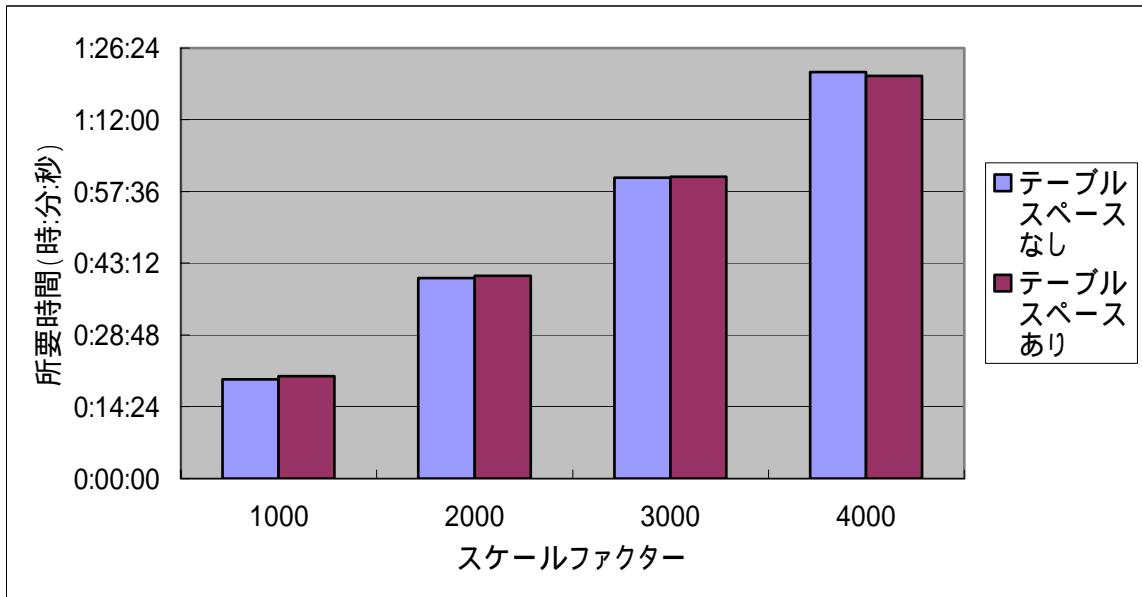


図 7.4-11 バックアップ時間

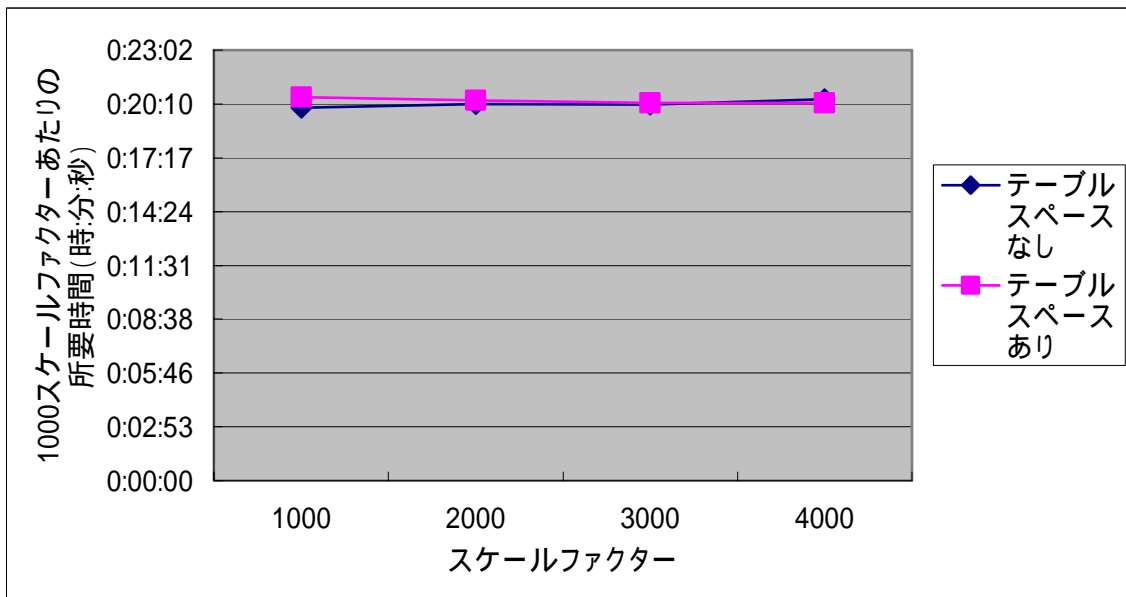


図 7.4-12 1000スケールファクターあたりのバックアップ時間

バックアップについては、テーブルスペースの有無による性能差は、ほとんど無かった。

(3) バックアップファイルからのリカバリ

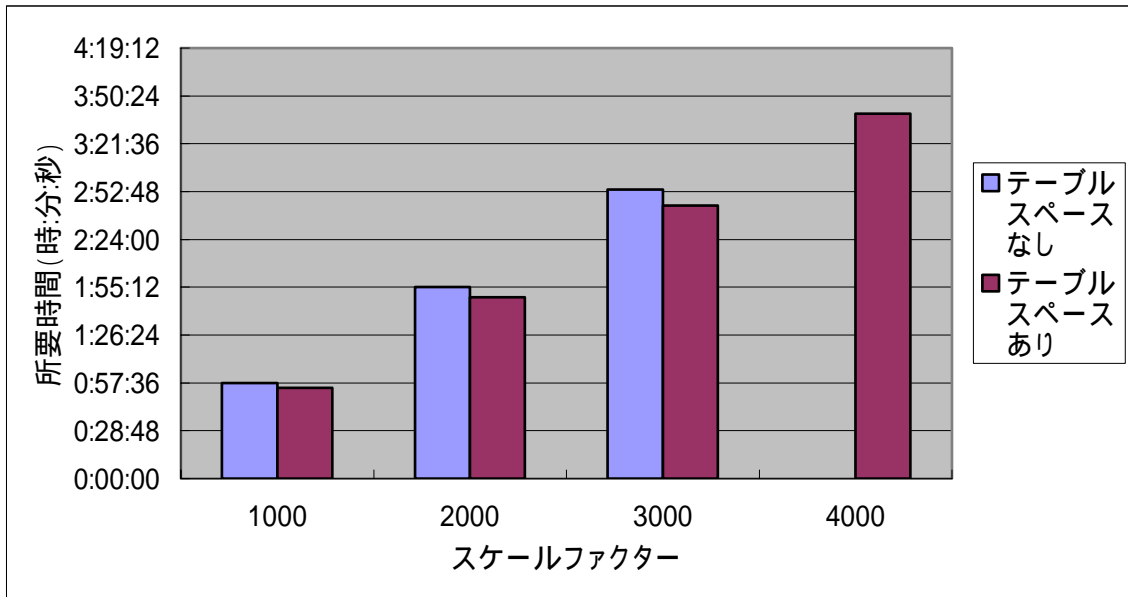


図 7.4-13 リカバリ時間

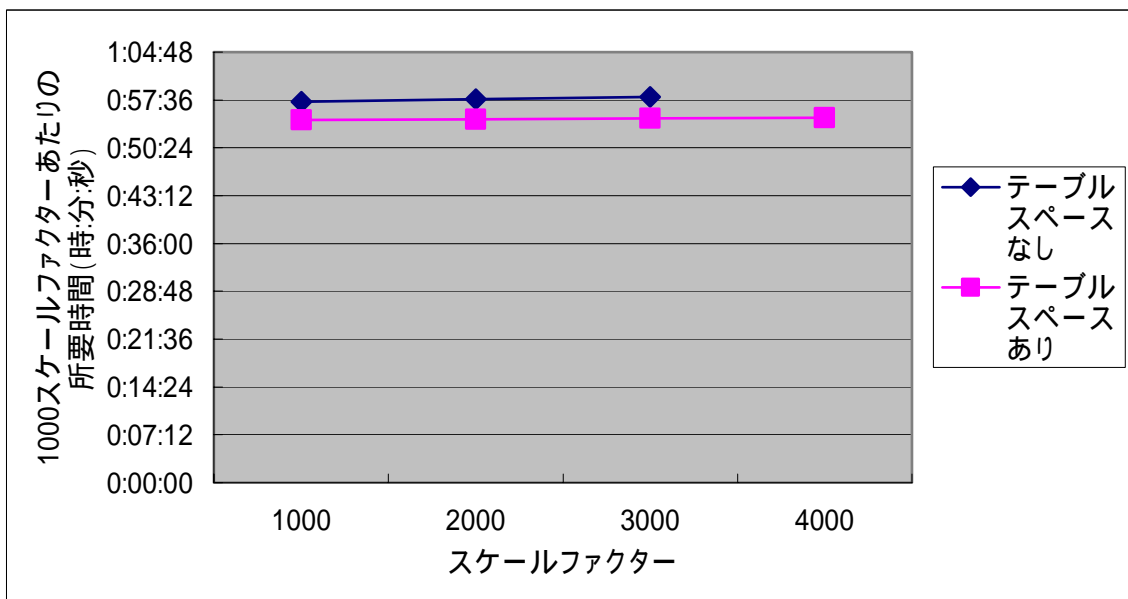


図 7.4-14 1000 スケールファクターあたりのリカバリ時間

テーブルスペースを使用した方が、約 5%ほど性能が良い結果となった。  
 また、テーブルスペース無しではディスク容量不足のエラーとなっていたが、テーブルスペースを使用することで、問題なく実行できた。

(4) インデックス再構築

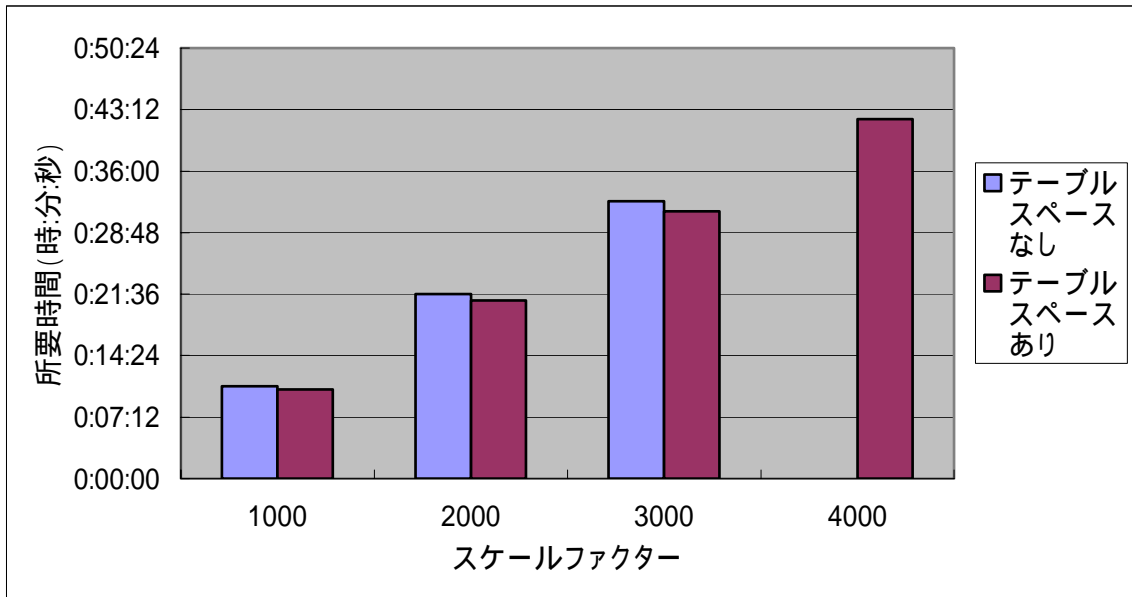


図 7.4-15 インデックス再構築時間

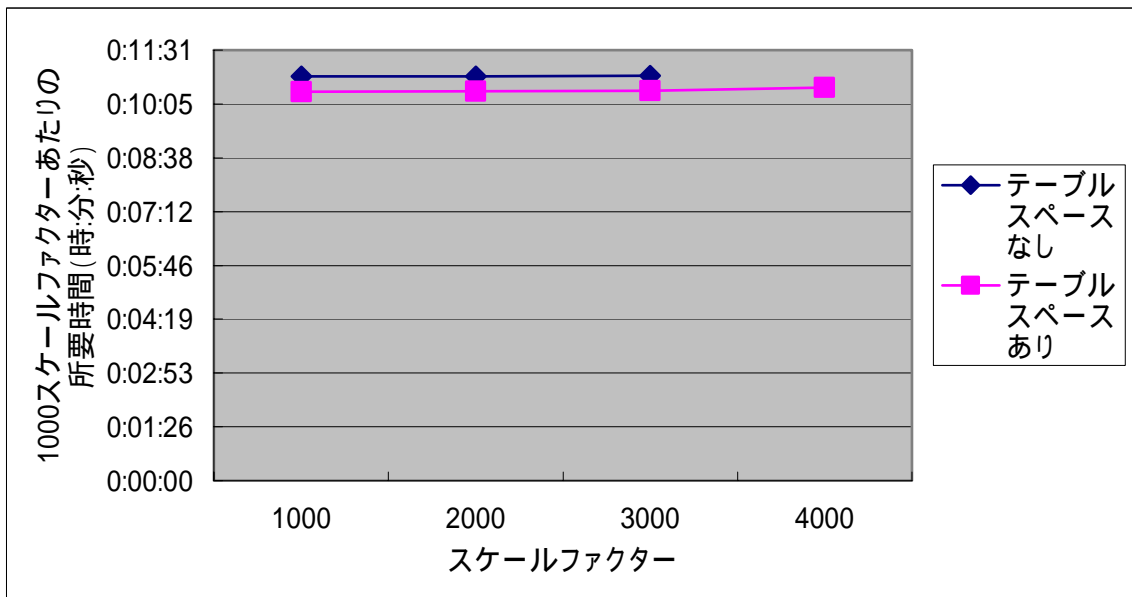


図 7.4-16 1000 スケールファクターあたりのインデックス再構築時間

テーブルスペースを使用した方が、約 5%ほど良い結果となった。  
 また、テーブルスペース無しではディスク容量不足のエラーとなっていたが、テーブルスペースを使用することで、問題なく実行できた。

