

2005 年度上期

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

～OSDL DBT-1/3 による DBMS 評価編～

作成

OSS 技術開発・評価コンソーシアム

商標表記

- 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。
- 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。
- MySQL は MySQL AB の登録商標です。
- Oracle は、Oracle Corporation の登録商標です。
- 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.	OSDL DBTシリーズによるRDBMSの評価	1-1
1.1.	目的	1-1
1.2.	プロジェクトのスコープ	1-2
1.3.	概要	1-3
1.3.1.	評価内容	1-3
1.3.2.	評価環境	1-4
1.3.3.	オープンソースRDBMSについて	1-4
1.4.	負荷ツールOSDL DBTについて	1-5
1.4.1.	オープンソース・性能評価ツールDBTの概要	1-5
1.4.2.	OSDL- DBT 1 (Open Source Development Labs Database Test1)	1-6
1.4.3.	OSDL- DBT 3 (Open Source Development Labs Database Test3)	1-8
2.	全体の考察	2-1
2.1.	ポータビリティ向上	2-1
2.1.1.	OSDL DBT-1 のMySQL(ODBC)用改変	2-1
2.1.2.	OSDL DBT-1 のPostgreSQL(ODBC)用改変	2-1
2.1.3.	ポータビリティ向上のまとめ	2-1
2.2.	評価	2-3
2.2.1.	評価数値の取り扱い	2-3
2.2.2.	OSDL DBT-1 MySQL(ODBC)版の評価	2-3
2.2.3.	OSDL DBT-1 PostgreSQL(ODBC)版の評価	2-5
2.2.4.	OSDL DBT-1 MaxDB(ストアド)版 による 32bit/64bitの評価	2-6
2.2.5.	OSDL DBT-3 による 32bit/64bitの評価	2-8
2.2.6.	評価のまとめ	2-8
3.	OSDL DBT-1 のMySQL用改変	3-1
3.1.	現状の問題点	3-1
3.2.	改変方針	3-2
3.2.1.	概要	3-2
3.2.2.	構造	3-4
3.3.	改変作業	3-7
3.3.1.	スクリプト	3-7
3.3.2.	ソースコード	3-8
3.3.3.	追加ディレクトリ構成	3-10
3.3.4.	モジュール生成	3-10
3.3.5.	動作確認	3-10
3.4.	まとめ	3-11

4.	OSDL DBT-1 のPostgreSQL用改変.....	4-1
4.1.	改変の目的	4-1
4.2.	改変方針.....	4-1
4.2.1.	概要.....	4-1
4.2.2.	改変ファイル一覧.....	4-2
4.2.3.	構造.....	4-6
4.2.4.	動作確認.....	4-8
4.3.	まとめ.....	4-9
5.	OSDL DBT-1/3 測定手順.....	5-1
5.1.	環境設定	5-1
5.1.1.	Linuxのインストール	5-1
5.1.2.	DBT-1 用オプション設定	5-1
5.1.3.	MaxDB用オプション設定	5-2
5.2.	ミドルウェア.....	5-3
5.2.1.	MaxDBのインストール.....	5-3
5.2.2.	MySQLのインストール	5-5
5.2.3.	MyODBCのインストール	5-6
5.2.4.	PostgreSQLのインストール	5-7
5.3.	負荷ツール	5-10
5.3.1.	DBT-1 用の事前環境設定	5-10
5.3.2.	sshの準備.....	5-11
5.3.3.	DBT-1 のインストール.....	5-13
5.3.4.	DBT-3 のインストール.....	5-14
5.4.	DBT-1 評価手順.....	5-18
5.4.1.	前提条件.....	5-18
5.4.2.	テストデータの作成.....	5-18
5.4.3.	データベースの作成.....	5-19
5.4.4.	パラメータの設定.....	5-21
5.4.5.	起動方法.....	5-23
5.4.6.	グラフ化.....	5-25
5.5.	DBT-3 評価手順.....	5-27
5.5.1.	前提条件.....	5-27
5.5.2.	評価環境.....	5-27
5.5.3.	レポートの作成	5-32
5.6.	出力情報.....	5-35
5.6.1.	MaxDBのログ情報.....	5-35

5.6.2.	MySQLのログ情報.....	5-35
5.6.3.	PostgreSQLのログ情報.....	5-35
5.6.4.	DBT-1 のログ情報.....	5-36
5.6.5.	DBT-3 のログ情報.....	5-37
6.	OSDL DBT-1 MySQL(ODBC)版の評価.....	6-1
6.1.	概要.....	6-1
6.1.1.	環境定義書.....	6-1
6.1.2.	システム構成.....	6-2
6.1.3.	環境構築.....	6-2
6.2.	測定結果と考察.....	6-3
6.2.1.	チューニング前測定結果.....	6-3
6.2.2.	チューニング前の結果の考察.....	6-4
6.2.3.	チューニングのポイント.....	6-5
6.2.4.	チューニング後の測定結果.....	6-8
6.2.5.	チューニング後の結果の考察.....	6-12
6.2.6.	計測データ.....	6-14
6.3.	まとめ.....	6-15
7.	PostgreSQL (ODBC) 版OSDL DBT-1 の評価.....	7-1
7.1.	概要.....	7-1
7.2.	環境定義.....	7-1
7.2.1.	システム構成.....	7-1
7.2.2.	ソフトウェア.....	7-1
7.3.	環境構築と測定.....	7-2
7.4.	測定と考察.....	7-2
7.4.1.	測定結果の収集.....	7-2
7.4.2.	測定の観点.....	7-2
7.4.3.	ODBC接続のダイレクト接続による基本的な性能の変化.....	7-2
7.4.4.	DBT-1 PostgreSQL(ODBC)版とDBT-1 PostgreSQL(ストアド)版の性能比較 7-5	
7.4.5.	簡単なチューニングによる性能の変化.....	7-7
7.4.6.	DBT-1 PostgreSQL(ODBC)版でのPostgreSQLバージョン間の性能比較.....	7-10
7.5.	まとめ.....	7-12
8.	DBT-1 によるMaxDBの評価 -32bitと 64bit環境の比較-.....	8-1
8.1.	概要.....	8-1
8.1.1.	目的.....	8-1
8.2.	環境定義書.....	8-2

8.2.1.	システム構成.....	8-2
8.2.2.	Linuxのインストール	8-4
8.2.3.	MaxDB のインストール.....	8-4
8.2.4.	パラメータの設定.....	8-5
8.3.	評価手順書	8-7
8.4.	測定結果と考察	8-7
8.4.1.	測定結果.....	8-7
8.4.2.	考察.....	8-11
8.4.3.	インタラクションにおける性能の違い	8-12
8.4.4.	計測データ	8-13
8.4.5.	MaxDBログから見る 32bit と 64bit の違い.....	8-14
8.5.	まとめ.....	8-15
9.	DBT-3 によるPostgreSQLの評価 -32bitと 64bit環境の比較-	9-1
1.1.	概要	9-1
9.1.1.	評価概要.....	9-1
9.1.2.	目的.....	9-1
9.2.	環境定義書	9-2
9.2.1.	システム構成.....	9-2
9.2.2.	PostgreSQLのインストール	9-2
9.2.3.	ワークロード実施手順（初期セットアップ作業）	9-3
9.2.4.	ワークロード実施手順	9-3
9.3.	評価結果（評価1）	9-4
9.3.1.	実施状況.....	9-4
9.3.2.	指標値	9-5
9.3.3.	クエリごと所要時間.....	9-6
9.3.4.	実行時システム負荷	9-8
9.3.5.	考察.....	9-9
9.3.6.	スクリプトの問題点.....	9-10
9.4.	評価結果（評価2）	9-11
9.4.1.	測定結果.....	9-11
9.4.2.	考察.....	9-13
9.5.	まとめ.....	9-14
10.	付録資料一覧	10-1
10.1.	DBT-1 のMySQL用改変	10-1
10.2.	DBT-1 のPostgreSQL用改変	10-1

1. OSDL DBT シリーズによる RDBMS の評価

1.1. 目的

前年度に引き続きオープンソース RDBMS を企業システム等で利用する上で重要な評価手法の整備を行う。

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、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、より汎用的なツールとして活用できるよう改変を施した上で、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、最近注目を浴びている、IA32 アーキテクチャの 64bit 拡張版である CPU を使用し、32bit と 64bit の違いも合わせて検証する。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。

本評価ではデータベースのワークロードツールとして、OSDL の DBT シリーズを利用した。DBT はオープンソースライセンスで提供され、安価に簡便に RDBMS の評価ツールとして利用できる。ツールの選定に関して、着目した条件は以下の通りである。

- (1) DB ベンチマークに利用できる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 の簡易版性能評価のためのワークロード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、本評価で利用するものとして妥当と判断する。

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により、オープンソース RDBMS をより有効に活用するための指針となる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。

- ・ OSDL DBT-1(以下 DBT-1)に関しては、より汎用性を高め利用し易くするために、個々の DBMS に依存していたストアードプロシージャを廃し、C ベースにポーティングを行い、ポータビリティの向上を行った。
- ・ 手順策定の対象 RDBMS として、オープンソース RDBMS の PostgreSQL 及び MySQL を選択した。
- ・ 32bit と 64bit(今回は EM64T を対象とする)にて評価するアプリケーションモデルとしては、前年度採用した Web トランザクション系のアプリケーションである DBT-1、及び意思決定支援システム系のアプリケーションである OSDL DBT-3(以下 DBT-3)を対象とした。

本年度プロジェクトのスコープを昨年度と比較した表を、表 1.2-1 に記す。

表 1.2-1 プロジェクトのスコープ

	2004 年度下期	2005 年度上期
DBT-1	MaxDB(ストアード)版 PostgreSQL(ストアード)版 Oracle(ストアード)版-新規 の各評価	MySQL(ODBC)/PostgreSQL(ODBC)へのポーティング及び評価 32bit/64bit 環境における MaxDB(ストアード)版の評価
DBT-3	PostgreSQL の評価	32bit/64bit 環境における PostgreSQL の評価

1.3. 概要

1.3.1. 評価内容

本評価では以下の2点を評価項目として考えた。

1.3.1.1. DBT-1 のポータビリティ向上と適用範囲の拡大

(1) MySQL 用改変・評価

2004年度のMySQL改変調査結果に基づきDBT-1をMySQL用に改変する。(ストアードプロシージャからSQL文の発行へ)。また、改変後のDBT-1を利用してMySQLの評価を行う。

- ・ データベース及び関連製品：MySQL 5.0
- ・ オペレーティングシステム：MIRACLE LINUX V3.0
- ・ 性能評価ツール：OSDL DBT-1 MySQL(ODBC)版

(2) PostgreSQL 用改変・評価

(1)で作成したソースコードの一部をPostgreSQL用に改変し、DBT-1がストアードプロシージャに依存せずに正しく動作することを確認する。また、改変後のDBT-1を利用してPostgreSQLの評価を行う。

- ・ データベース及び関連製品：PostgreSQL8.0,8.1
- ・ オペレーティングシステム：MIRACLE LINUX V3.0
- ・ 性能評価ツール：OSDL DBT-1 PostgreSQL(ODBC)版

1.3.1.2. 64bit 環境での性能評価

(1) DBT-1 を利用した 32bit/64bit 環境での性能・信頼性評価

2004年度の成果物であるMaxDB用DBT-1を利用し、32bit/64bit環境での性能評価を実施する。

- ・ データベース及び関連製品：MaxDB 7.5
- ・ オペレーティングシステム：MIRACLE LINUX V3.0 及び V3.0 for x86-64
- ・ 性能評価ツール：OSDL DBT-1 MaxDB(ストアード)版

(2) DBT-3 を利用した 32bit/64bit 環境での性能・信頼性評価

2004年度の成果物であるPostgreSQL用DBT-3を利用し32bit/64bit環境での性能評価を実施する。

- ・ データベース及び関連製品：PostgreSQL7.4,8.0
- ・ オペレーティングシステム：Red Hat Enterprise Linux AS release 4
- ・ 性能評価ツール：OSDL DBT-3

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

- MIRACLE LINUX V3.0 – Asianux Inside
- Red Hat Enterprise Linux AS release 4

※カーネルバージョンは各章を参照

(2) RDBMS

- MaxDB 7.5.0.19 , 7.5.0.22
- MySQL 5.0.7(MyODBC 3.51.10)
- PostgreSQL 7.4.7 , 8.0 , 8.1

(3) 負荷ツール

- OSDL DBT-1
- OSDL DBT-3

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) MySQL <http://www.mysql.com/>

世界で最も普及しているオープンソース RDBMS。機能を削減しても高速性を追及していたが、バージョン 5 よりストアードプロシージャが実装されるなど、機能拡張が進んでいる。

(2) PostgreSQL <http://www.postgresql.org/>

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。

1.4. 負荷ツール OSDL DBT について

1.4.1. オープンソース・性能評価ツール DBT の概要

OSDL DBTシリーズは表 1.4-1に示す4つの負荷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 1.4-1 DBT シリーズ概要

OSDL DBT-1	TPC-Wの簡易版データベース負荷ツール。 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負荷ツール。 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負荷ツール。 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。
OSDL DBT-4	TPC-App の簡易版データベース負荷ツール。TPC-App は、コスト面や測定内容が特化されにくい等、制約が多い TPC-W を踏まえ、B2B のアプリケーションサーバのアクティビティをシミュレーションし、Web ベースのアプリケーションサーバの処理能力を測定するベンチマークである。

※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。

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

DBT http://www.osdl.jp/lab_activities/kernel_testing/osdl_database_test_suite/

1.4.2. OSDL- DBT 1 (Open Source Development Labs Database Test1)

OSDL DBT-1 (Open Source Development Labs Database Test1) 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。この負荷ツールは 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、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。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利用できるように作成された。

TPC ベンチマークで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、DBT-1 による負荷テスト結果とは、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。

実際のシステム開発現場では、それらの規則を順守することはあまり現実的ではない。

従って、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。

1.4.2.1 ドキュメント中の表記について

DBT-1 の負荷は、14 種類のインタラクションにより構成される。ドキュメント中の図において、表 1.4-2 の略号を用いる場合がある。

表 1.4-2 インタラクション略号

インタラクション	略号	インタラクション	略号
Admin Confirm	AC	New Products	NP
Admin Request	AR	Order Display	OD
Best Sellers	BS	Order Inquiry	OI
Buy Confirm	BC	Product Detail	PD
Buy Request	BR	Search Request	SR
Customer Registration	CR	Search Results	SU
Home	HO	Shopping Cart	SC

また、本ドキュメント中では、対象となる RDBMS を明確にするため、次のような表

記としている。

DBT-1 [RDBMS 名] (ストアド | ODBC) 版

Ex) DBT-1 MaxDB (ストアド) 版

DBT-1 PostgreSQL (ストアド) 版

DBT-1 MySQL (ODBC) 版 ・ ・ ・ 今回作成

DBT-1 PostgreSQL (ODBC) 版 ・ ・ ・ 今回作成

1.4.3. OSDL- DBT 3 (Open Source Development Labs Database Test3)

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。

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 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。

TPC ベンチマークで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、DBT-3 による負荷テスト結果とは、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。

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、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。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。

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) ポータビリティ向上(→2.1)
- (2) 改変版 OSDL DBT-1 による RDBMS の評価(→2.2.2/2.2.3)
- (3) DBT-1 を利用した 32bit/64bit 環境での性能・信頼性評価 (→2.2.4)
- (4) DBT-3 を利用した 32bit/64bit 環境での性能・信頼性評価(→2.2.5)

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. ポータビリティ向上

2.1.1. OSDL DBT-1 の MySQL(ODBC)用改変

今回の改変作業では、ストアードプロシージャを使用せず、SQL 文を直接発行するモジュールを作成し、測定まで至る事ができた。但し、ポータビリティ向上のため、Linux 上で一般的と言い難い ODBC を採用したことや、MySQL の独自機能でシーケンス番号取得処理を実装する等、課題も残すこととなった。

2.1.2. OSDL DBT-1 の PostgreSQL(ODBC)用改変

MySQL(ODBC)用に改変されたモジュールをベースとして、PostgreSQL(ODBC)を使用するモジュールへの改変作業を行った。ストアードプロシージャから SQL 文直接呼出しへ変更する際、C 言語ヘッダファイルに SQL 文を集約し、制御部には手を加えない予定で作業を行った。しかし、RDBMS 固有の機能にて実装せざるを得ない機能があり、制御部を完全に同一コードを共有するまでには至らなかった。一部制御部への修正が必要であったが、PostgreSQL(ODBC)用の改変作業は完了し、性能測定を行う事ができるようになった。

2.1.3. ポータビリティ向上のまとめ

ストアードプロシージャを廃止し、SQL 文を直接呼び出す方式に切り替えたことと、インターフェースに ODBC を採用することで、ポータビリティを向上させることを目指し、MySQL(ODBC)用に改変を加えた後、改訂版をベースとして PostgreSQL(ODBC)への改変作業を実施した(図 2.1-1 DBT-1 モジュール構成参照)。改変そのものは完了したが、一部機能(ユニークなシーケンスの取得)に関しては、同一の SQL 文並びに制御構造を使用して実装するまでには至らなかった。これは、DBT-1 の前提である TPC-W の DB 構造をそのまま使用したためであり、今後 DBT-1 をより純粋な RDBMS 負荷ツールとして成長させていくためには、検討の余地が大いにあるといえる。とは言え、今回の改変作業にて、DBT-1 の対応 RDBMS は、MySQL(ODBC)と PostgreSQL(ODBC)の 2 種類が追加され、前期の成果である 4 種類の RDBMS と併せ合計 5 種類の測定を行う事が可能になった。また、未対応の RDBMS に対しても、対象 RDBMS の知識を有する技術者であれば、比較的容易に

移植作業を行う事ができる環境を構築できた。

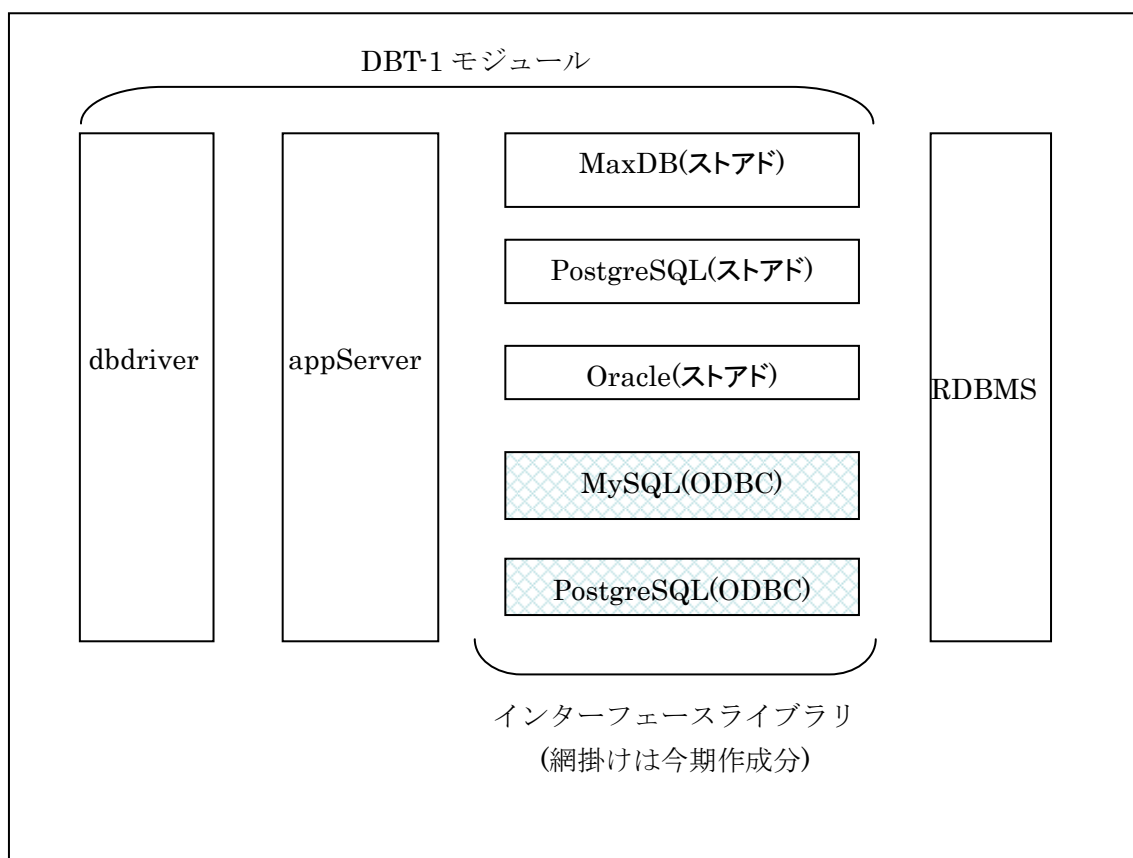


図 2.1-1 DBT-1 モジュール構成

尚、今回作成したモジュールは、IPA にて公開されるほか、OSDL ジャパンの協力を仰ぎ、メンテナとして OSDL にて更新を行う予定である。

2.2. 評価

2.2.1. 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、6 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、各章に於ける異なる環境間の個別評価の結果を、数値のみによって比較を行うべきではない。

2.2.2. OSDL DBT-1 MySQL(ODBC)版の評価

図 2.2-1 は、チューニング前後の MySQL(ODBC)版OSDL DBT-1 の擬似トランザクション数(BogusTransation/秒 以下BT/秒)推移をグラフ化したものである。チューニング前デフォルト状態に於いても、仮想ユーザ数(EmulateUser 以下EU)=1200 まで理論値*1通りの値となっており、チューニングすることでEU=1800まで上限を上げる事ができた。また、図 2.2-2 チューニング後 平均応答時間と合わせて考えると、今回のシステム構成におけるピーク処理性能は、毎秒 250 トランザクションで、平均応答時間は約 1 秒と見積もる事ができる。