### 7.4.1.2.3 考察

大量のデータが投入されるテーブルと、そのテーブルのインデックス（プライマリキー）を、テーブルスペース機能を使って、専用のディスクに割り当てた。テーブルスペースを使わない場合と比較した結果を表 7.4-5 にまとめる。

表 7.4-5 テーブルスペースでの測定

大規模データのロード	テーブルスペースを使用すると、スケールファクター1000/2000/3000 で若干だが性能が良く、4000 では特に良くなった。
バックアップ	ほとんど変わらなかった。
リカバリ	テーブルスペースを使用すると、約 5%ほど性能が良くなった。
インデックス再構築	テーブルスペースを使用すると、約 5%ほど性能が良くなった。

上記より、ディスクへの書き込みがある処理で効果があることが確認できた。特に、リカバリとインデックス再構築は、インデックスのファイルを生成するにあたって、元となるテーブルのファイルを読む必要があるが、それらを別のハードディスクに割り当ててあるので、読み込みと書き込みが分散することになる。この効果により、約 5%という性能の向上につながったと考えられる。

バックアップでテーブルスペースの効果がなかったのは、バックアップではテーブルをシーケンシャルに読み取り、バックアップファイルに追記するので、インデックスにテーブルスペースを設定しても、それを読み取ることは無い。また、複数のテーブルを同時に読み取るわけでは無いので、テーブルに対してテーブルスペースを設定しても、読み込みが分散されるわけでは無い。このような理由により、単なるバックアップでは、テーブルスペースの効果がなかったと考えられる。

今回の評価では実施しなかったが、バックアップ中に別のトランザクションがデータベースにアクセスしているような状況では、テーブルスペースの効果があるものと考えられる。

結論として、テーブルスペースの有効性は、ディスクへの書き込みを行うようなトランザクション数が 1 の場合に、約 5%の性能向上が確認できた。トランザクション数を増やした場合については、さらなる検証の必要があるのだが、効果は期待できるものと考えられる。

### 7.4.1.3 クラッシュリカバリの測定

#### 7.4.1.3.1 目的

PostgreSQL8.0のクラッシュリカバリに要する時間と、アーカイブログの量を測定して、その対応関係を求める。

また、クラッシュリカバリを有効にするには、アーカイブログを保存する必要があるため、アーカイブログの保存による性能への影響と、アーカイブログの増加量を測定する。そして、クラッシュリカバリを使用しない場合との比較を行う。

#### 7.4.1.3.2 結果

測定結果を表 7.4-6 に示す。なお、PITR なしの測定結果は、表 7.4-2 を参照のこと。

表 7.4-6 PostgreSQL8.0.0beta5 (PITR あり、テーブルスペースなし)

		スケールファクター			
		1000	2000	3000	4000
大規模データのロード	所要時間	00:53:41	01:47:45	02:42:27	(a)
	データ領域の使用量	14761M	29508M	44254M	
	アーカイブログの増量	16661M	33321M	49998M	
バックアップ	所要時間	00:19:58	00:39:27	01:00:11	
	バックアップファイルの量	273M	541M	810M	
	アーカイブログの増量	0M	0M	0M	
リカバリ	所要時間	01:01:11	02:01:53	03:04:47	
	アーカイブログの増量	16677M	33337M	49998M	
インデックス再構築	所要時間	00:12:09	00:24:27	00:37:27	
	アーカイブログの増量	1715M	3445M	5175M	
クラッシュリカバリ	所要時間	00:19:01	00:41:21	01:00:41	
	アーカイブログの増量	0M	0M	0M	

(a)は、アーカイブログの保存でディスク容量が不足したために測定できなかった。

(1) 大規模データのロード

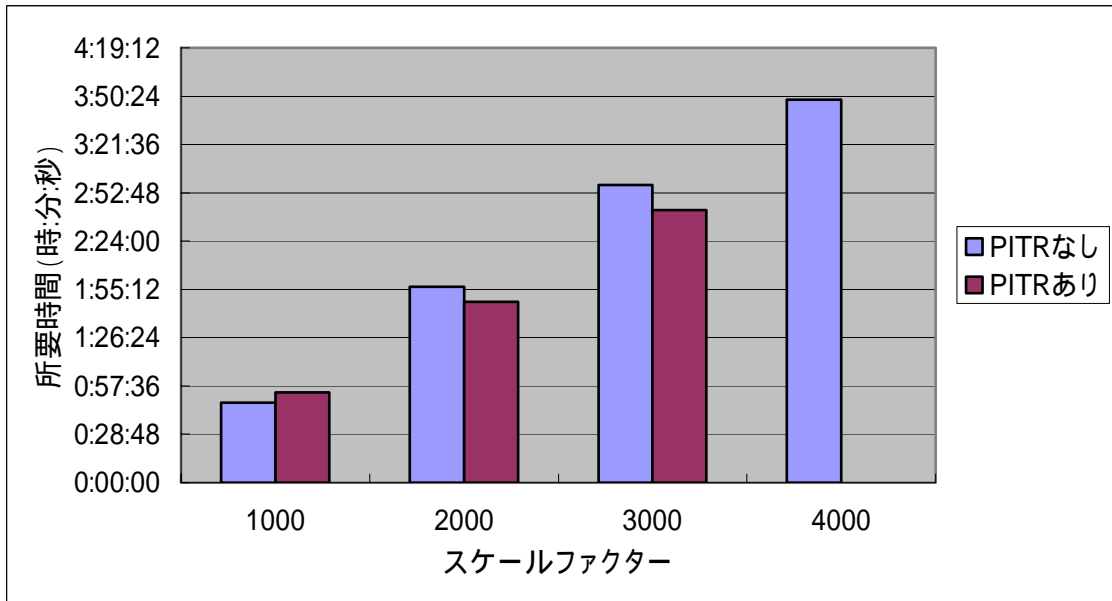


図 7.4-17 ロード時間

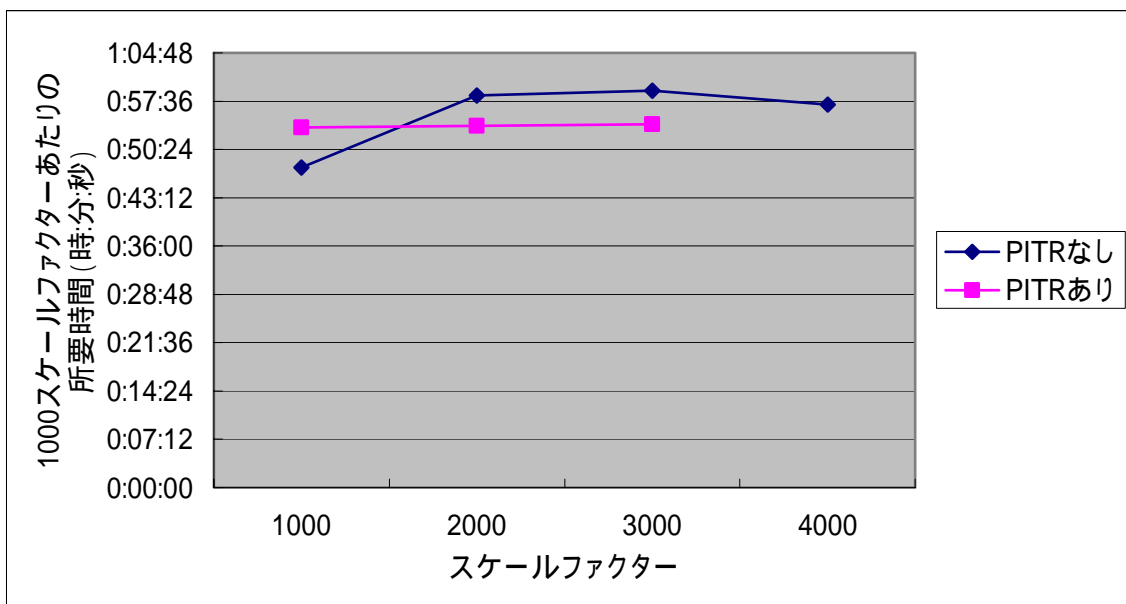


図 7.4-18 1000 スケールファクターあたりのロード時間

PITR を使用した場合の、規模あたりのロード時間は一定であり、PITR 無しとは異なる結果となった。この原因については、今回の評価では特定するには至らなかった。

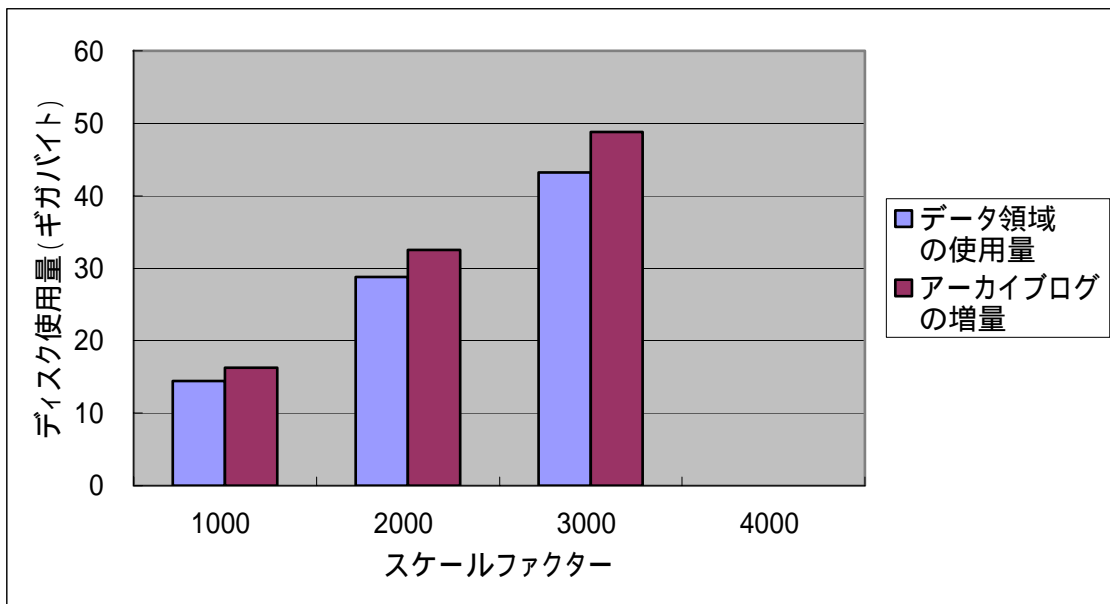


図 7.4-19 アーカイブログの増加量

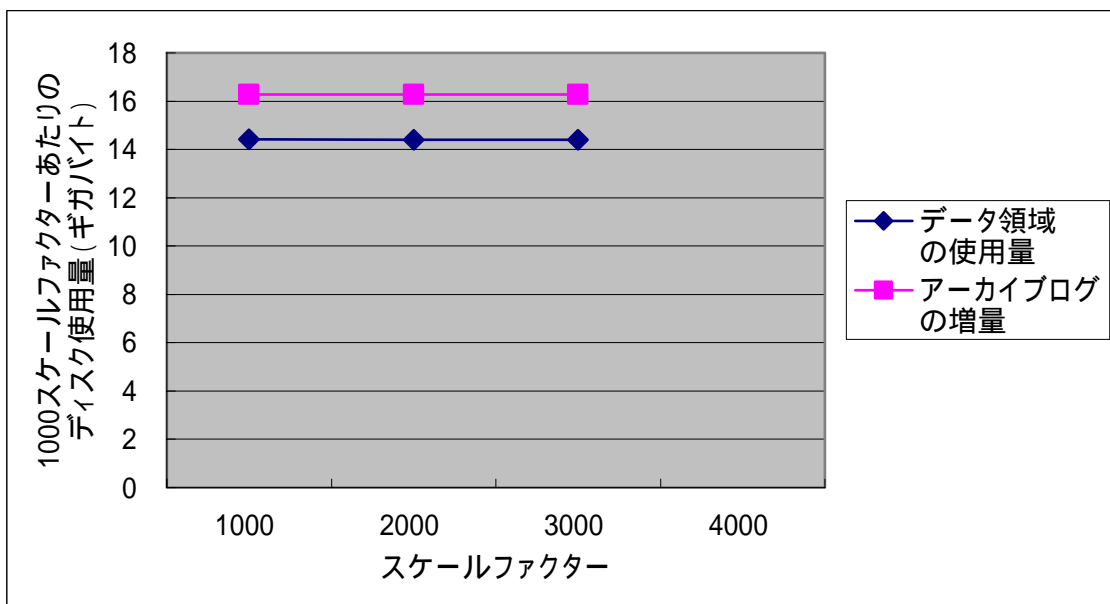


図 7.4-20 1000 スケールファクターあたりのアーカイブログの増加量

大規模データのロード時でのアーカイブログの増加量は、規模に応じて一定しており、それは PostgreSQL のデータベースクラスタの容量の約 1.12 倍になった。