*1 : DBT-1 における BT/秒理論値は、 $EU \div \text{ユーザオペレーション間隔(ThinkTime)}$ で算出される

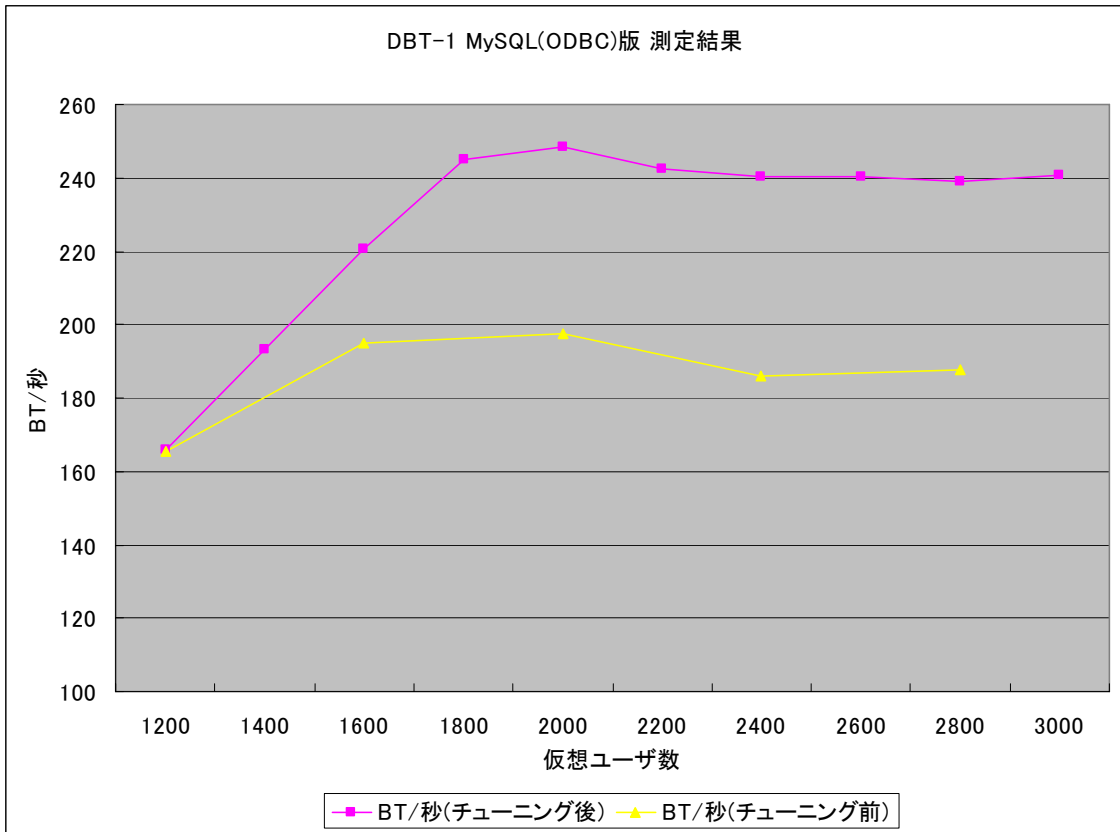


図 2.2-1 MySQL(ODBC)BT/秒推移

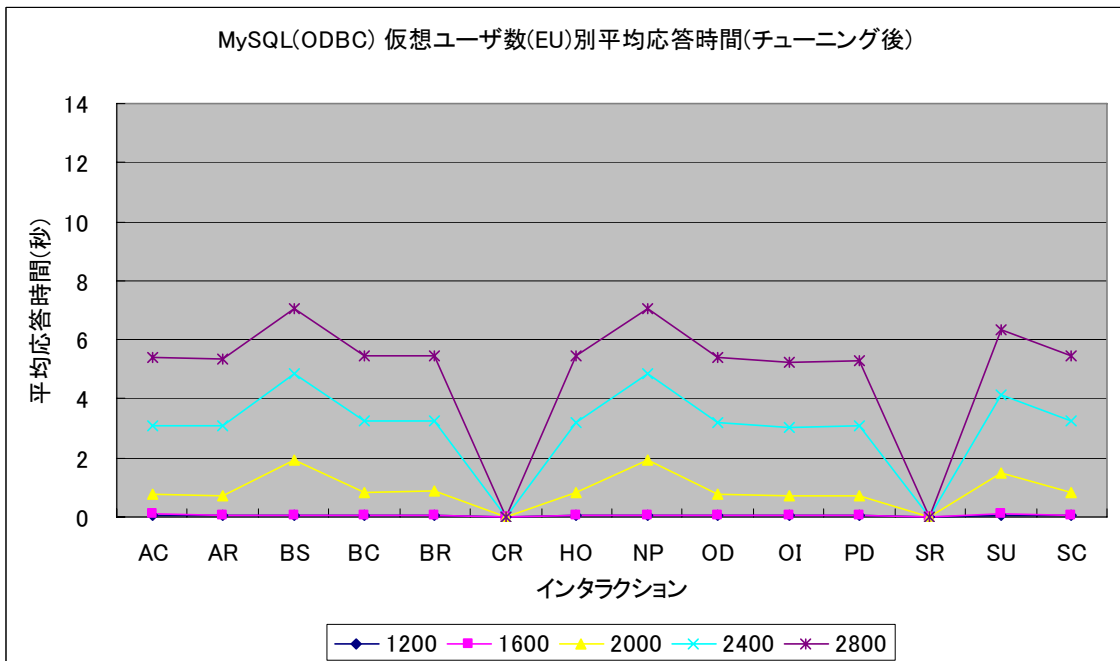


図 2.2-2 チューニング後 平均応答時間

※インタラクションは、第1章 表 1.4-2 インタラクション略号を参照のこと

今回の測定におけるチューニング作業では、パラメータ設定値の利用状況が取得できるパラメータ以外は試行錯誤による調整を行わざるを得なかった。また、一般的とされるパラメータであっても、効果が得られないものもあった。詳細な目安を導き出すには至らなかったが、DBT-1 のトランザクションパターンだけではなく、他の負荷ツールでも検証し、具体的指標を作成することが肝要であろう。

2.2.3. OSDL DBT-1 PostgreSQL(ODBC)版の評価

本評価に際し、PostgreSQL(ODBC)の評価ポイントを、ODBC 接続時の特性検証並びに、PostgreSQL のバージョンの違いによる測定結果の差分にあてて測定を行った。本節で使用したグラフは、全てチューニング前の測定結果を使用している。図 2.2-3 は、前年度後半において評価を行った PostgreSQL(ストアド)版と比較した性能遷移である。若干の違いはあるものの、最大 BT/秒が得られた EU や、グラフの形状に関して、大きな差は見られなかった。しかし、システムのリソース状況を vmstat や SQL ログなどから分析すると、同じインタラクションを比べた場合、PostgreSQL(ODBC)のほうが、実行される SQL 命令が少ないため、消費されるシステム・リソースが減少している。(詳細は第 7 章参照)

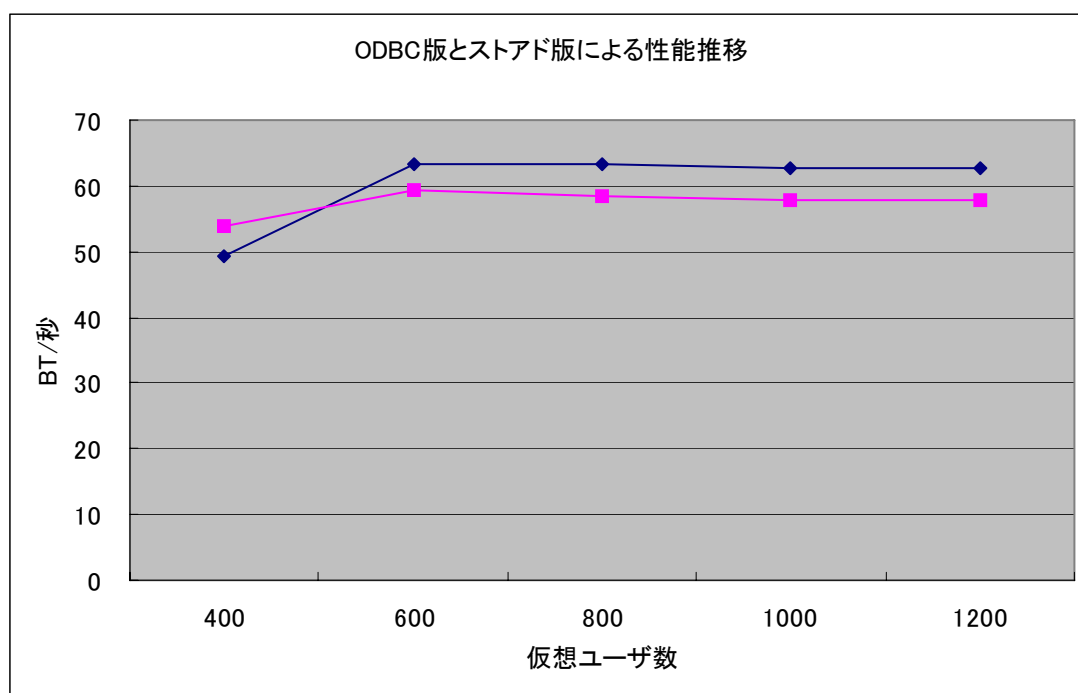


図 2.2-3 DBT-1 モデルの違いによる性能推移

図 2.2-4 は、PostgreSQL バージョンの違いによる性能推移である。バージョン 8.0 台のモジュールを使用した場合は、図 2.2-3 のグラフと性能及び特性で特筆すべき差は見受けられないが、バージョン 8.1 ベータでは、マルチ CPU 環境での効率が改善されたため、大幅な性能向上が確認できた。サーバに対するチューニングを施していないことから、チュー

ニングによる更なる性能向上が期待できる。マルチ CPU のハードウェア環境で PostgreSQL を運用する場合、バージョン 8.1 以降を使用するのが望ましいと言える。

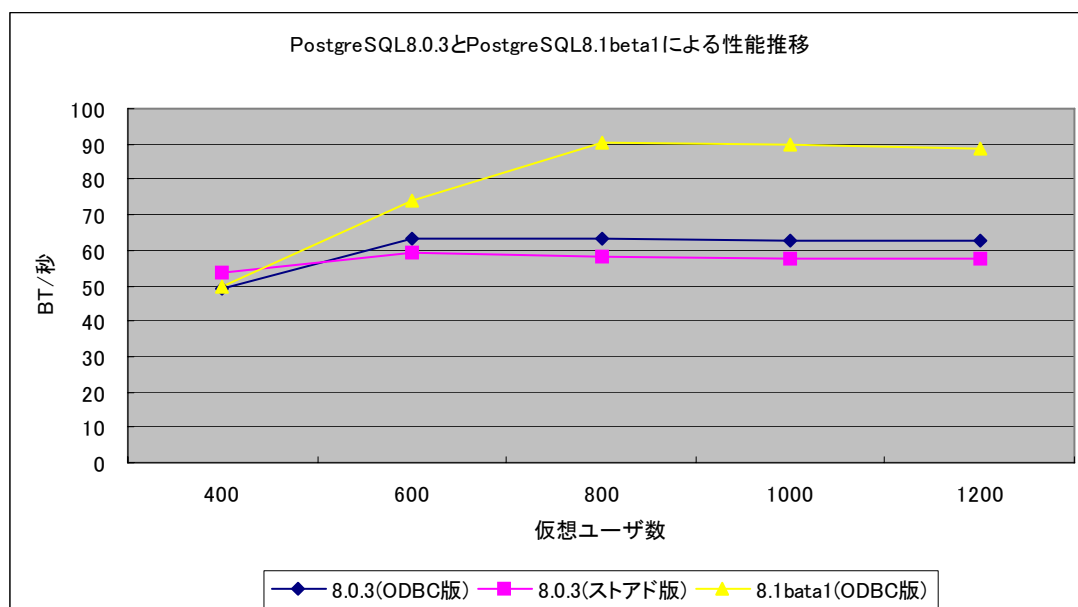


図 2.2-4 PostgreSQL バージョンの違いによる性能推移

今回の PostgreSQL の性能評価において、ネイティブインターフェースによるストアードプロシージャ呼び出しを行った場合と、ODBCによる SQL 文の直接実行を行った場合では、チューニング前のデフォルト状態においては、性能及び特性に差が無い事が確認された。但し、1 台のサーバで全てのアプリケーションを稼働させた場合との条件は付く。複数のサーバにアプリケーションを分散配置した環境での測定も今後必要になるであろう。また、バージョン 8.1 で大幅に性能が向上した事が確認できた。現在はベータ版しか入手できないが、正式版のリリースが待たれるところである。

2.2.4. OSDL DBT-1 MaxDB(ストアド)版 による 32bit/64bit の評価

今回の DBT-1 の評価に於いて、BT/秒は最大約 8%の差を確認することが出来た。(図 2.2-5 32bit OS 環境と 64bit OS 環境における BT 値の推移 参照)

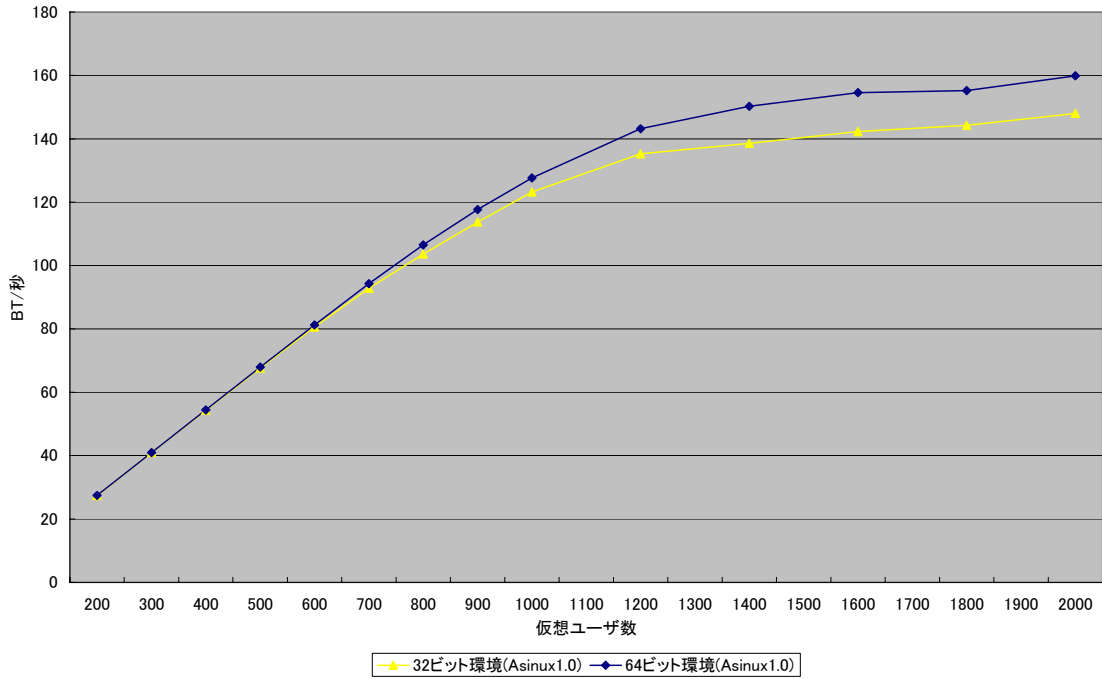


図 2.2-5 32bit OS 環境と 64bit OS 環境における BT 値の推移

この差は、ユーザプロセスの CPU 利用率が 32bit の方が消費しやすい事に起因していると思われる。(図 2.2-6 プロセス別 CPU 利用率 参照)

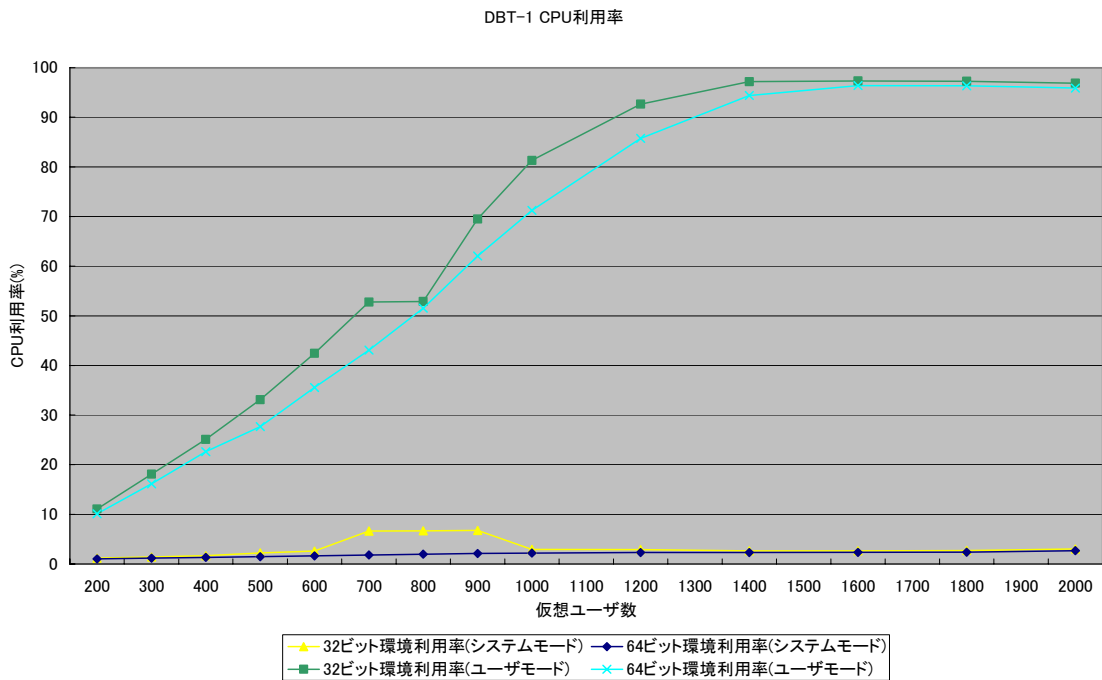


図 2.2-6 プロセス別 CPU 利用率

DBT-1 MaxDB(ストアド)版のみについて言及すると、64bit 対応のプロダクトで統一可能であれば、32bit 環境より性能向上が見込めると言える。

2.2.5. OSDL DBT-3 による 32bit/64bitの評価

DBT-3 による測定の指標値では、64bit の方が 5%程度良い数値を出した(図 2.2-7 指標値の比較 参照)。その他の値においても数値的には 64bit の方が良い値が得られたが、DBT-3 の処理内容とシステムの状態を関連付けして解析するまでには至らなかった。

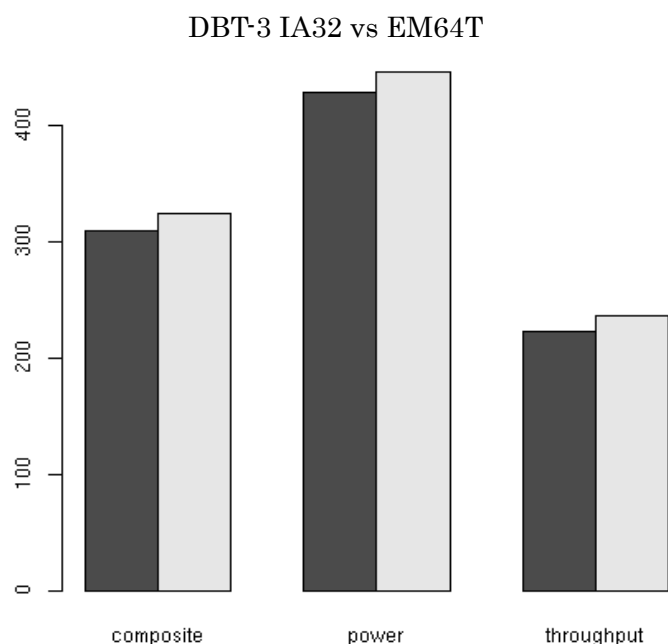


図 2.2-7 指標値の比較

DBT-3 に於いては、どこまで差を出すことが出来るか、今回の検証とは違った視点にて、チューニングを含めより詳細な検証が必要であると言える。

2.2.6. 評価のまとめ

今回の評価では、DBT-1 MySQL(ODBC)版、DBT-1 PostgreSQL(ODBC)版、DBT-1 MaxDB(ストアド)版、DBT-3 PostgreSQL を使い、それぞれ仕様の異なるハードウェア上にて、測定及び評価を行った。それぞれの評価において、チューニングのポイント、RDBMS バージョン或いはインターフェース間での性能の違い、32bit/64bit 環境での性能の違いを確認する事ができ、標準的な評価手順の確立ができた。また、各々のオープンソース RDBMS の適正な利用範囲を見極める上での判断基準の一部を提供できたものとする。

3. OSDL DBT-1 の MySQL 用改変

3.1. 現状の問題点

現状の OSDL DBT-1(以下 DBT-1)は RDBMS 毎に異なるストアードプロシージャ及び RDBMS ネイティブ関数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プロシージャ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プロシージャ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、結果のビジュアライズ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。(デフォルトアイソレーションレベルによる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) 検索リクエストインタラ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、共通インターフェースを利用できるよう ODBC による実装を行い、解決を目指す。

(6) に関しては、アイソレーションレベルを設定する関数呼び出しは追加するが、アイソレーションレベルの違いによる評価は行わない。(7)~(8)に関しては、DBT-1 自体の動向を見定める事とし、今回の改変には組み入れない。

3.2. 改変方針

3.2.1. 概要

設計方針に基づき、DBT-1 を改変する。具体的には ODBC インタラ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) 新規ディレクトリ `interfaces/odbc_com,include/inc_mysql` にモジュールを追加
- (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
- (4) 環境作成/起動シェルスクリプトの修正

3.2.1.1. データベース生成スクリプト

MaxDB や PostgreSQL では、関数“NEXTVAL()”にてユニークなシーケンスを取得する事が可能となっている。MySQL には当該関数がインプリメントされていない。そのため、該当するカラムを“auto_increment”属性で定義した。また、“orders”テーブルの“o_id”は、“shopping_cart”テーブルの“sc_id”をそのままセットするが、“orders”テーブルには事前にデータがロードされ、“shopping_cart”には初期データがロードされない。そのため、“shopping_cart”インサート時にキーが重複しないよう、初期化する必要がある。“shopping_cart”の“sc_id”に“orders 件数+1”の値を明示的に指定したダミーデータを初期値としてインサートすることで、このエラーが発生しないよう設定した。

3.2.1.2. SQL 文

“NEXTVAL()”にて取得したユニークなシーケンス番号をセットしていた項目を、“auto_increment”属性に変更したことにより、セットされた値を取得するため、“last_insert_id()”関数で取得するように変更した。

3.2.1.3. ODBC インタラクションプログラム

各インタラクションに対する処理を実装する。以下にインタラ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、各インタラクション処理関数を

```
int execute_XXXX(struct db_context_t *, struct XXXX t *)
```

の形式 (XXXX はインタラクション名) 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。
 - ① `SQLExecuteDirect`
SQL の実行。
 - ② `SQLBindCol`
パラメータの設定。SQL 条件として渡すデータを設定する。
 - ③ `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。
 - ④ `SQLCloseCursor` (選択処理のみ)
実行結果を破棄する。
- (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) インタ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。

3.2.1.4. ODBC インタクションプログラムヘッダ

各インタ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) 各インタ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。

3.2.1.5. SQL ヘッダ

各インタ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 = %lld"`

3.2.2. 構造

3.2.2.1. 変更前の構造

現行の MaxDB7.5 版 DBT-1 の実装概要を図 3.2-1 に示す。インタラクションライブラリでは ODBC 経由で MaxDB 上に実装されているストアードプロシージャをコールし、MaxDB は該当するストアードプロシージャを実施して値を返している。