(2) バックアップ

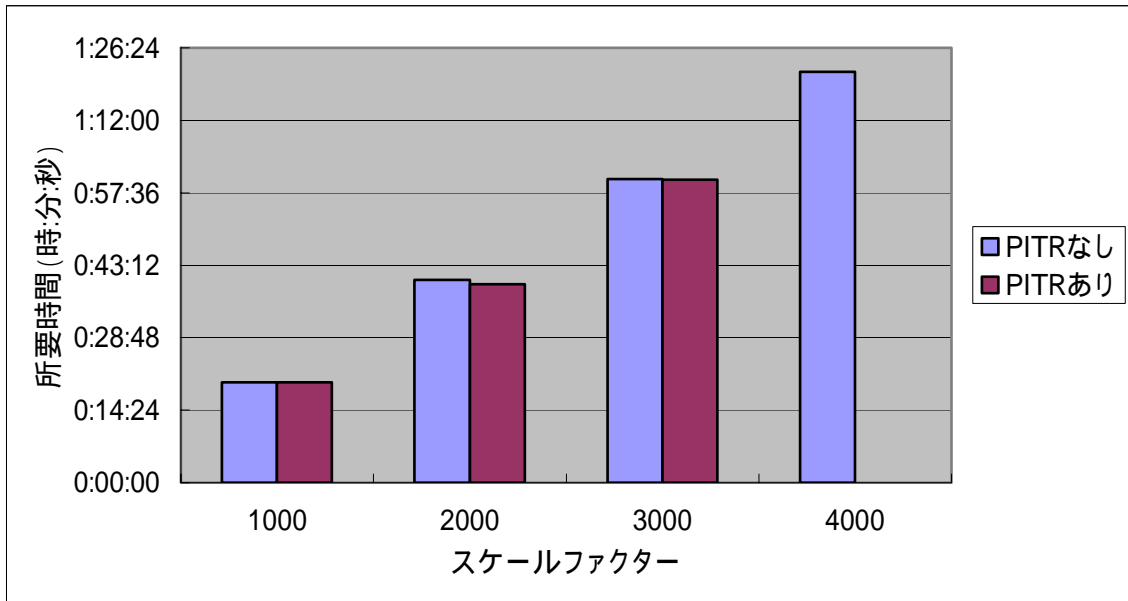


図 7.4-21 バックアップ時間

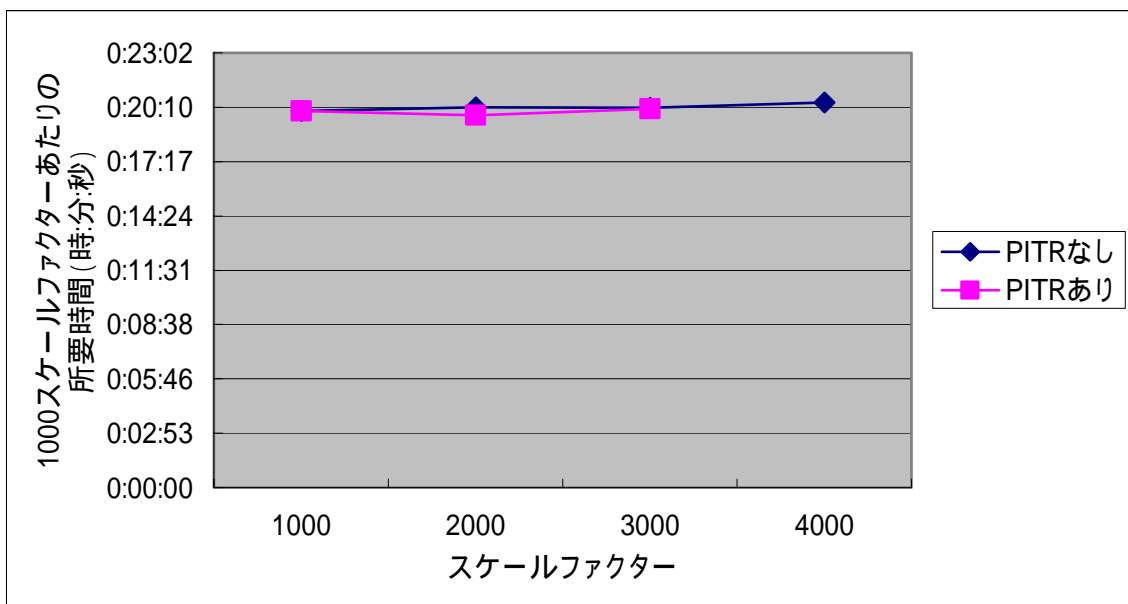


図 7.4-22 1000 スケールファクターあたりのバックアップ時間

PITRの有無による性能の違いは、ほとんど無かった。  
アーカイブログは増加しなかった。

(3) バックアップファイルからのリカバリ

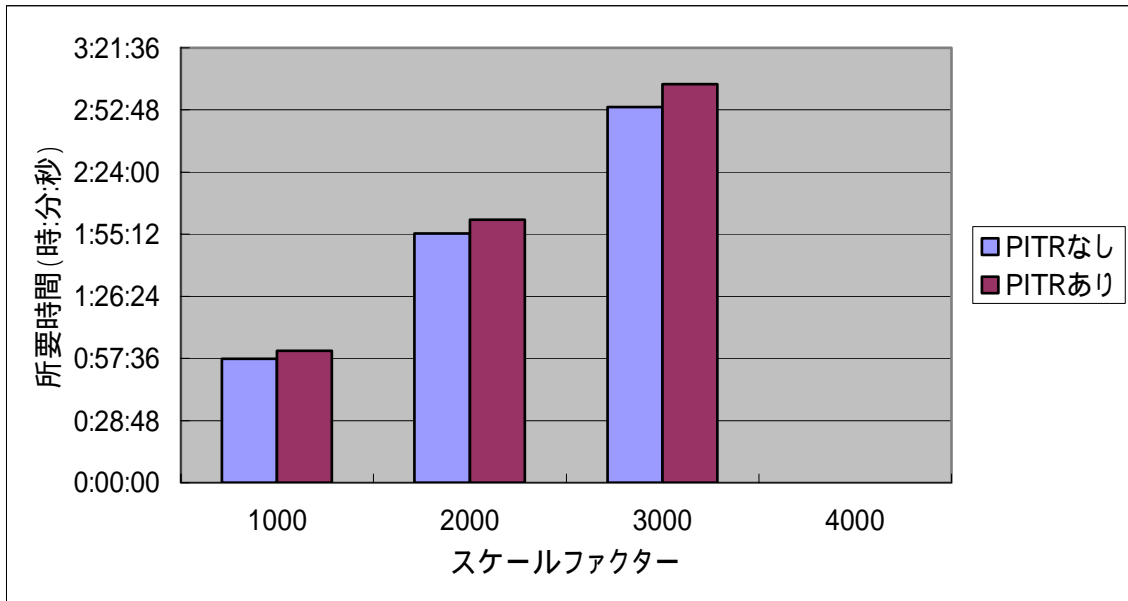


図 7.4-23 リカバリ時間

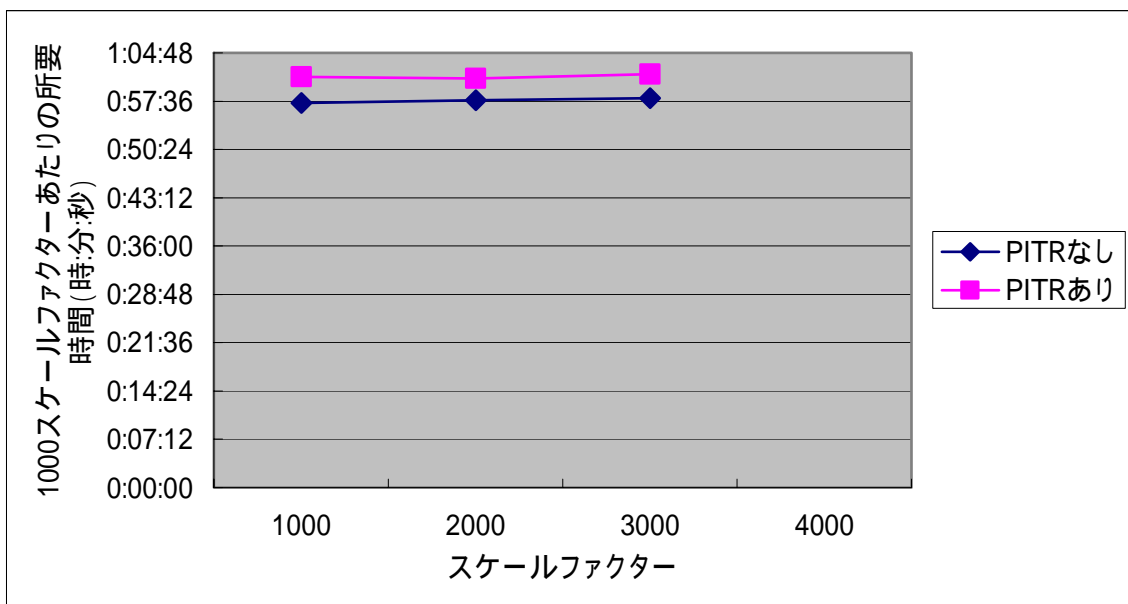


図 7.4-24 1000 スケールファクターあたりのリカバリ時間

PITR を使用すると、約 5%ほどバックアップファイルからのリカバリ時間が長くなった。

アーカイブログの増加量は、ロード時とほとんど同じだけ増加した。

(4) インデックス再構築

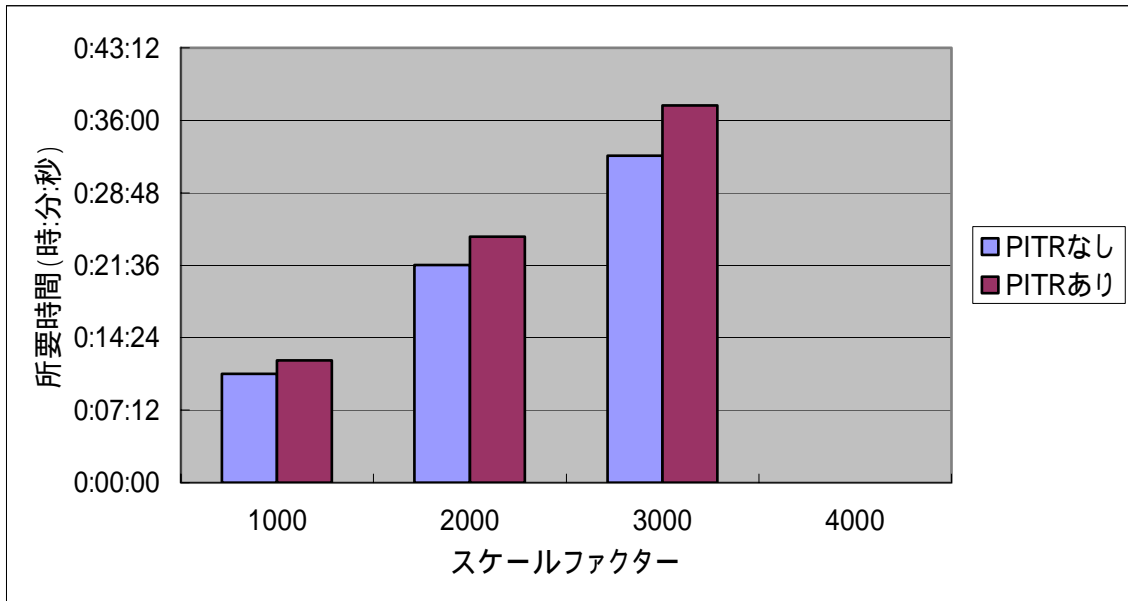


図 7.4-25 インデックス再構築時間

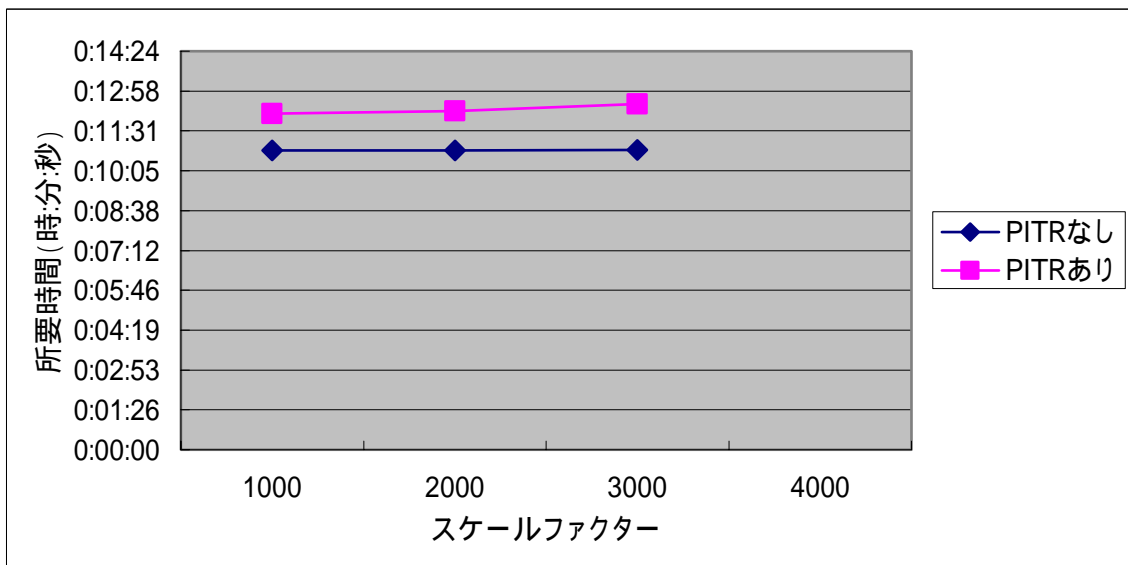


図 7.4-26 1000 スケールファクターあたりのインデックス再構築時間

PITR を使用すると、インデックス再構築時間は約 13%ほど長くなった。また、規模が大きくなると、インデックス再構築時間の増加率は増える傾向にあることが分かった。アーカイブログも少しずつ増加することを確認した。