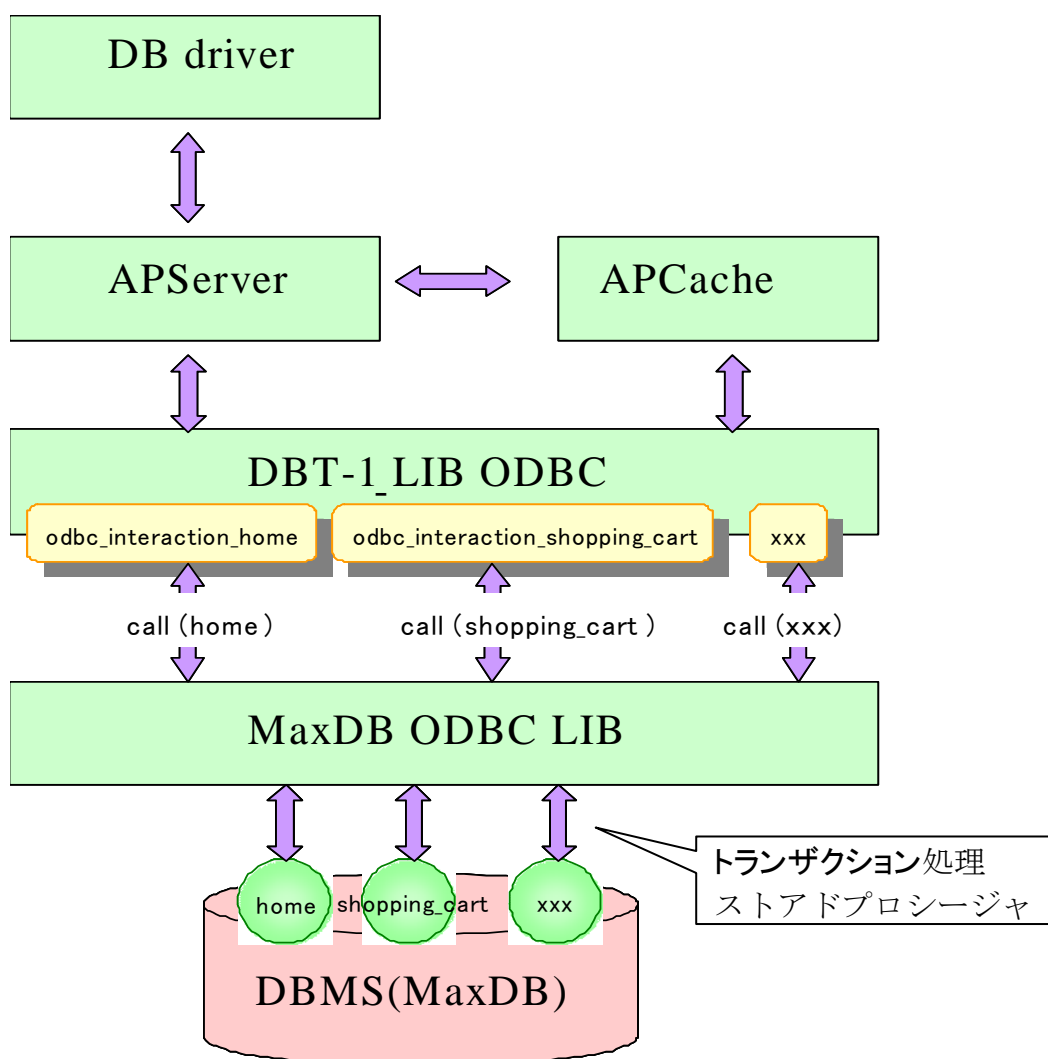


図 3.2-1 変更前の構造

注) call のカッコ内 home/shopping_cart は代表的なインタラクションをして明記したもの。その他のインタラクションは call(xxx) と表現した。

図 3.2-2 は今回の変更後の想定構造を示している。ストア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。

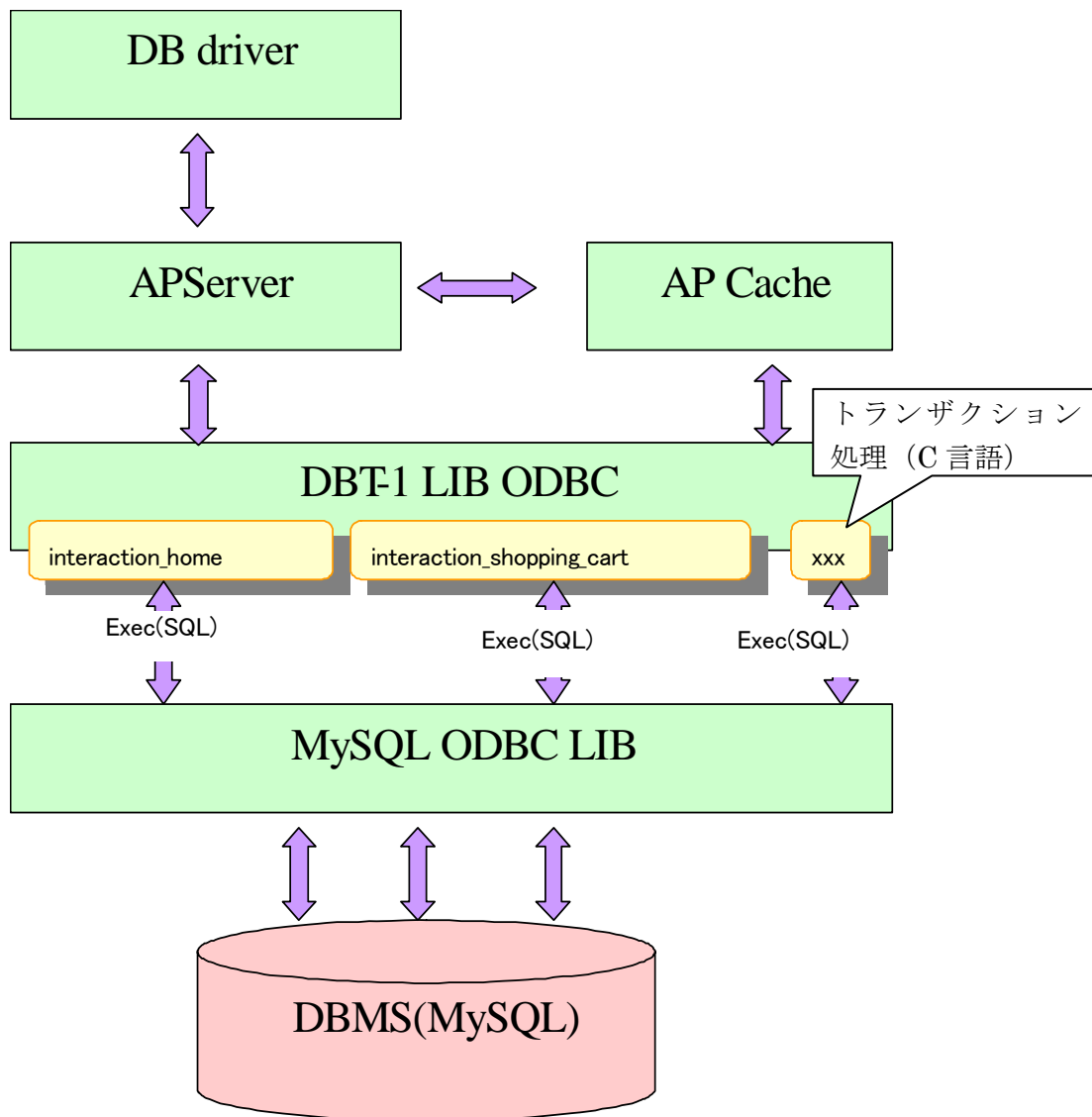


図 3.2-2 変更後の構造

3.2.2.2. 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/秒の遷移或いはBTファイル内の各インタラクションの平均時間をプロットする。

(1) 対象データ

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

BT/秒の遷移またはインタラクション毎の分布グラフを出力する。

※利用ツール：gnuplot



図 3.2-3 グラフ出力結果

3.3. 改変作業

3.3.1. スクリプト

環境構築及び測定用スクリプトには、MySQL コマンドへの変更を施した。表 3.3-1 に一覧を記す。

表 3.3-1 改変環境構築及び測定用スクリプト一覧

ファイル名	説明
add_aux_structure.sh	add_aux_structure.sql を実行するシェルスクリプト
add_aux_structure.sql	新製品/ベストセラー/検索結果/関連商品各テーブルを作成する。データはアイテムテーブルから抽出する
backup_db.sh	データベースのバックアップ
address.sql	address テーブルにデータをロードする SQL
author.sql	author テーブルにデータをロードする SQL
cc_xacts.sql	cc_xacts テーブルにデータをロードする SQL
country.sql	country テーブルにデータをロードする SQL
build_db.sh	一連の DB 作成処理
create_db.sh	データベースを生成するシェルスクリプト
create_fk.sh	create_fk.sql を実行するシェルスクリプト
create_fk.sql	外部参照キーを生成する SQL
create_indexes.sh	create_indexes.sql を実行するシェルスクリプト
create_indexes.sql	インデックスを生成する SQL
create_tables.sh	create_tables.sql を実行するシェルスクリプト
create_tables.sql	テーブルを生成する SQL
drop_db.sh	データベースを削除するシェルスクリプト
drop_tables.sh	drop_tables.sql を実行するシェルスクリプト
drop_tables.sql	テーブルを削除する SQL
load_db.sh	一連のデータロード処理を実行するシェルスクリプト
item.sql	item テーブルにデータをロードする SQL
order_line.sql	order_line テーブルにデータをロードする SQL
orders.sql	orders テーブルにデータをロードする SQL
reset.sql	テーブル状態を初期化する SQL
mysql_reset.sh	DBT-1 実行時 DB に加えた変更を元に戻す
db_stats.sh	DB ログ情報収集シェルスクリプト

3.3.2. ソースコード

作成するヘッダファイル及びソースファイルは、新たにサブディレクトリを作成し、格納する。また、今回はテストツール (interaction_test.c) で動作確認を行ったため、不足しているパラメータを補う等の改造も施すこととなった。表 3.3-2 にファイル一覧を記す。

表 3.3-2 改変ソースファイル一覧

ファイル名	説明	行数
DigSyl.h	ユーザ名/パスワード生成	19
InsertCust.h	顧客データ追加	70
addToSC.h	ショッピングカートデータ追加	61
createSC.h	ショッピングカートレコード作成	39
getCustInfo.h	顧客情報取得	49
getOrderItems.h	注文商品詳細情報取得	28
getPromoImages.h	販売促進商品画像取得	30
getSCDetail.h	ショッピングカート内商品詳細情報取得	28
getSCSubtotal.h	ショッピングカート商品小計計算	25
initOrderItems.h	注文商品データ初期化	20
initSCItems.h	ショッピングカートデータ初期化	20
interaction.h	インタラクション処理定義	44
interaction_admin_confirm.h	インタラクション管理者承認	35
interaction_admin_request.h	インタラクション管理者要求	28
interaction_best_sellers.h	インタラクションベストセラー	30
interaction_buy_confirm.h	インタラクション購買確認	88
interaction_buy_request.h	インタラクション購買要求	30
interaction_home.h	インタラクションホームページ	23
interaction_new_products.h	インタラクション新製品	29
interaction_order_display.h	インタラクション購買品目表示	43
interaction_order_inquiry.h	インタラクション購買品目紹介	25
interaction_product_detail.h	インタラクション商品詳細	31
interaction_search_request.h	インタラクション検索要求	23
interaction_search_results.h	インタラクション検索結果	43
interaction_shopping_cart.h	インタラクションショッピングカート追加	27
refreshSC.h	ショッピングカート再計算	33
updateSC.h	ショッピングカート更新	41
DigSyl.c	ユーザ名/パスワード生成	50
InsertCust.c	顧客データ追加	161
addToSC.c	ショッピングカートデータ追加	174

createSC.c	ショッピングカートレコード作成	98
getCustInfo.c	顧客情報取得	70
getOrderItems.c	注文商品詳細情報取得	60
getPromoImages.c	販売促進商品画像取得	64
getSCDetail.c	ショッピングカート内商品詳細情報取得	67
getSCSubtotal.c	ショッピングカート商品小計計算	47
initOrderItems.c	注文商品データ初期化	32
initSCItems.c	ショッピングカートデータ初期化	34
interaction.c	インタラクション処理	173
interaction_admin_confirm.c	インタラクション管理者承認	160
interaction_admin_request.c	インタラクション管理者要求	91
interaction_best_sellers.c	インタラクションベストセラー	107
interaction_buy_confirm.c	インタラクション購買確認	322
interaction_buy_request.c	インタラクション購買要求	66
interaction_home.c	インタラクションホームページ	55
interaction_new_products.c	インタラクション新製品	101
interaction_order_display.c	インタラクション購買品目表示	124
interaction_order_inquiry.c	インタラクション購買品目紹介	49
interaction_product_detail.c	インタラクション商品詳細	178
interaction_search_request.c	インタラクション検索要求	30
interaction_search_results.c	インタラクション検索結果	100
interaction_shopping_cart.c	インタラクションショッピングカート追加	86
refreshSC.c	ショッピングカート再計算	45
updateSC.c	ショッピングカート更新	100
	合計	3607
interaction_test.c	テストモジュール	10
graph.pl	グラフ生成スクリプト	514

3.3.3. 追加ディレクトリ構成

今回の改変作業では、3つのディレクトリを追加した。図 3.3-1 に追加したディレクトリ構成を記す。ヘッダディレクトリ、ソースディレクトリ及びスクリプトのディレクトリを追加した。

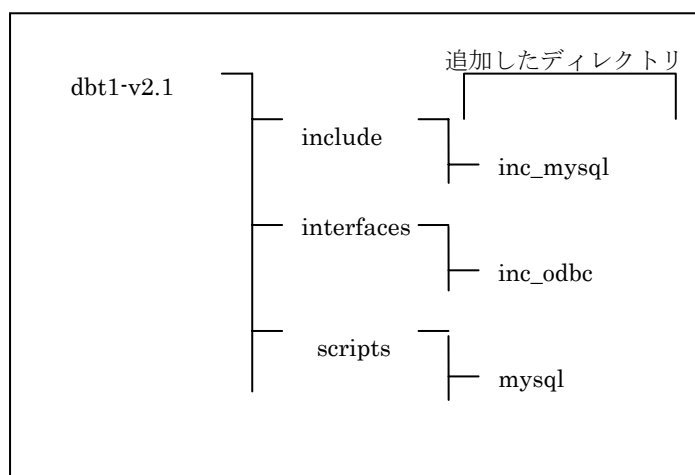


図 3.3-1 改変ディレクトリ構成

3.3.4. モジュール生成

モジュールの生成は、”dbt1-v2.1”ディレクトリにて、”make”コマンドを実行することで、生成される。IPAにて公開される dbt1-v2.1-MySQL-ODBC-1.0.tar.gz は、MySQL(ODBC)専用モジュールとなる。

3.3.5. 動作確認

動作確認は、DBT-1 に付属するツール、interaction_test を使用し MaxDB 版の結果と比較することで確認を行った。環境は、MaxDB と MySQL(ODBC)それぞれのデータベースを、同一のデータで生成することで、確認をしやすくした。また、乱数に依存する部分では、それぞれの SQL を個別に発行し、データの整合性を確認した。

3.4. まとめ

MySQL の ODBC 接続を使用し、ストアードプロシージャを使用せず、SQL 文を直接実行するコードへの改変作業を完了させ、DBT-1 による測定を実施する事ができた。手順並びに結果は、第 5/6 章を参照の事。今回の改変では、ODBC 接続を使用することで、インターフェースを統一し、ヘッダファイル中に記述される SQL 文のみを各 RDBMS 向けに動作確認を行うことで、ポータビリティを向上させることを目標とした。MySQL(ODBC)向け改変では、MaxDB で実装していたシーケンスの取り扱い手順では対応できず、MySQL 独自の機能を使用することとなった。結果として、他の RDBMS へのポーティングの際に、SQL 文のみ改変し制御コードを共通化する当初の目標とは、若干のズレを生じさせる結果となった。

ビジュアルイズツールは、従来手作業で行っていた結果の可視化作業を、自動化するもので、随時結果を確認できるようになった。

今回の改変においては、ODBC を使用しインターフェースを統一することで、ポータビリティを向上させ一定の成果はえられた。同様の考えに立てば JDBC 版も需要があるものと思われる。また、RDBMS の最大パフォーマンスを引き出すためには、ネイティブコネクションを使用したバージョンも必要になると思われる。

今回作成したモジュールは、IPA にて公開されるほか、OSDL ジャパンの協力を仰ぎ、メンテナとして OSDL にて更新を行う予定である。

4. OSDL DBT-1 の PostgreSQL 用改変

4.1. 改変の目的

3章で行った MySQL 用の OSDL DBT-1 (以下 DBT-1) 改変作業によって、データベース固有のストアードプロシージャによるトランザクションロジックが排除され、DBT-1 のポータビリティが向上している。ODBC のプロトコルによるストアードプロシージャを使用しない純粋な SQL の処理の流れは、基本的に各 RDBMS で共通である。MySQL 用に改変された DBT-1 のトランザクションプログラムは、従来のデータベース固有のストアードプロシージャを使用した DBT-1 に対して、ポータビリティが向上したと考えられる。ポータビリティの向上を確認するため、MySQL 用に改変された DBT-1 をさらに PostgreSQL 用に改変する。

改変した結果、PostgreSQL に対して DBT-1 は、ストアードプロシージャを利用した測定と、ODBC による直接 SQL の 2 つの形態の測定を実施できることになり、適用範囲が広がったことになる。

改変の主な目的を以下に列挙する。

- (1) ストアードプロシージャを使用しない DBT-1 ODBC プログラムのポータビリティを確認すること。
- (2) 純粋な SQL 処理での効率評価を可能とすること。
- (3) トランザクションを実装すること。

4.2. 改変方針

4.2.1. 概要

MySQL 用に改変された DBT-1 を PostgreSQL 用に改変し、ポータビリティが向上した DBT-1 に対して、PostgreSQL で作動できることを確認する。ポータビリティが向上されたことを確認するために、C 言語のトランザクションプログラムに、修正の必要がないまたは少ないことを確認する。MySQL と PostgreSQL の RDBMS としての差異は、基本的に PostgreSQL 用のヘッダファイルとして吸収する。ヘッダファイルでは、特に SQL 文法に差がある部分に関して MySQL 用の SQL から PostgreSQL 用の SQL に改変する。

従来の DBT-1 は、PostgreSQL7.4.x での作動を確認していたが、今回改変する DBT-1 は、PostgreSQL の現行最新のバージョン 8.0.3 を使用する。

ポータビリティは向上しているが、ポーティング後ソースは、修正なしに他のバージョンや他の RDBMS で動作することを保証しない。理由は、ポータビリティが向上したとはいえ、前述のヘッダファイルやデータベース作成スクリプトなどの修正が必要だからである。なお、DBT-1 の MaxDB(ストアード) 版、PostgreSQL (ストアード) 版、MySQL (ODBC)

版、PostgreSQL(ODBC)版のソースをひとつとすることは、本プロジェクト終了後の課題としている。

また、DBT-1 では、RDMBS の統計情報を収集しているが、実行時の統計情報（例えば、内部のヒープブロックの読み込み数など）は、RDBMS 固有の統計ツールの実行を必要とする。

改変する作業内容は、以下のとおり。

- (1) ODBC ヘッダファイルを MySQL 用から PostgreSQL 用に修正
- (2) SQL にあわせて、データベース生成スクリプトを修正
- (3) C 言語のソースプログラムの作動確認
- (4) configure/make 関連ファイルの修正

4.2.2. 改変ファイル一覧

DBT-1 MySQL(ODBC)版に対して、改変したファイルとその行数を表 4.2-1 一覧表に示す。具体的な改変内容については、次節以降で解説する。

表 4.2-1 改変ファイル一覧

ファイル名	説明	行数
Interaction_buy_confirm.h	インタラクシオン購買確認	4
GetCustInfo.h	顧客情報取得	1
InsertCust.h	顧客データ追加	5
createSC.h	ショッピングカートレコード作成	2
addToSC.h	ショッピングカートデータ追加	1
interaction_product_detail.h	インタラクシオン商品詳細	1
CreateSC.c	ショッピングカートレコード作成	20
InsertCust.c	顧客データ追加	25
build_db.sh.in	テーブルを作成する SQL スクリプト。	1
load_dbproc.sh.in	シーケンシャルを生成するスクリプト	1
create_tables.sh.in	表を生成するスクリプト	24
start_db.sh.in	データベース起動スクリプト	1

4.2.2.1. SQL ODBC プログラムヘッダファイル

(1) 日付の演算

DBT-1 MySQL(ODBC)版では、日付または時間を加算する場合、関数 `adddate()` または `addtime()` を用いる。PostgreSQL は、標準でそれらの関数が実装されていないため、DBT-1 PostgreSQL(ODBC)版では、加算記号 “+” で加算するように修正する。

(2) 文字列から TIMESTAMP 型への型変換

DBT-1 MySQL(ODBC)版では、文字列から TIMESTAMP 型の列への値の代入は、TIME 関数で型変換することができる。DBT-1 PostgreSQL(ODBC)版の場合、TIMESTAMP 関数で型変換する必要がある。

(3) ユニークなシーケンスの取得

DBT-1 PostgreSQL (ストアド) 版では、ストアドプロシージャの中で、”nextval()”関数でユニークなシーケンスを取得していた。DBT-1 MySQL(ODBC)版では、MySQL には当該関数が実装されていない。そのため、該当する列を”auto_increment”属性で定義し、セットされた値が必要な場合は、”last_insert_id()”関数で取得するように変更された。DBT-1 PostgreSQL(ODBC)版は、DBT-1 PostgreSQL (ストアド) 版と同様に nextval()関数を使用するように SQL を変更し直す。

(4) 関数 MAX

DBT-1 MySQL(ODBC)版のポーティングにおいて、最新のユニークなシーケンスを検索するため、customer 表の主キー列 c_id に対する関数 MAX(c_id)が、今回採用された。PostgreSQL バージョン 8 の場合、アクセスプランが、シーケンシャルスキャンによる全件読みによる検索となった。シーケンシャルスキャンによる全件読みの場合、データベースの容量により、処理効率が変わってしまう。DBT-1 の互換性を高めるためには、関数 MAX を変更すべきではないが、この SQL が、(3) の RDBMS 固有の機能に関連し使用されること、処理時間が長くなることでロック待ちなど他の SQL への影響の懸念があること、そして、DBT-1 PostgreSQL(ストアド)版と比較において現実のトランザクションモデルが想定している応答時間に近づけるため、アクセスプランをインデックススキャンになるように SQL を以下のように変更した。

なお、この問題は、次期 PostgreSQL で修正され、関数 MAX もインデックスを使用するアクセスプランを選択できる予定であるため、バージョン 8.1.x の正式リリースを待って、関数 MAX を使用することが可能になる。

修正前

```
SELECT MAX(c_id) FROM customer
```

修正後

```
SELECT c_id FROM customer ORDER BY c_id DESC limit 1
```

(5) NULL 値の取り扱い

shopping_cart_line 表へのアイテムの追加処理 (addToSC) において、レコードの数の合計を数えるために、関数 SUM が使用されている。PostgreSQL バージョン 8 では、対象レコードが存在しないときに結果は NULL 値を返し、NOT_FOUND が返されることはない。ストアドプロシージャ内では、NULL 値が返されてもエラーハンドリングをしていな

いため、トランザクションが終了することはないが、プログラム側に NULL 値が返されるとトランザクションプログラムの ODBC 関数に、NULL 値の考慮が必要となる。そのため、NULL 値をトランザクションプログラム側に返さないように SQL を変更した。

修正前

```
SELECT sum(scl_qty) FROM shopping_cart_line WHERE scl_sc_id=%lld;
```

修正後

```
SELECT sum(scl_qty) FROM shopping_cart_line  
WHERE scl_sc_id=%lld HAVING sum(scl_qty) IS NOT NULL;
```