(5) クラッシュリカバリ

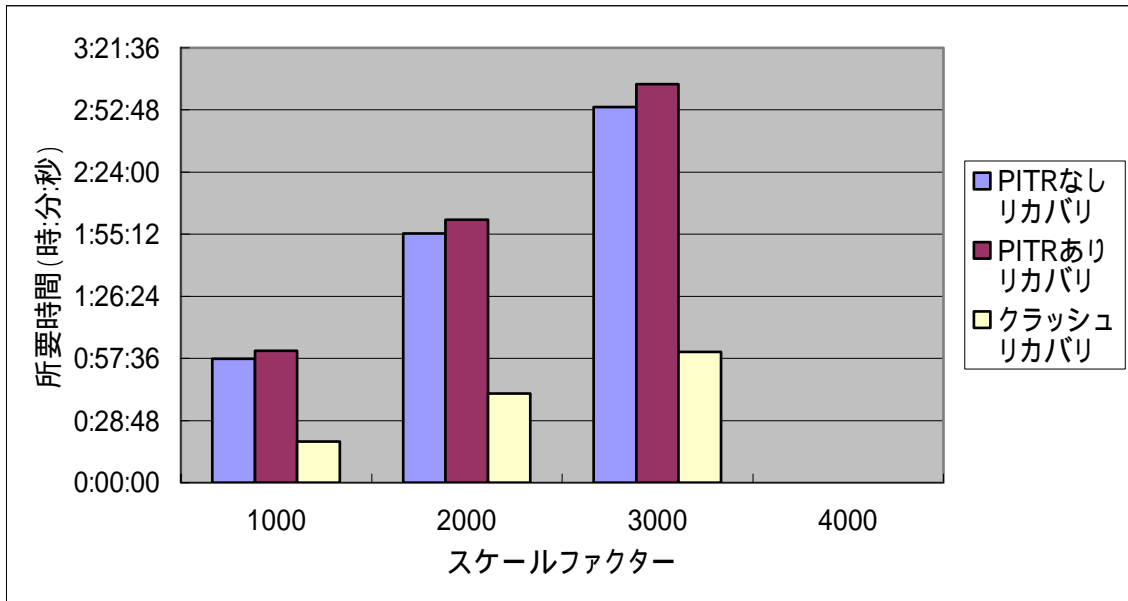


図 7.4-27 クラッシュリカバリ時間

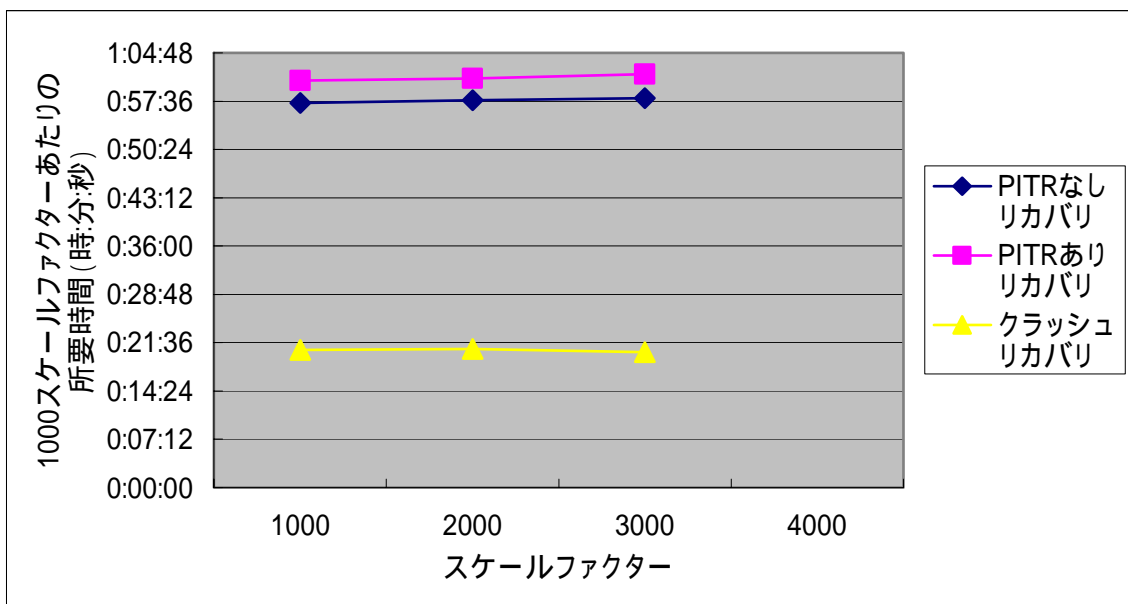


図 7.4-28 1000 スケールファクターあたりのクラッシュリカバリ時間

PITR を使用したクラッシュリカバリは、PITR を使用しないバックアップファイルからのリカバリと比べて、約 1/3 の時間で完了した。

アーカイブログのディスク使用量は変わらなかった。

### 7.4.1.3.3 考察

PITR を使用した状態で、ロード、バックアップ、リカバリ、インデックス再構築、クラッシュリカバリに要する時間を測定した。その結果を表 7.4-7 にまとめる。

表 7.4-7 クラッシュリカバリの測定

大規模データのロード	規模あたりの性能は一定であり、PITR を使用しない場合と異なる結果となった。 アーカイブログの増加量は、データベースクラスタのディスク使用量の約 1.12 倍となった。
バックアップ	PITR の有無による性能差は無かった。 アーカイブログは増加しなかった。
リカバリ	PITR を使用すると、約 5%ほど時間が長くなった。 アーカイブログの増加量は、ロードとほとんど同じだった。
インデックス再構築	PITR を使用すると、約 13%ほど時間が長くなり、規模が増えると所要時間の増加率が増える傾向にあった。 アーカイブログも少しずつ増加した。
クラッシュリカバリ	PITR を使用しない場合のバックアップファイルからのリカバリと比べて、約 1/3 の時間で完了した。 アーカイブログのディスク使用量は変わらなかった。

ロードは、規模あたりの性能は一定であり、8.0.0beta5 の PITR 無しの結果とは異なり、7.4.6 と同様の結果となった。pgbench のロード処理の特性によるものと考えられるが、今回の評価では原因を突き止めるまでには至らなかった。

バックアップの性能は、ほとんど変わらなかった。バックアップは検索処理なので、アーカイブログに書き込むことは無いためである。

リカバリとインデックス再構築は、PITR を使用すると若干の性能低下と、アーカイブログの増加が確認できた。この二つはデータベースを更新する処理なので、それによりアーカイブログへのコピーが発生したためである。

クラッシュリカバリは、バックアップファイルからのリカバリと比べると、約 1/3 の時間でリカバリが完了した。

PITR を使用すると、更新系の処理でアーカイブログのコピーが発生し、それによる若干の性能低下が確認できた。今回の評価では、cp コマンドでコピーしたが、別のコマンドを使用することで、性能が変わることが考えられる。

クラッシュリカバリは、生成直後のデータベースからのリカバリ時間を測定した。今回の評価では実施しなかったが、ある程度のデータが入っているデータベースからのリカバリ時間は、今回の結果とは異なると考えられる。

## 7.4.2 まとめ

大規模データベースの運用性について、PostgreSQL7.4.6 と 8.0.0beta5 について評価を行った。

ロード、バックアップ、リカバリ、インデックス再構築、クラッシュリカバリという大規模なデータベースでは必須となる機能が、実用的な時間内で完了するのかを検証した。特に、PostgreSQL8.0 に追加された PITR を使用したクラッシュリカバリは、信頼性のあるデータベースを運用するための重要な機能である。PITR を機能だけでなく性能においても実用性があるかを検証することは、システム構築において PITR を採用するかの、一つの判断材料になる。

今回の評価で分かったことを、以下にまとめる。

- ・ バックアップ、リカバリ、インデックス再構築、クラッシュリカバリの性能は、規模に応じてほぼ一定であった。ロードは、一定の場合と若干の変動がある場合があった。
- ・ インデックス再構築は、8.0.0beta5 は 7.4.6 より約 10%ほど速かった。
- ・ テーブルスペースを使用すると、リカバリとインデックス再構築で約 5%ほど速かった。ロードも若干速かったが、スケールファクター4000 では特に速かった。
- ・ PITR を使用すると、リカバリで約 5%ほど、インデックス再構築で約 13%ほど遅くなった。
- ・ PITR を使用すると、ロードとリカバリで、アーカイブログの増加量がデータベースクラスタのディスク使用量の約 1.12 倍となった。インデックス再構築でも、少しずつ増加した
- ・ 今回の評価条件では、PITR を使用したクラッシュリカバリでは、通常のリカバリと比べて約 1/3 の時間で完了した。

今回の評価では、以下の問題について原因を特定するまでには至らなかった。

- ・ pgbench を使用したロードは、規模あたりの性能が一定の場合と、変動する場合があった。pgbench は入力ファイルを使用しないので、その影響も考えられるが、原因と特定するまでには至らなかった。
- ・ スケールファクター4000 でのリカバリは、7.4.6 は問題ないが、8.0.0beta5 ではディスク容量不足のエラーとなった。エラーの直前のディスク使用量は 90%であった。

今回の評価では、PostgreSQL を利用した大規模データベースの運用性について、スケールファクター4000 までの規模であるが、データベースのサイジングのための基礎データが得られた。

また、今回の評価を行った時点では、実際に利用しているユーザが少ない PostgreSQL8.0 においても、大規模データベースの運用性について、実際に動かして評価することができた。特に、8.0 で追加されたテーブルスペースと、PITR を使用したクラッシュリカバリについて、効果があることが確認できた。

PITR については、今回の評価はバッチ的な処理で測定したので、最大 13%の性能低下となったが、通常の OLTP での運用中においては、そこまでの性能低下とはならないだろう。また、アーカイブログのコピーとして、cp コマンドを使用したか、その他のコマンドを使用することで、性能が変わることも考えられる。また、アーカイブログの増加率についても、注意してシステムを設計する必要がある。

このようなデメリットはあるが、クラッシュリカバリの所要時間は、通常のバックアップファイルからのリカバリと比べると、約 1/3 の時間で完了したのは、大きなメリットである。障害時の短時間による復旧を要求されるシステムでは、重要な機能であると言える。PITR の使用は、このようなトレードオフを考慮して、採用すべきかを決定する必要があるだろう。

なお、今回の評価では、データベース作成直後の状態からのクラッシュリカバリなので、データベースクラスタの物理イメージによるリカバリ時間はほとんど 0 秒であり、アーカイブログからのリカバリが、通常と比べて約 1/3 の時間となった。

#### 今後の課題

今回の評価で原因を特定できなかった問題について、さらなる調査を行う必要があるだろう。

また、クラッシュリカバリにおいて、実際にデータの入った状態で物理イメージのバックアップを行った場合について、性能測定を行うのが望ましいだろう。

## 8 DBT-1 の MySQL 用改変基本設計

### 8.1 DBT-1 Architecture 概要

#### 8.1.1 OSDL- DBT 1 (Open Source Development Labs Database Test1)

OSDL DataBase Test1 プロジェクト(OSDL-DBT-1)は、LinuxOS やオープンソースソフトウェアのためのトランザクションベースの負荷テストツールとして、簡便に利用できるように開発された。これにより、より広い開発コミュニティで共有できる事を期待している。

このテストツールは TPC-W から派生し、単純化したものである。TPC-W によるワークロードは、現実的なパフォーマンスを最適化していると考えられているので、OSDL-DBT-1 では TPC-W をテンプレートとして使用している。

TPC-W ワークロードは、ブラウジングによりオンライン書店から商品を購入するウェブユーザの活動をシミュレートする。OSDL-DBT-1 テストは、TPC-W のワークロード特性を使用して、実行時のボトルネックを検出するため、また相対的なパフォーマンス向上のための指標として、オープンソース開発者によって作成された。

TPC ベンチマークは厳密な比較を行うためのツールとして使用されるように意図されているため、OSDL-DBT-1 によるベンチマーク結果とは、いくつかの点において比較することはできない。TPC は厳密な公表に対応できるように全ての結果を公表し、競争者間の公平な比較を保証する監査規則をもっている。また、TPC の規則では全体的な有効性とベンチマークに使用された全ての製品に対して、トランザクション・コストの開示を必要としている。

実際にシステム開発現場でそれらの規則を順守することは現実的ではない。これらから、OSDL-DBT-1 テストによってもたらされた結果は TPC-W ベンチマークとは完全に一致しない。

#### 8.1.2 OSDL-DBT-1 の概要

OSDL-DBT-1 はトランザクションに基づいたテストキットである。DBT1 は、TPC-W 仕様によって定義されたものと同様のワークロードによりデータベースを動作させる。DBT-1 は、データベース、トランザクション管理サーバおよびドライバで構成されている。

なお、この章で記述するのは DBT-1 の SAP DB 用の ODBC 版を基にしている。

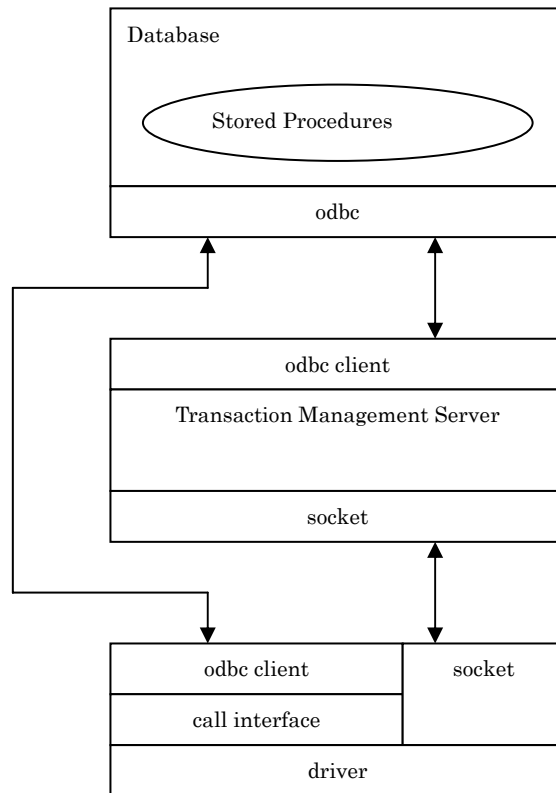


図 8.1-1 OSDL-DBT-1 コンポーネント 構成

OSDL-DBT-1 ドライバは、TPC-W の RBE ( Remote Browse Emulators ) と同様のタスクを実行する。OSDL-DBT-1 ドライバは EB(Emulated Browser)を生成し、管理する。EU(Emulated Users)は、TPC-W ベンチマーク中の EB と同様なロジックであるが、HTTP リクエストの代わりにデータ構造を生成する。

トランザクション管理サーバはデータベースにドライバを接続する中間層である。それはトランザクション管理機能を実行し、ODBC を通してデータベースと通信を行う。

OSDL-DBT-1 および TPC-W のデータベースは同じテーブルとテーブル定義であり、同じ生成ルールに従っている。

ストアプロシージャは同様のビジネス・ロジックを実行する。OSDL-DBT-1 のストアプロシージャのうち、いくつかは TPC-W で定義されたものより少ないデータ量を返す。

## 8.2 基本的な構造(現状)

DBT-1 は表 8.2-1 の様にモジュールに分かれて構成されている。

表 8.2-1 DBT-1 構成表

名称	機能概要
dbdriver	トランザクションを起動する仮想ユーザの動作をシミュレートする
appServer	トランザクション管理サーバ
appCache	キャッシュサーバ
DBT-1 LibODBC	DBT-1 の ODBC インターフェース。このライブラリを経由してデータベースにアクセスする
MaxDB ODBC	MaxDB の ODBC ドライバ
ストアードプロシージャ	個々のトランザクション処理を実装している

appCache が DB にアクセスするのは、初期化時のみ(アプリケーションサーバのキャッシュをシミュレート) appCache の内部テーブルに保存する。

- ・ search\_result\_author を item\_count の 10 分の 1
- ・ search\_result\_title を item\_count の 5 分の 1

item\_count は起動時の引数。デフォルトは 1000 で動作する。

LibODBC 各関数の呼び出しは、interfaces/db.c の process\_interaction() が振り分けを行う。

dbdriver と AppServer は C 言語で記述されており、ストアードプロシージャを呼び出すために ODBC APIs を使用している。

開発当初、データベースは SAPDB v7.3.0.21、OS は RedHat7.2 を想定していたが、任意の標準的な Linux ディストリビューション上で実行するよう設計されている。

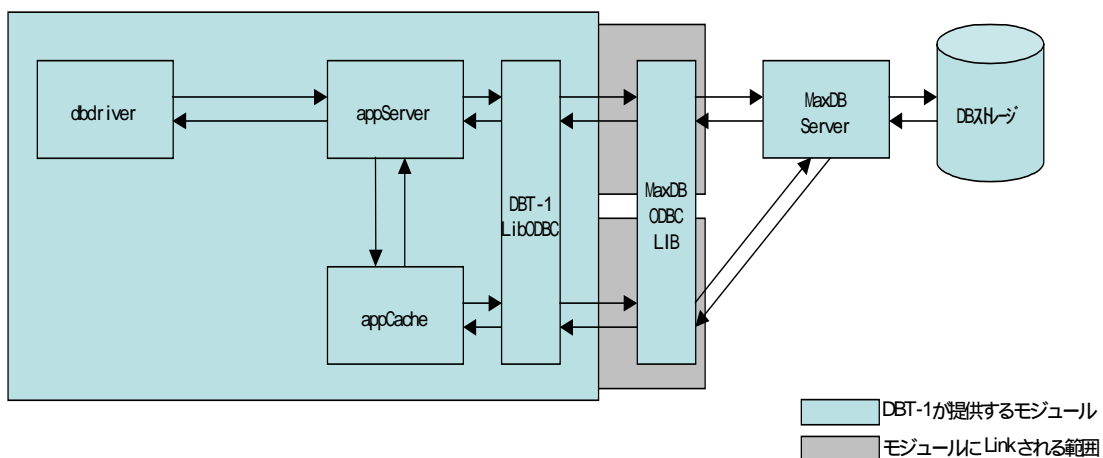


図 8.2-1 DBT-1 構成概要

## 8.2.1 dbdriver の構造

dbdriver は想定された仮想ユーザ数のスレッドを立ち上げ、個々のユーザ操作を開始する。dbdriver\_p1 と dbdriver\_p2 から構成される。仮想ユーザの活動をシミュレートする各スレッドから構成されマルチスレッドで動作する。

- dbdriver\_p1

ODBC インターフェイスにリンクされ、データベースと直接通信する。また、appServer を迂回した形態をとる。

このドライバは、RDBMS ストアドプロシージャの単純な機能的なテストのために使用ができる。

- dbdriver\_p2

appServer と通信する。ソケットインターフェイスを利用して接続する。

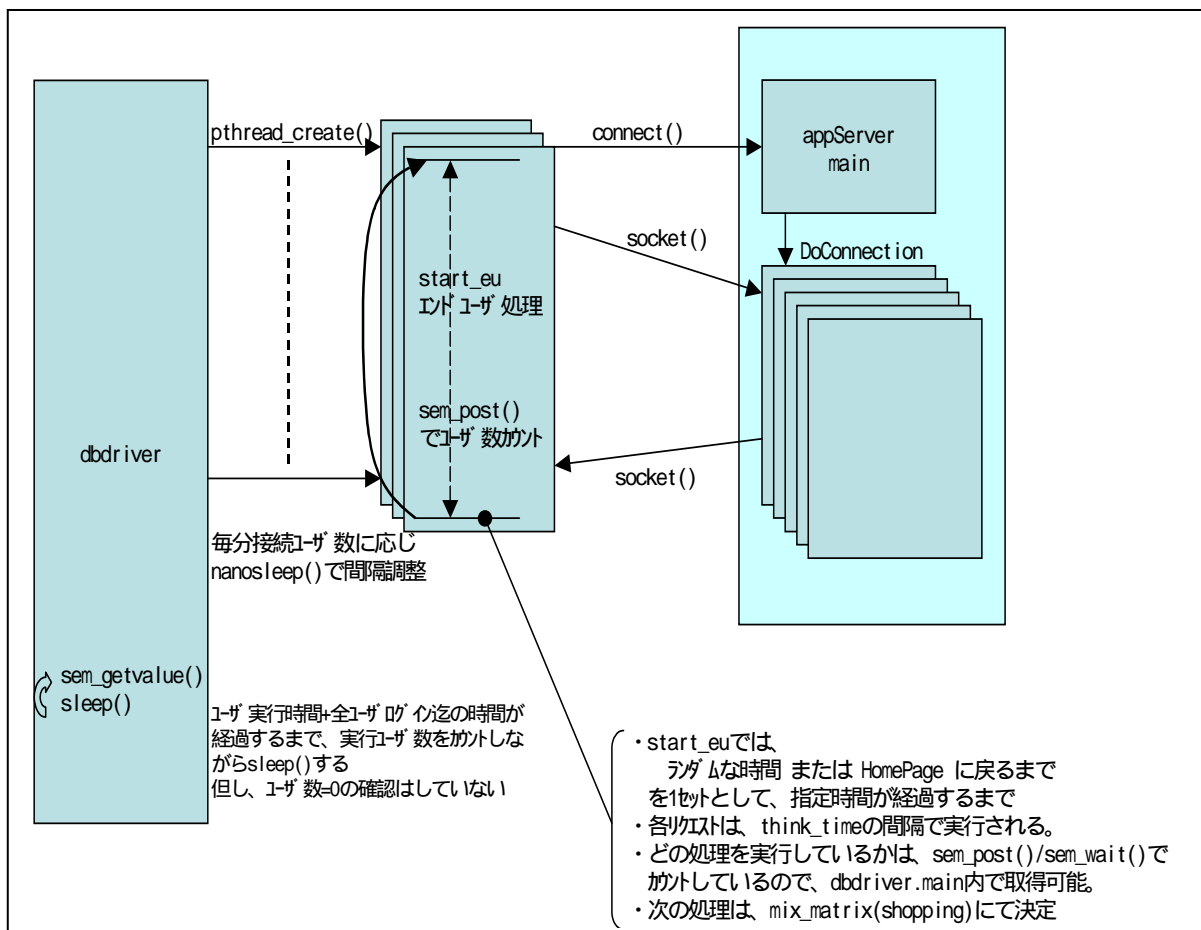


図 8.2-2 dbdriver 構造

## 8.2.2 appServer の構造

appServer は Web 3 階層モデルにおける、アプリケーション層にあたるソフトウェアである。dbdriver から要求されたトランザクションを受け取り、データベースにクエリを送り、得られた結果を dbdriver に処理結果を返す。