4.2.2.2. データベース生成スクリプト

DBT-1 MySQL(ODBC)版の実装により、DBT-1 PostgreSQL (ストアド) 版で使用されていたストアドプロシージャの定義は不要になる。従来のスクリプトを用いてストアドプロシージャがデータベースに定義されても DBT-1 PostgreSQL(ODBC)版の実行には差し支えないが、実際にストアドプロシージャを使用していないことを確認するために、データベース生成スクリプトにおいてストアドプロシージャが定義されないように修正した。

また、PostgreSQL バージョン 8 から、接続パラメタとしての `tcpip_socket` の設定が `listen_addresses` と変更になり、デフォルトで `localhost` での TCP/IP 接続を許可する設定となった。PostgreSQL バージョン 7 以前では、スクリプトに `tcpip_socket` を明示的に指定し設定する必要があるが、バージョン 8 では、不要である。バージョン 8 に合わせて、スクリプトパラメタを認識しないように修正した。同様に、`tcpip_socket` を明示的に指定する DBT-1 用の設定パラメタ `db_param` が無効になるように修正した。

DBT-1 のマニュアルでの手順では、表定義がなされる前に GRANT 文が実行されエラーとなっていたため、表定義が実行されたあとに、アクセス権限を付与するように修正した。

4.2.2.3. C 言語のソースプログラム (ODBC トランザクションプログラム)

各トランザクションプログラムの ODBC の実装は、基本的に MySQL と PostgreSQL とで共通の SQL 処理で実施することができるため、修正を極力加えず、MySQL 用ヘッダファイルから PostgreSQL 用ヘッダファイルの置き換えで対応することを目標にした。しかしながら、4.2.2.1(3)のユニークなシーケンスを獲得する手段の違いより、一部トランザクションプログラムの SQL を実行する順番を変えた。MySQL の場合、`autoincrement` 属性の列に代入する値が 0 の場合、自動的に最後の値に 1 を加えてレコードが追加されるが、PostgreSQL にはこのような機能がない。そのため、DBT-1 PostgreSQL (ストアド) 版と同様に先にシーケンスを採取してから挿入するように変更した。

今回のポーティングでは、トランザクションを実装し、DBT-1 MySQL(ODBC) 版と同様に `autocommit off` でプログラミングしている。`autocommit on` に変更する場合、`interaction.c` での `connection` 部分を変更する必要があるが、今回は、測定を実施していな

い。

```
rc = SQLSetConnectAttr(odbc->hdbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF, 0);
```

MySQL は、ODBC からの問い合わせが連続する場合、SQLCloseCursor を実行しなくても暗黙に実行されるが、PostgreSQL は、明示的に実行する必要がある。MySQL も明示的に実行することに問題はないため、正常ケースにおいては、MySQL も PostgreSQL も SQLCloseCursor を実行するように実装している。しかしながら、エラーがあっても次のトランザクションが継続的に実行されるように、DBT-1 PostgreSQL(ODBC)版では、エラー処理において SQLCloseCursor が実行されるように改変している。

4.2.2.4. configure/make 関連ファイル

PostgreSQL用のpsqlodbcのライブラリを参照するようにリンカオプションを修正した。また、unixODBCがインストールされているかどうかをチェックするために、sql.h sqlxext.h sqltypes.h がシステムにインストールされているか、また、SQLDriverConnect 関数がライブラリ参照内に含まれているかを検証するように修正した。

DBT-1 PostgreSQL (ストアド) 版が必要としていた autoconf を排除し、DBT-1 MySQL(ODBC)版と同様に新規に作成されたファイルを正しくコンパイルされるようにコンパイルするファイルとインクルードファイルの構成を修正した。

この改変により、IPAにて公開される、dbt1-v2.1-PostgreSQL-ODBC-1.0.tar.gz は、PostgreSQL(ODBC)専用モジュールとなる。

4.2.3. 構造

4.2.3.1. 変更前の DBT-1 MySQL(ODBC)版構造

ポーティング結果を改めて比較するため、3章で示された、PostgreSQL 変更作業の対象となった DBT-1 MySQL(ODBC) 版の DBT-1 のトランザクション実装概要を図 4.2-1 に示す。トランザクション管理サーバ層では MySQL の ODBC ライブラリを経由して、SQL を発行している。

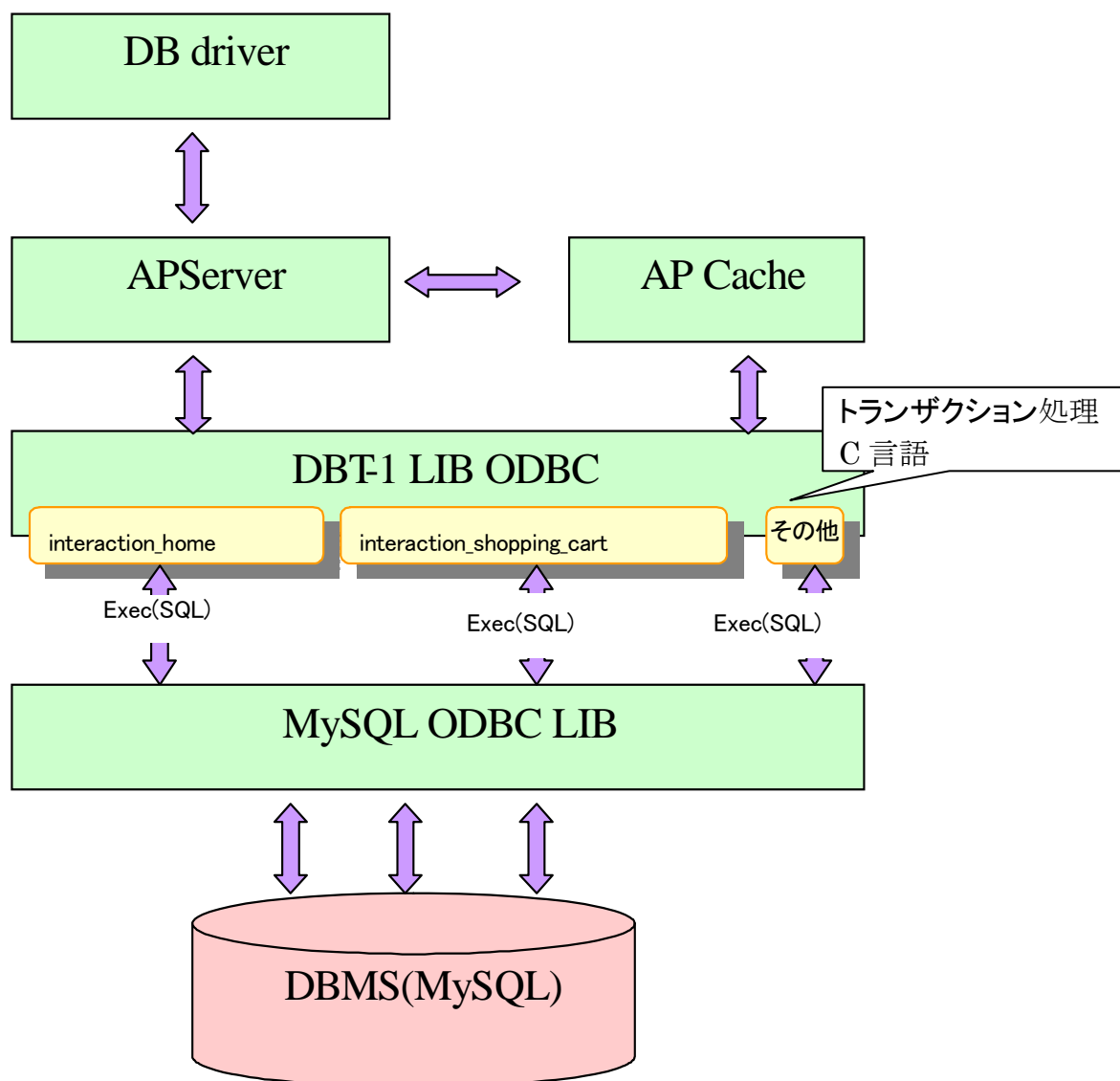


図 4.2-1 ポーティング前 DBT-1 MySQL(ODBC)版の構造

4.2.3.2. DBT-1 PostgreSQL(ストアド)版の構造

同様にポーティングの対比を示すため、DBT-1 PostgreSQL (ストアド) 版の想定構造を図 4.2-2に示す。C言語のプログラムよりネイティブな関数libpqインターフェースにより、ストアドプロシージャを呼び出し、結果を取得する。

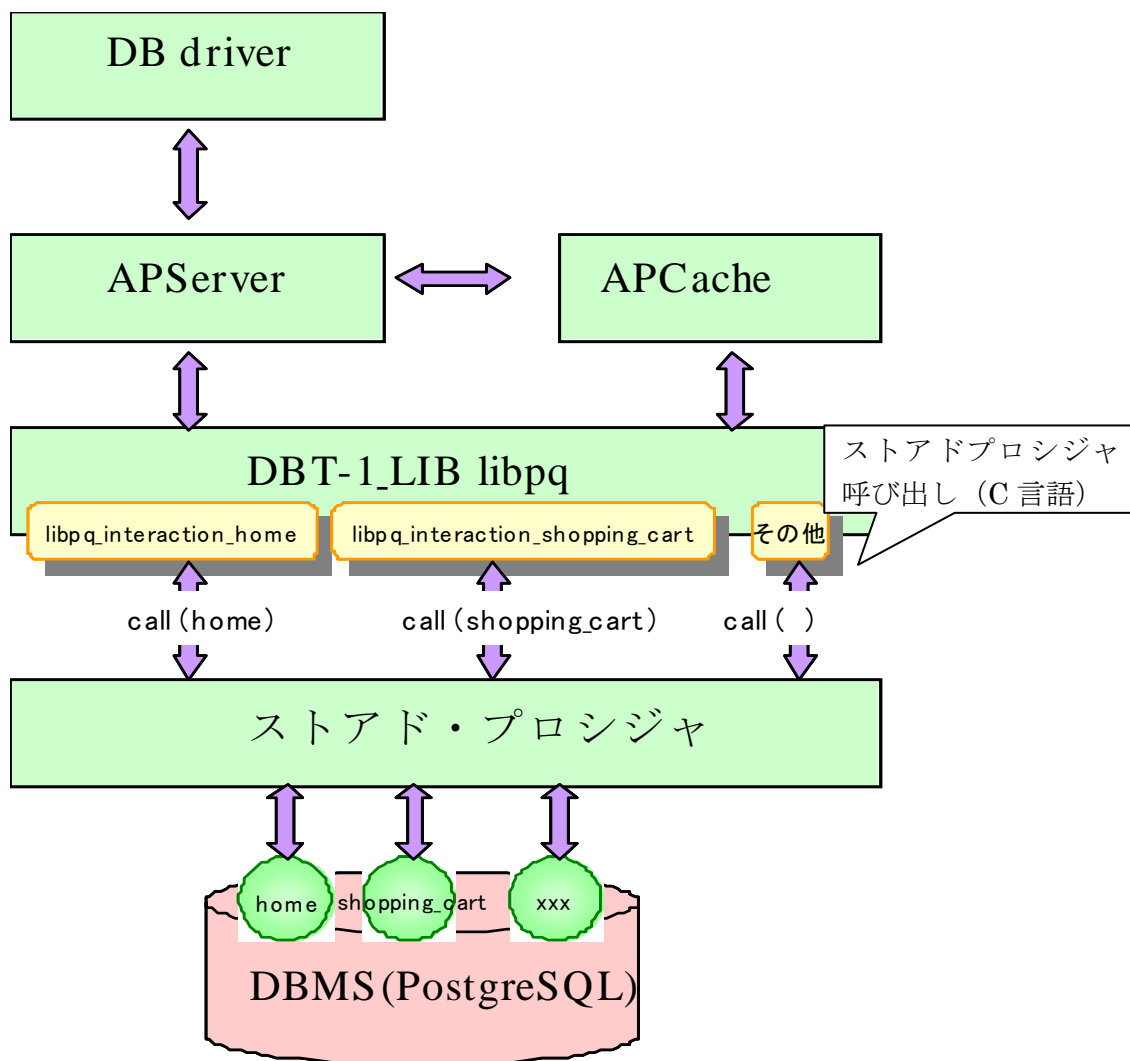


図 4.2-2 DBT-1 PostgreSQL(ストアド) 版の構造

4.2.3.3. PostgreSQL (ODBC)版用ポーティング後の構造

図 4.2-3 は今回の改変後の想定構造を示している。DBT-1 MySQL(ODBC)版と同様に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SQLを発行し、結果セットを取得する。