appServer は多くの個々の仮想ユーザを扱うために、指定された接続数でデータベース接続を行う。

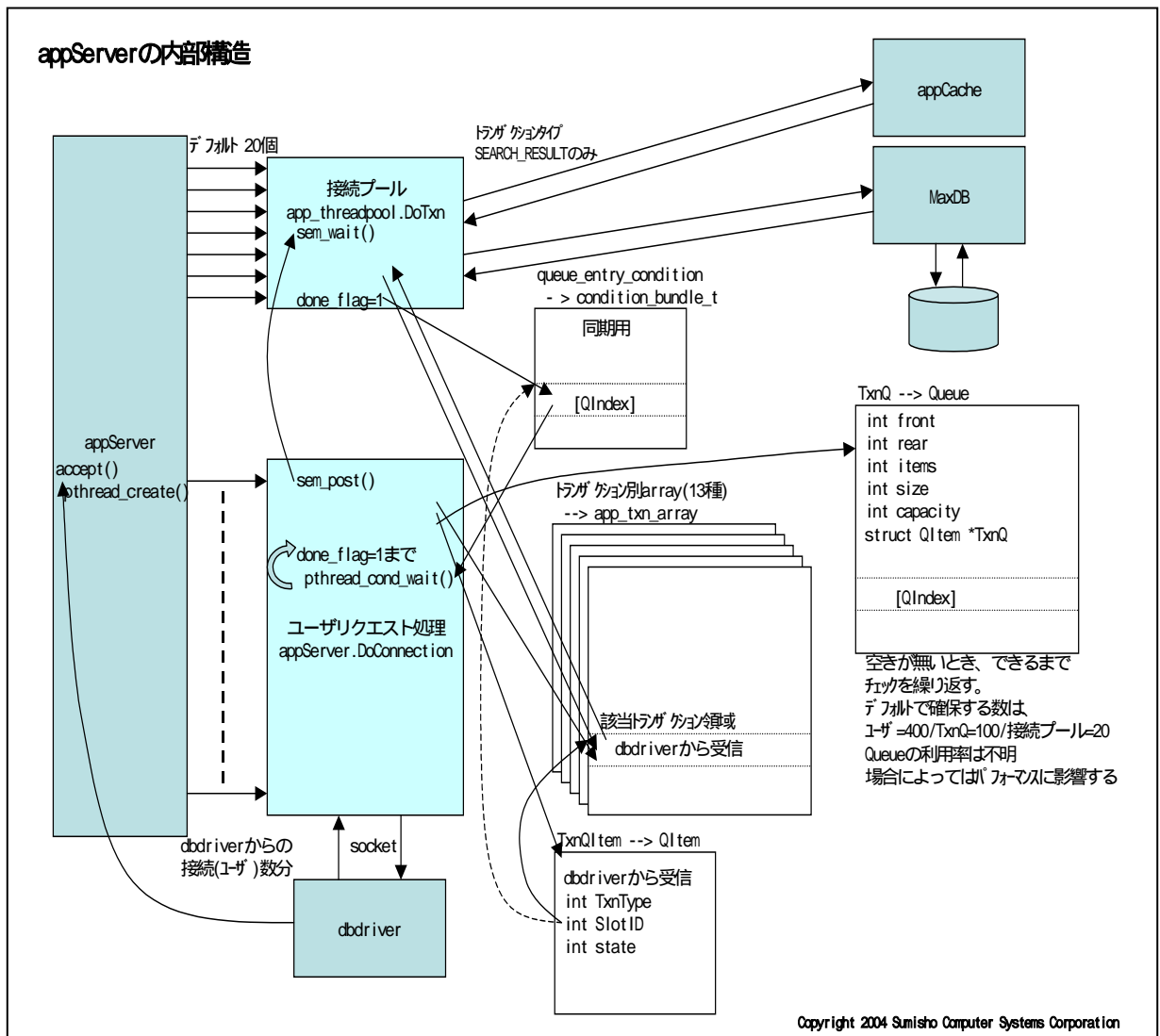


図 8.2-3 appServer 構造

### 8.3 DBT-1 のトランザクション概要

DBT-1 では表 8.3-2 に示す TPC-W に準拠したトランザクションパターンを利用している。

表 8.3-1 TPC-W トランザクションパターン一覧

番号	トランザクション	内容
1	ホーム	仮想店舗のホームページ。製品リスト（新製品、ベストセラー製品）へのリンクを含むウェブページの情報を返す。
2	ショッピングカート	関連する顧客のカート情報(日付、追加した新しいアイテム、既存のアイテム情報)を更新し、最新のカート情報を返す。ショッピング・セッションが未定義の場合、新しいセッションを作成する。
3	顧客登録	既存顧客の場合、登録されている必要な情報を返し、新規顧客の場合には登録を促すページを返す。
4	購入リクエスト	登録済みの新規顧客または認識されている顧客かを確認し、顧客についての情報を表示する。また、カートに関するサマリ情報と、クレジットカード情報や SHIPPING・オプションの入力フィールドを含めたページを表示する。
5	購入確認	既存顧客の関連するカートの内容を新しく注文として作成し、支払い認証を実行する。新しく作られた注文の明細を含むページを返す。
6	注文照会	再訪した顧客の本人認証のためのページを返す。以降の処理のエントリページとなる。
7	注文表示	顧客によって出された最終の注文のステータスを返す。
8	検索リクエスト	指定した商品を検索する条件を入力するページを返す。
9	検索結果表示	与えられた条件に適合する商品リストのページを返す。
10	新商品	最近リリースされた商品一覧のページを返す。
11	ベストセラー	ベストセラーの商品一覧のページを返す。
12	商品詳細	選択した商品の詳細情報のページを返す。
13	管理者リクエスト	顧客に商品の最新版へのアクセスを許可するページを返す。
14	管理者確認	商品を更新し、更新した商品の詳細を確認するページを返す。

トランザクション 3 に関しては現在実装されていない。(今後削除される予定)

トランザクション 8 に関しては現在実行されていない。

## 8.4 DBT-1 動作特性

### 8.4.1 負荷パターン

図 8.4-1 の通り、DBT-1 の負荷パターンは設定した仮想ユーザのコネクションが全て接続した後で、最初の仮想ユーザから順次処理を終えるまでの間に負荷のピークが来るように設計されている。従って、計測時には負荷のピークのタイミングを想定する必要がある。

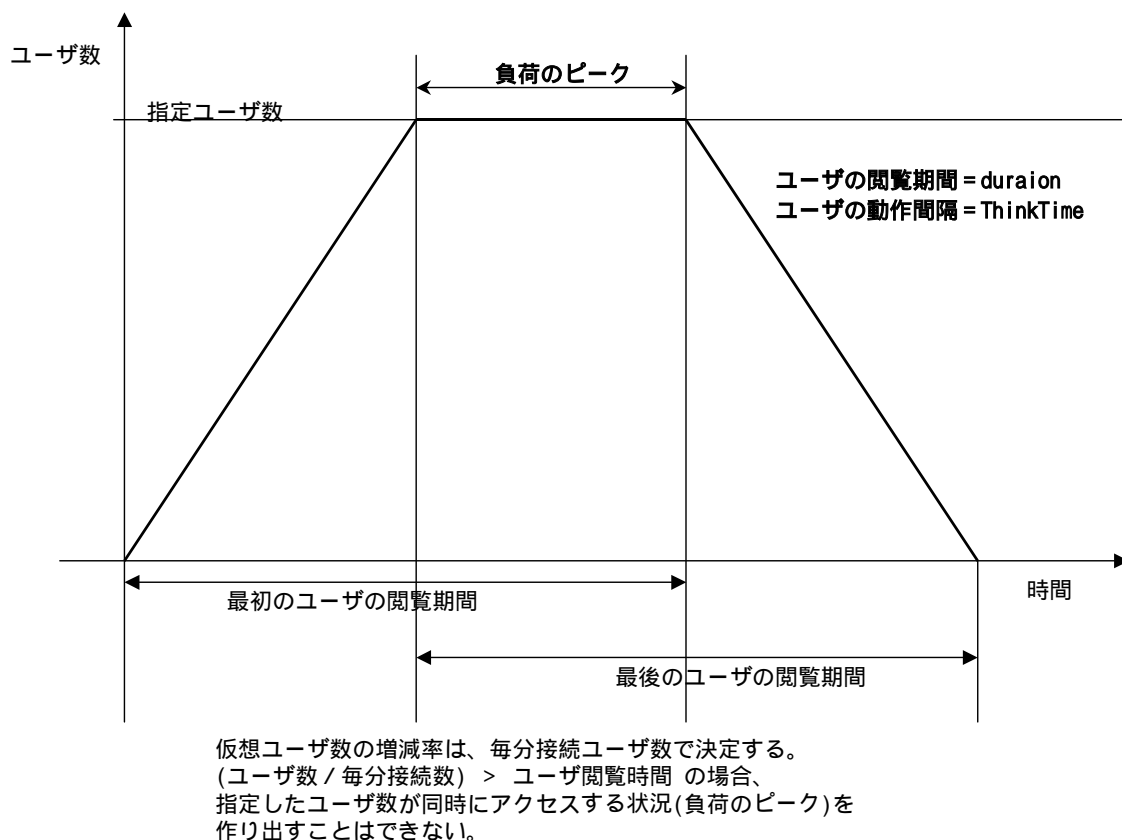


図 8.4-1 負荷のかかり方

Duration：仮想ユーザがブラウザを使用しているトータルの時間（秒数）を表す。

ThinkTime：仮想ユーザがアクションを起こしてから次のアクションをするまでの平均待ち時間（秒数）を表す。

## 8.4.2 DBT-1 計測タイミングの調整について

### 8.4.2.1 動作タイミング

dbdriver は終了時にメッセージを出力するが、このメッセージが出力される前に、BT 値が計算される。また、BT 値計算後も統計収集は継続されている。

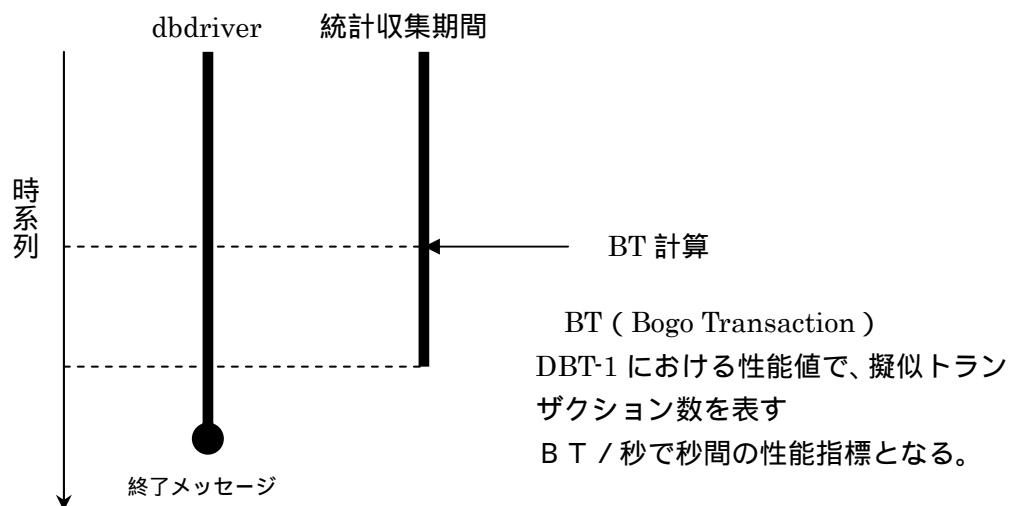


図 8.4-2 動作タイミング

### 8.4.2.2 タイミングの計算

#### (1) BT 値算出の対象期間と算出ロジック

収集回数

run\_duration が 1200 より大きい場合

$$\text{collect\_count} = (\text{run\_duration} - 20 * 60) / 10$$

1200 以下の場合

$$\text{collect\_count} = (\text{run\_duration}) / 10$$

10 秒間隔で上記計算を繰り返す

run\_duration = 3000 の場合、収集回数は 180 回となる

計算までの待ち時間

$$\text{run\_duration} - \text{collect\_count} * 10$$

dbt1.config で指定した値を経過した後に BT を計算

(2) dbdriver の動作時間

$\text{duration} + (\text{int}) ((\text{eus} / \text{rampuprate}) * 60);$

処理時間は、dbt1\_config に ユーザ数 ÷ 1 分間の接続ユーザ数 分加えた時間

注) スクリプト側では、この追加分が考慮されていない。現状では指定 duration やユーザ数が大きくなるほど、差も大きくなる。

計算例

ユーザ数 10 毎分 5 ユーザ接続 duration = 3000 の場合、

スクリプト =  $(3000 - 20 * 60) / 10 * 10 = 1800$  秒

dbdriver =  $3000 + (10 / 5) * 60 = 3120$  秒

## 8.5 現状の問題点

現状の DBT-1 は各 RDBMS 毎に異なるストアードプロシージャによる実装を行っているために、以下の点が課題として考えられる。

- (1) データベースによって固有のストアードによるトランザクション・ロジックの実装のために、RDBMS によって動作状況が異なり正当な評価指標とし難い。
- (2) 上記(1)により、DBT-1 自身のポータビリティが低下している。
- (3) 最近のシステム開発の傾向として Java を利用したオブジェクト指向開発が盛んになり、ビジネスロジックをストアードで実装する事が少なくなった。従って純粋な SQL 処理でのパフォーマンスを評価する環境が必要である。
- (4) RDBMS の種類によってデータ取得の実装が異なり、一定数の取得やフェッチ動作で代用をしている。
- (5) DBT-1 の実行結果はテキストファイルのみであり、概要を把握するにはグラフ出力等がほしい。結果のビジュアル化が必要と思われる。
- (6) TPC-W と異なる部分が多くある。特にトランザクションのアイソレーションレベルなど根本的な問題も残っている。(オリジナルは SAP DB のトランザクション処理を想定している)
- (7) TPC-W で規定されている http リクエストが処理されていない。
- (8) 検索リクエストトランザクションに関しては実行されていない。(顧客登録トランザクションに関しては現在実装されていないが、削除の予定となっている)

このうち特に(1)～(5)に関して解決を目指す。

(6) に関しては、RDBMS によっては、トランザクションサポートが SAP DB と異なるものが多く、本プロジェクトでは取り扱わない。(7)～(8)に関しては、DBT-1 自体の動向を見定める事とし、今回の設計には組み入れない。

## 8.6 設計方針

### (1) ポータビリティの向上

現状の RDBMS 毎に異なるストアドプロシージャをベースとせず、アプリケーション側にトランザクション機能を実装する事を基本とする。

実行に必要なパラメータは、極力集中させ、メンテナンスビリティを向上させる。

また、他のデータベースに対応する際の改変コスト削減を考慮し、SQL は全てヘッダ部分に記述する。

### (2) 構成のフレキシビリティの向上

性能評価を実施する構成として、以下を想定した作りとする。

- ・ ターゲットとテストサーバが同一の場合
- ・ ターゲットとテストサーバを分離する場合

### (3) 結果のビジュアライズ化

BT ファイル内各インタラクションの平均時間をプロットする。

ただし、単発の結果で作成できるもの想定し、複数テスト結果のサマリは対象外とする。

### (4) DBT-1 の利用方法、アルゴリズムなど、基本機能については変更しない。

原則的に、現状の DBT-1 が実装している TPC-W との準拠性 / 非準拠性は変更しない。

ただし、現状バージョン( MaxDB 対応版 )が持っている不具合を原則として解消する。

- ・ DB 検索結果取得時の件数制限を廃止
- ・ スレッドスタックサイズを含む変数領域の調整

詳細は基本設計書を参照の事

## 8.7 設計概要(基本設計)

### 8.7.1 概要

設計方針に基づき、DBT-1 を改変する。具体的には ODBC トランザクション処理部分の構造を以下のように変更する。

改変する作業内容は、以下のとおり。

- (1) DBT-1 LIB ODBC にトランザクション処理機能を追加
- (2) 結果表示のグラフ化を行うプログラムを新規追加
- (3) make 環境の修正

### 8.7.1.1 ODBC トランザクションプログラム

各トランザクションに対する処理を実装する。以下にトランザクションプログラムにおける処理概要を記述する。

- (1) 現在の仕様と同様、各トランザクション処理関数を

`int execute_XXXX(struct db_context_t *, struct XXXX_t *)`

の形式 (`XXXX` はトランザクション名) で実装し、内部で処理に必要なデータ設定、SQL の実行を実施する。

- (2) データベースとの接続は現在と同様、上位プログラムより第 1 引数として受取る。

- (3) コミット / ロールバックは `execute_XXXX` 関数の戻り値により判定し、エラー終了の場合はロールバック、正常終了の場合はコミットを実施する。

- (4) ODBC の SQL 処理は基本的に以下の流れで実装する。

SQLPrepare

SQL ヘッド (後述) で宣言した SQL 文を RDBMS 側で利用できるよう準備する。

SQLBindParameter

パラメータの設定。SQL 条件として渡すデータを設定する。

SQLExecute

SQL の実行。

SQLFetch (選択処理のみ)

実行結果をフェッチする。SQL で取得されたすべてのデータをフェッチする。

SQLGetData (選択処理のみ)

フェッチしたデータを列ごとの値として取得する。

- (5) ODBC の各 SQL 処理終了時に、エラー判定を実施する。SQL 処理の各戻り値が `SQL_SUCCESS` もしくは `SQL_SUCCESS_WITH_INFO` の場合は正常処理とみなす。

- (6) エラーの場合、エラー処理を実施し、エラー終了(1)を戻り値として返す。

- (7) トランザクション処理が正常に終了すると、正常終了(0)を戻り値として返す。

- (8) 再度 SQL を実施するため、SQL ステートメントをクリアする。

### 8.7.1.2 ODBC トランザクションプログラムヘッダ

各トランザクションプログラムでインクルードするヘッダファイルを実装する。

- (1) `odbc_interaction.h` のインクルード。
- (2) 各トランザクションで使用する SQL 文を定義したヘッダファイルのインクルード
- (3) `execute_XXXX()` のプロトタイプ宣言。

### 8.7.1.3 SQL ヘッダ

各トランザクションプログラムで使用する SQL 文を宣言する。

(例) `#define STMT_HOME "SELECT c_fname, c_lname from customer where c_id=?"`

また、MySQL 対応のため、以下に記述する現在 MaxDB に対応しているデータベース・スクリプト (テーブル作成、データロード等) を修正する。

- |   |
|---|
| <ol style="list-style-type: none"><li>1. <code>build.sh</code> . . . データベーススクリプトを実行するスクリプト。</li><li>2. <code>address.sql</code> . . . <code>address</code> テーブルにデータをロードする SQL スクリプト。</li><li>3. <code>author.sql</code> . . . <code>author</code> テーブルにデータをロードする SQL スクリプト。</li><li>4. <code>cc_xacts.sql</code> . . . <code>cc_xacts</code> テーブルにデータをロードする SQL スクリプト。</li><li>5. <code>country.sql</code> . . . <code>country</code> テーブルにデータをロードする SQL スクリプト。</li><li>6. <code>create_db.sh</code> . . . データベースを作成するスクリプト。</li><li>7. <code>create_indexes.sql</code> . . . 索引を作成する SQL スクリプト。</li><li>8. <code>create_indexes.sh</code> . . . 索引を作成するスクリプト。</li><li>9. <code>create_keys.sql</code> . . . 外部キーを作成する SQL スクリプト。</li><li>10. <code>create_keys.sh</code> . . . 外部キーを作成するスクリプト。</li><li>11. <code>create_tables.sql</code> . . . テーブルを作成する SQL スクリプト。</li><li>12. <code>create_tables.sh</code> . . . テーブルを作成するスクリプト。</li><li>13. <code>drop_db.sh</code> . . . データベースを削除するスクリプト。</li><li>14. <code>drop_tables.sql</code> . . . 全てのテーブルを削除する SQL スクリプト。</li><li>15. <code>drop_tables.sh</code> . . . 全てのテーブルを削除するスクリプト。</li><li>16. <code>item.sql</code> . . . <code>item</code> テーブルにデータをロードする SQL スクリプト。</li><li>17. <code>load_db.sh</code> . . . 全てのテーブルにデータをロードするスクリプト。</li><li>18. <code>order_line.sql</code> . . . <code>order_line</code> テーブルにデータをロードする SQL スクリプト。</li><li>19. <code>orders.sql</code> . . . <code>orders</code> テーブルにデータをロードする SQL スクリプト。</li></ol> |
|---|

## 8.7.2 構造

### 8.7.2.1 変更前の構造

現行の MaxDB7.5 版 DBT - 1 のトランザクション実装概要を図 8.7-1 に示す。トランザクション管理サーバ層では ODBC 経由で MaxDB 上に実装されているストアプロシージャをコールし、MaxDB は該当するストアプロシージャを実施して値を返している。

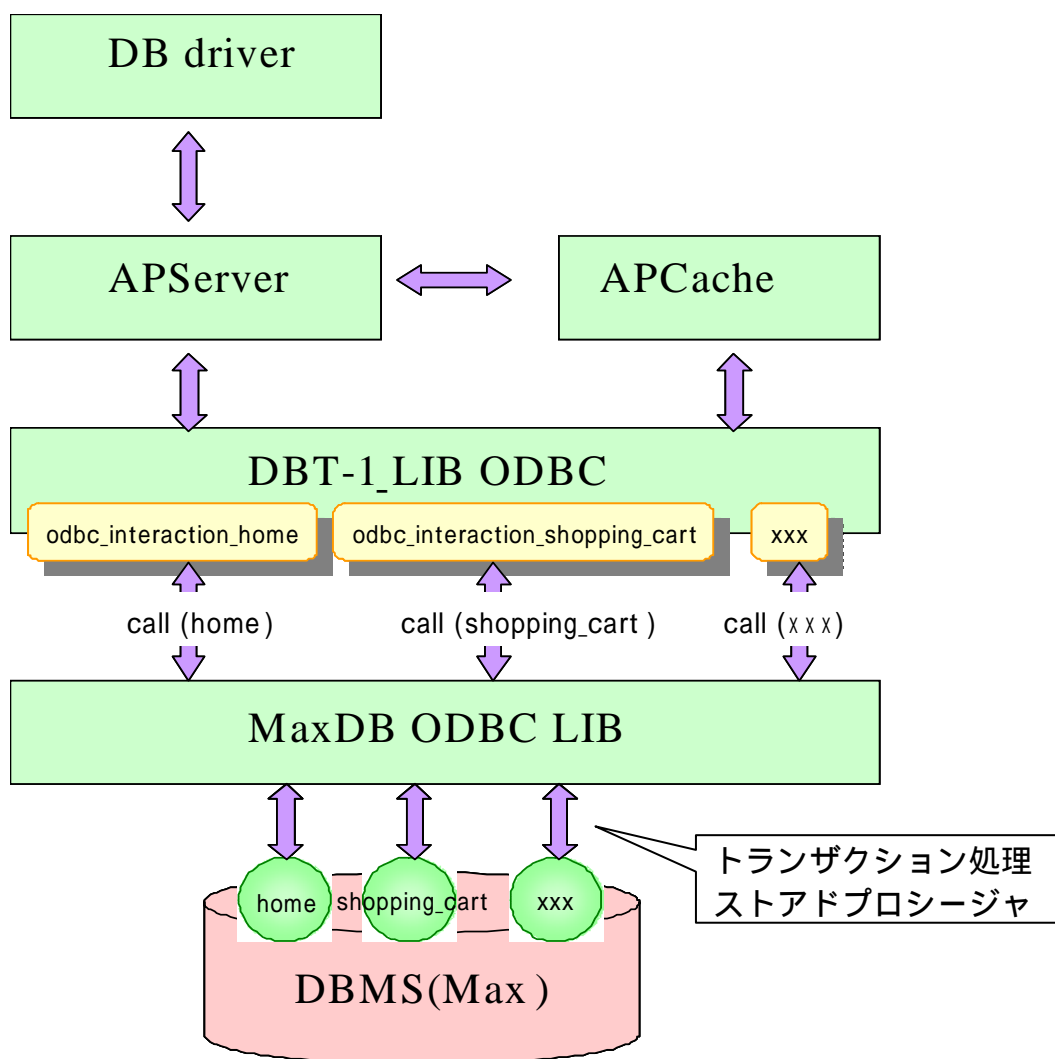


図 8.7-1 変更前の構造

### 8.7.2.2 変更後の構造

図 8.7-2 は今回の変更後の想定構造を示している。ストアードプロシージャを使用せず、プログラムより直接に SQL 文を発行し、結果セットを取得する。

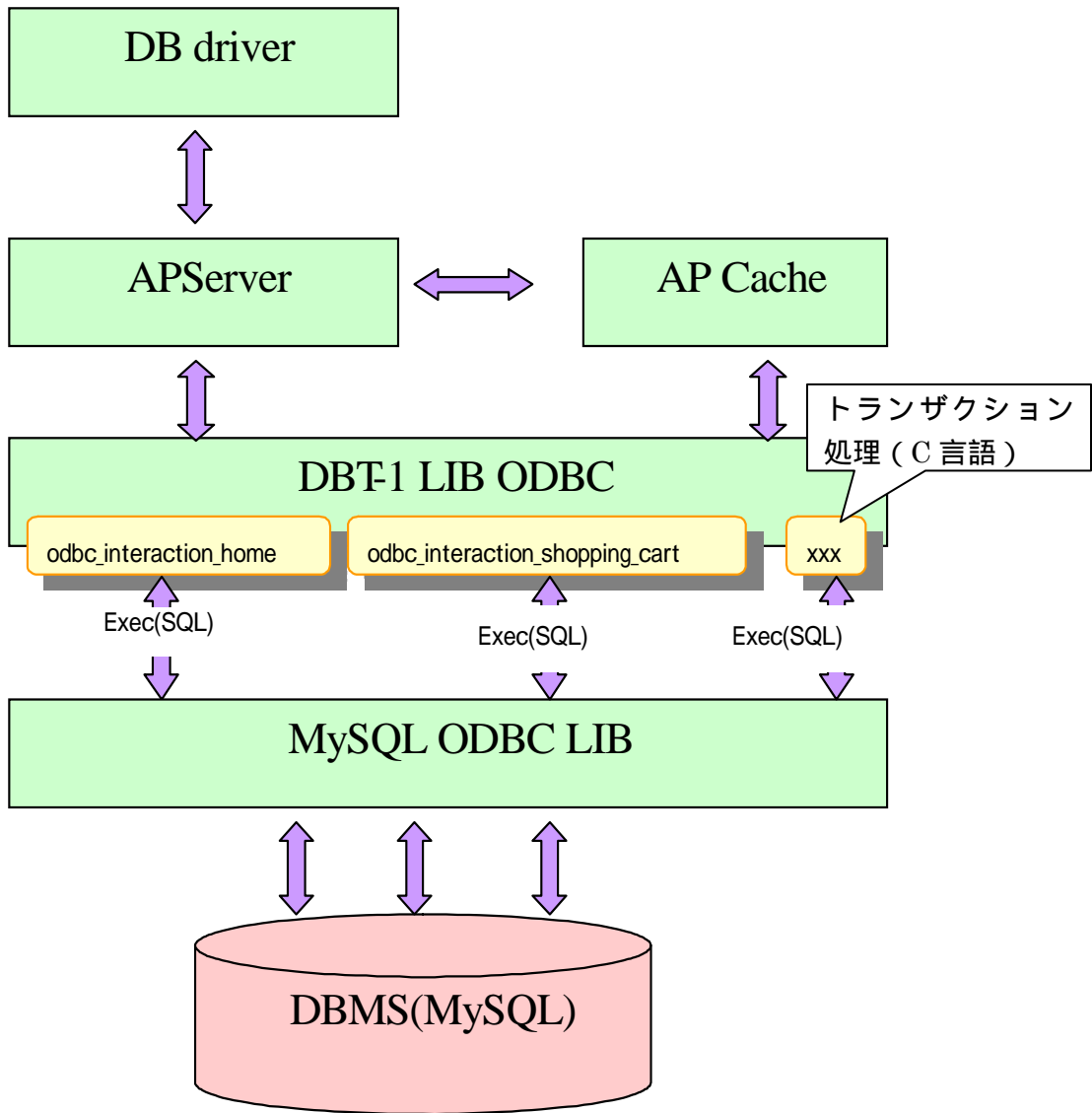


図 8.7-2 変更後の構造

### 8.7.2.3 結果のビジュアライズ化

DBT-1の出力結果からグラフを生成する処理を追加する。DBT-1テストとは独立した形態で次の機能を追加する。

出力グラフは、BTファイル内の各INTERACTIONの平均時間をプロットする。

#### (1) 対象データ

コマンドラインよりファイル名を指定する。

入力ファイルはDBT-1により出力されるBTファイルとする。

#### (2) グラフ出力

インタラクション毎の分布グラフを出力する。

ここで想定するのは、1回の性能評価テストによる出力ファイルから得られるインタラクション別の分布グラフであり、複数回実施した後の推移情報はスコープ外としている。

利用想定ツール：gnuplot

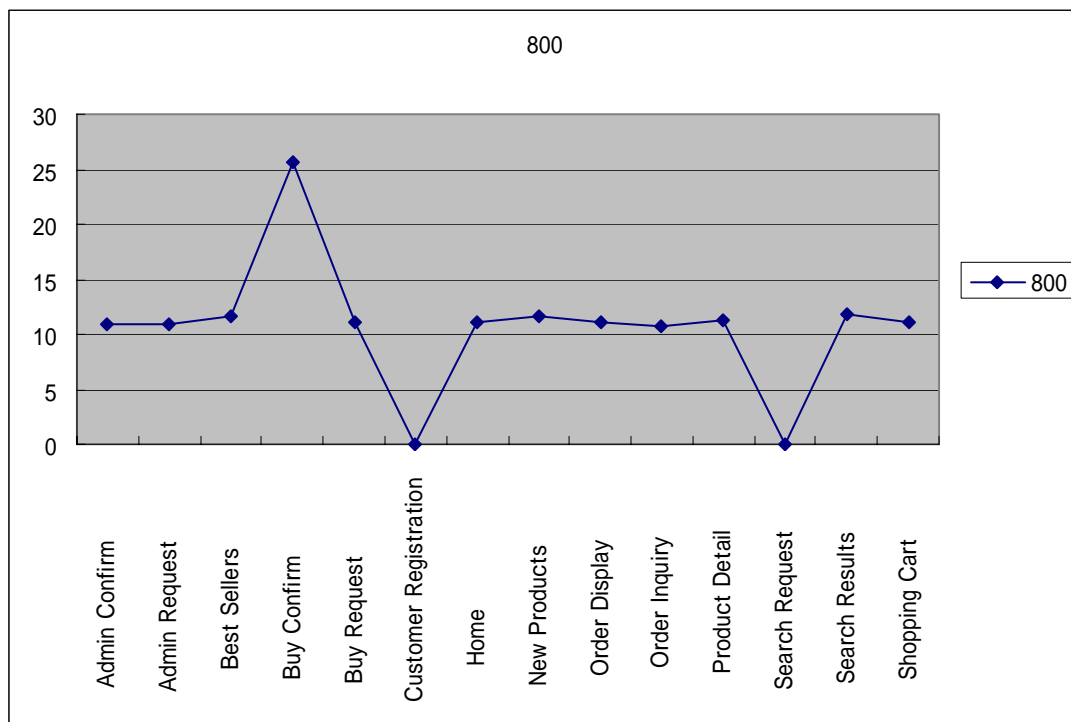


図 8.7-3 グラフ出力イメージ

### 8.7.3 改変見積り

改変作業に関して、以下のステップ数による見積りを行う。

#### 8.7.3.1 トランザクションプログラム改変

表 8.7-1 中の ( ) 内はストアードプロシージャを含む改変前のステップ数を示す。改変後は C 言語のステップ数を示す。改変前後のステップ数の差異はプログラムの分割等により生じる。

【ステップ数】改変前 約 7850 ステップ  
改変後 約 8200 ステップ

表 8.7-1 トランザクションプログラム別ステップ数

No	プログラム名	ステップ数 <sup>1</sup>	備考
1	Odbc_interaction_home.c	160 (160)	初期処理 (宣言、変数領域初期化等)・・・20 ODBC 処理 (含エラー判定処理)・・・50 ヘッダ処理 (プロトタイプ宣言、SQL 宣言等)・・・20 後処理 (領域解放処理等)・・・20 その他 (ロジック記述、終了処理等)・・・50
2	Odbc_interaction_shopping_cart.c	180 (500)	初期処理 (宣言、変数領域初期化等)・・・20 ODBC 処理 (含エラー判定処理)・・・50 ヘッダ処理 (プロトタイプ宣言、SQL 宣言等)・・・20 後処理 (領域解放処理等)・・・20 その他 (ロジック記述、終了処理等)・・・70
3	Odbc_interaction_buy_request.c	180 (620)	初期処理 (宣言、変数領域初期化等)・・・20 ODBC 処理 (含エラー判定処理)・・・50 ヘッダ処理 (プロトタイプ宣言、SQL 宣言等)・・・20 後処理 (領域解放処理等)・・・20 その他 (ロジック記述、終了処理等)・・・70
4	Odbc_interaction_buy_confirm.c	810 (480)	初期処理 (宣言、変数領域初期化等)・・・40 ODBC 処理 (含エラー判定処理)・・・600 ヘッダ処理 (プロトタイプ宣言、SQL 宣言等)・・・40 後処理 (領域解放処理等)・・・40 その他 (ロジック記述、終了処理等)・・・90
5	Odbc_interaction_order_inquiry.c	150 (100)	初期処理 (宣言、変数領域初期化等)・・・20 ODBC 処理 (含エラー判定処理)・・・50 ヘッダ処理 (プロトタイプ宣言、SQL 宣言等)・・・20

			後処理（領域解放処理等）・・・20 その他（ロジック記述、終了処理等）・・・40
6	odbc_interaction_order_display.c	250 (580)	初期処理（宣言、変数領域初期化等）・・・30 ODBC 処理（含エラー判定処理）・・・100 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30 後処理（領域解放処理等）・・・30 その他（ロジック記述、終了処理等）・・・60
7	odbc_interaction_search_request.c	100 (95)	初期処理（宣言、変数領域初期化等）・・・20 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・20 後処理（領域解放処理等）・・・20 その他（ロジック記述、終了処理等）・・・40
8	odbc_interaction_search_result.c	350 (340)	初期処理（宣言、変数領域初期化等）・・・40 ODBC 処理（含エラー判定処理）・・・150 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30 後処理（領域解放処理等）・・・40 その他（ロジック記述、終了処理等）・・・90
9	odbc_interaction_new_products.c	330 (740)	初期処理（宣言、変数領域初期化等）・・・50 ODBC 処理（含エラー判定処理）・・・100 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30 後処理（領域解放処理等）・・・20 その他（ロジック記述、終了処理等）・・・130
10	odbc_interaction_best_sellers.c	550 (750)	初期処理（宣言、変数領域初期化等）・・・40 ODBC 処理（含エラー判定処理）・・・300 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・40 後処理（領域解放処理等）・・・40 その他（ロジック記述、終了処理等）・・・130
11	odbc_interaction_product_detail.c	160 (240)	初期処理（宣言、変数領域初期化等）・・・20 ODBC 処理（含エラー判定処理）・・・50 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・20 後処理（領域解放処理等）・・・20 その他（ロジック記述、終了処理等）・・・40
12	odbc_interaction_admin_request.c	150 (160)	初期処理（宣言、変数領域初期化等）・・・20 ODBC 処理（含エラー判定処理）・・・50 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・20 後処理（領域解放処理等）・・・20 その他（ロジック記述、終了処理等）・・・40
13	odbc_interaction_admin_confirm.c	270 (225)	初期処理（宣言、変数領域初期化等）・・・30 ODBC 処理（含エラー判定処理）・・・100 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30

			後処理（領域解放処理等）・・・30 その他（ロジック記述、終了処理等）・・・50
14	getPromoImatges	440 (40)	初期処理（宣言、変数領域初期化等）・・・30 ODBC 処理（含エラー判定処理）・・・300 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30 後処理（領域解放処理等）・・・30 その他（ロジック記述、終了処理等）・・・50
15	CreateSC	250 (25)	初期処理（宣言、変数領域初期化等）・・・30 ODBC 処理（含エラー判定処理）・・・100 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30 後処理（領域解放処理等）・・・30 その他（ロジック記述、終了処理等）・・・60
16	AddToSC	610 (50)	初期処理（宣言、変数領域初期化等）・・・40 ODBC 処理（含エラー判定処理）・・・400 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・40 後処理（領域解放処理等）・・・40 その他（ロジック記述、終了処理等）・・・90
17	RefreshSC	260 (210)	初期処理（宣言、変数領域初期化等）・・・30 ODBC 処理（含エラー判定処理）・・・100 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30 後処理（領域解放処理等）・・・30 その他（ロジック記述、終了処理等）・・・70
18	getSCSubTotal	210 (25)	初期処理（宣言、変数領域初期化等）・・・30 ODBC 処理（含エラー判定処理）・・・50 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30 後処理（領域解放処理等）・・・30 その他（ロジック記述、終了処理等）・・・70
19	GetSCDetail	210 (200)	初期処理（宣言、変数領域初期化等）・・・30 ODBC 処理（含エラー判定処理）・・・50 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30 後処理（領域解放処理等）・・・30 その他（ロジック記述、終了処理等）・・・70
20	InitSCItems	170 (190)	初期処理（宣言、変数領域初期化等）・・・170
21	GetCustInfo	260 (30)	初期処理（宣言、変数領域初期化等）・・・30 ODBC 処理（含エラー判定処理）・・・100 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30 後処理（領域解放処理等）・・・30 その他（ロジック記述、終了処理等）・・・70

22	InsertCust	410 (50)	初期処理（宣言、変数領域初期化等）・・・40 ODBC 処理（含エラー判定処理）・・・200 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・40 後処理（領域解放処理等）・・・40 その他（ロジック記述、終了処理等）・・・90
23	UpdateSC	330 (40)	初期処理（宣言、変数領域初期化等）・・・30 ODBC 処理（含エラー判定処理）・・・150 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30 後処理（領域解放処理等）・・・30 その他（ロジック記述、終了処理等）・・・90
24	DigSyl	80 (40)	その他（ロジック記述、終了処理等）・・・80
25	GetOrderItems	260 (220)	初期処理（宣言、変数領域初期化等）・・・30 ODBC 処理（含エラー判定処理）・・・100 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・30 後処理（領域解放処理等）・・・30 その他（ロジック記述、終了処理等）・・・70
26	SearchAuthor	350 (580)	初期処理（宣言、変数領域初期化等）・・・40 ODBC 処理（含エラー判定処理）・・・100 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・40 後処理（領域解放処理等）・・・40 その他（ロジック記述、終了処理等）・・・130
27	SearchSubject	350 (580)	初期処理（宣言、変数領域初期化等）・・・40 ODBC 処理（含エラー判定処理）・・・100 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・40 後処理（領域解放処理等）・・・40 その他（ロジック記述、終了処理等）・・・130
28	SearchTitle	350 (580)	初期処理（宣言、変数領域初期化等）・・・40 ODBC 処理（含エラー判定処理）・・・100 ヘッダ処理（プロトタイプ宣言、SQL 宣言等）・・・40 後処理（領域解放処理等）・・・40 その他（ロジック記述、終了処理等）・・・130

### 8.7.3.2 データベーススクリプト改変

表 8.7-2 中の ( ) 内は改変前のステップ数を示す。

【ステップ数】改変前 約 670 ステップ

改変後 約 400 ステップ

表 8.7-2 データベーススクリプト改変ステップ数

No.	プログラム名	ステップ数 <sup>1</sup>	備考
1	build.sh	100 ( 90 )	
2	address.sql	25 ( 10 )	
3	author.sql	25 ( 10 )	
4	cc_xacts.sql	25 ( 10 )	
5	country.sql	25 ( 10 )	
6	create_db.sh	120 ( 115 )	
7	create_indexes.sql	25 ( 15 )	
8	create_indexes.sh	25 ( 1 )	
9	create_keys.sql	40 ( 30 )	
10	create_keys.sh	25 ( 5 )	
11	create_tables.sql	25 ( 20 )	
12	create_tables.sh	25 ( 3 )	
13	drop_db.sh	25 ( 5 )	
14	drop_tables.sql	25 ( 10 )	
15	drop_tables.sh	25 ( 3 )	
16	item.sql	30 ( 25 )	
17	load_db.sh	25 ( 10 )	
18	order_line.sql	25 ( 10 )	
19	orders.sql	25 ( 15 )	

## 9 付録資料一覧

### 9.1 DBT-3 による PostgreSQL の評価

- ・ DBT-3 トランザクション特性の解析

以上

本書は、独立行政法人 情報処理推進機構から以下の 8 社への委託開発の成果として作成されたものです。

委託先企業：(株)日立製作所(幹事会社)

(株)SRA、(株)NTT データ、新日鉄ソリューションズ(株)

住商情報システム(株)、(株)野村総合研究所、ミラクル・リナックス(株)

ユニアデックス(株) (五十音順)