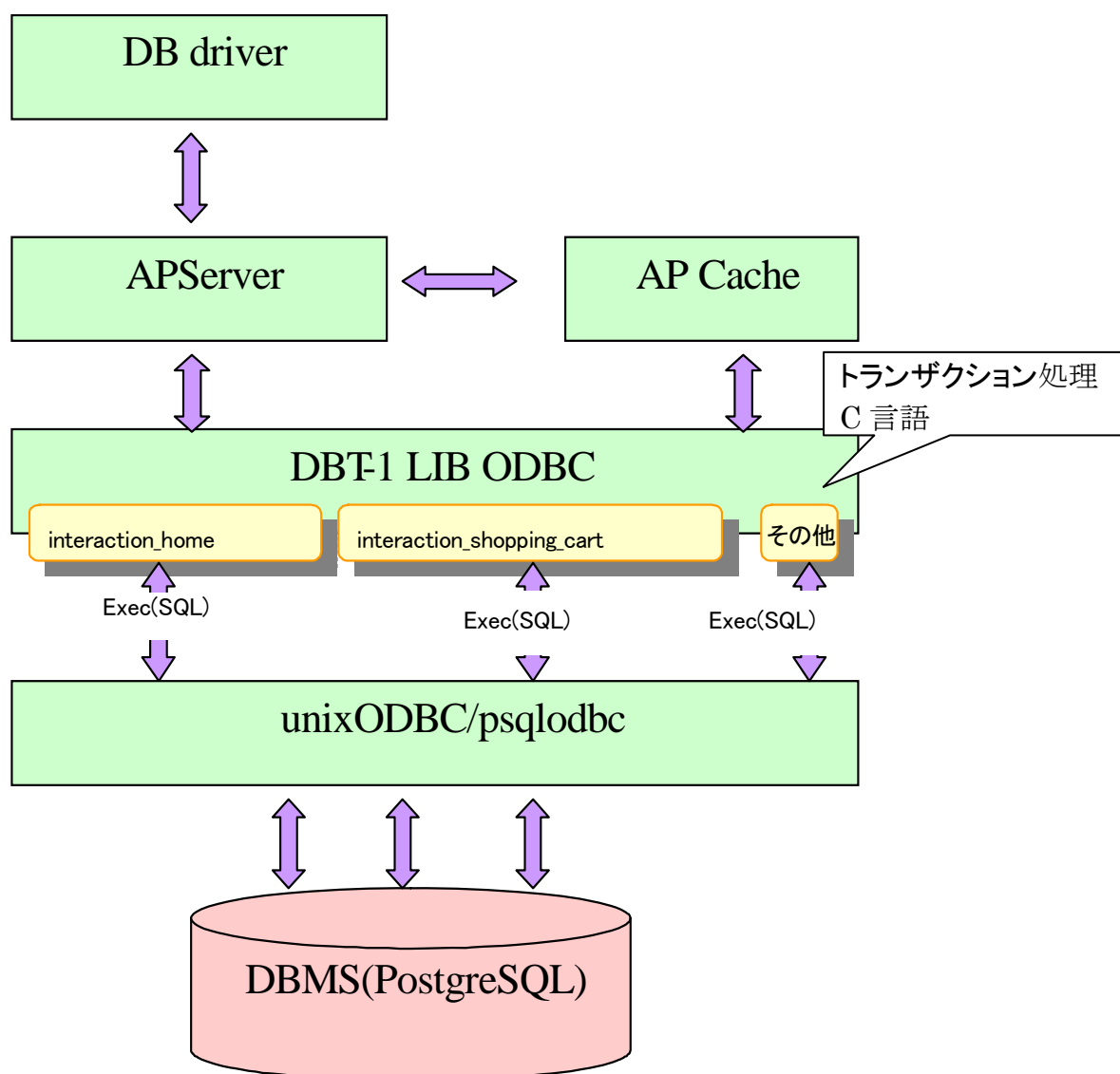


図 4.2-3 ポーティング後の構造

4.2.4. 動作確認

動作確認は、DBT-1 に付属するツール、`interaction_test` を使用し、インタラクションごとに DBT-1 PostgreSQL (ストアード) 版の結果と比較し確認した。データベースは、それぞれの DBT-1 で、同じ PostgreSQL のデータベースを利用した。また、乱数に依存する部分では、それぞれの SQL を適宜個別に発行し、確認した。

4.3. まとめ

- PostgreSQLを対象として、ストアードプロシージャを使用せず、ODBC接続によるSQLを直接発行するワークロードの性能測定を行うことができた。ワークロードの目的により、DBT-1は、PostgreSQLに対して、ストアードプロシージャによるトランザクションと純粋なSQLによるトランザクションの2種類のワークロードを測定できることとなる。
- 今回の改変作業によって、DBT-1のトランザクション実行部分のプログラムとそのSQLのポータビリティは、向上した。ヘッダに定義されたSQLを変更することで多くのRDBMSに対応することができるが、SQLの文法やシリアルナンバーの管理にRDBMS独自仕様を含むことを避けることができなかった。さらなるポータビリティ向上のためには、RDBMSに依存しない仕組みをDBT-1に組み込む必要があるが、今後のRDBMS接続の主流となることでのJDBC版と合わせて、今後の課題である。
- RDBMSによっては、SQLの互換性があっても、効率的に実用にならないアクセスプランを選択する場合がある。そのようなSQLは、必要に応じて現実的なSQLに改変する必要がある。
- 今回改変したDBT-1 PostgreSQL (ODBC)版モジュールは、IPAにて公開されるほか、OSDLジャパンの協力を仰ぎ、メンテナとしてOSDLにて更新を行う予定である。
OSDL DBT シリーズのトップページ URL :

http://www.osdl.jp/lab_activities/kernel_testing/osdl_database_test_suite/

5. OSDL DBT-1/3 測定手順

この章では、OSDL DBT-1/3 について、第 6 章以降の測定を実施するための手順について記述する。各測定には、オペレーティングシステム/ミドルウェア/負荷ツールがそれぞれ必要であり、各対象に応じてそれぞれを適宜組み合わせて、環境を構築する。表 5-1 に測定環境と対応する段落の対応表を記す。

表 5-1 構成組合せ

負荷ツール	使用 RDBMS	環境設定	ミドルウェア	負荷ツール	測定手順
OSDL DBT-1	MaxDB(StoredProc)	5.1	5.2.1	5.3.1/2	5.4
	MySQL(ODBC)	5.1	5.2.2/3	5.3.1/2	5.4
	PostgreSQL(ODBC)	5.1	5.2.4	5.3.1/2	5.4
OSDL DBT-3	PostgreSQL	5.1	5.2.4	5.3.3	5.5

5.1. 環境設定

5.1.1. Linux のインストール

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ディストリビューションのインストレーション・マニュアル等に従うこと。

対象ディストリビューション： MIRACLE LINUX V3.0 Asianux Inside

注)DBT-1 の測定には ssh、gcc、sysstat、libstdc++6.2.2 PostgreSQL 用には、autoconf2.59 が必要。また、グラフ作成用に、gnuplot 4.0 が別途必要となる。

5.1.2. DBT-1 用オプション設定

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、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数が少ないと、スレッドを生成しても DB 接続に失敗する。そのため、設定ファイル(/etc/security/limits.conf)に定義を追加し、上限を引き上げておく必要がある。

/etc/security/limits.conf

```
* soft nfile 4096
* hard nfile 65536
```

※“nfile”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。

※MIRACLE LINUX の場合、hard はデフォルトで 65536 に設定されているため、変更は不要。

5.1.3. MaxDB 用オプション設定

MaxDB の測定を行う、appServer から DB への接続時、セマフォ不足が発生することがあるため、以下のコマンドにて、セマフォを拡張する。

```
$ 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。

5.2. ミドルウェア

5.2.1. MaxDB のインストール

5.2.1.1. ダウンロード

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。

5.2.1.2. 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
```

5.2.1.3. 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。

5.2.2. MySQL のインストール

5.2.2.1. ダウンロード

MySQL AB(<http://www.mysql.com/>)より、アーカイブをダウンロードする。今回は、rpm パッケージではなく、tarballからの構築とする。

パッケージには、Standard/Max/Debug の 3 種類があり、それぞれのアーキテクチャ毎に用意されている。今回は Max パッケージを使用する。表 5.2-1 にパッケージの説明を記す。

表 5.2-1 パッケージ説明

パッケージ	説明
Standard	InnoDB を含んだ標準的なパッケージ
Max	Berkeley DB や一般用途以外の機能を含んだパッケージ
Debug	デバッグコードを含んだパッケージ。業務用途には推奨されない

5.2.2.2. MySQL の解凍

tar コマンドを使用し、ダウンロードした tarball を /usr/local ディレクトリに展開する。

```
# cd /usr/local
# tar -xzf 格納ディレクトリ/mysql-max-5.0.07-beta-linux-i686.tar.gz
```

5.2.2.3. 環境設定

バイナリインストールの手順は、"INSTALL-BINARY"ファイルに記載されている。

```
# groupadd mysql
# useradd -g mysql mysql
# cd /usr/local
# gunzip < /PATH/TO/MYSQL-VERSION-OS.tar.gz | tar xvf -
# ln -s FULL-PATH-TO-MYSQL-VERSION-OS mysql
# cd mysql
# scripts/mysql_install_db --user=mysql
# chown -R root .
# chown -R mysql data
# chgrp -R mysql .
# bin/mysqld_safe --user=mysql &
```

※Linux ディストリビューションによっては、ユーザ/グループが予め用意されている場

合がある。その場合は、MySQL の設定ファイルも”/etc/my.cnf”に存在する場合があるので、適宜編集を行う。

5.2.3. MyODBC のインストール

5.2.3.1. MyODBC のダウンロード

MySQL AB(<http://www.mysql.com/>)より、アーカイブをダウンロードする。今回は、複数のバージョンを使用しやすくするため、rpmパッケージではなく、tarballからの構築とする。

5.2.3.2. MyODBC のインストール

tar コマンドを使用し、ダウンロードした tarball を/usr/local ディレクトリに展開する。展開後、シンボリックリンクを張る。

```
# cd /usr/local
# tar -xzvf 格納ディレクトリ/MyODBC-3.51.10-pc-linux-i686.tar.gz
# ln -s /usr/local/MyODBC-3.51.10-pc-linux-i686 /usr/local/MyODBC
```

また、odbc.ini ファイルに以下の内容を記述する。ホスト名は、適宜変更する事。

```
[ホスト名:DBT1]
Driver           = /usr/local/MyODBC/lib/libmyodbc3.so
Description      = OSDL-DBT-1
Host             = localhost
user            = dbt
password        = dbt
database        = DBT1
server          = localhost
port            = 3306
socket          = /tmp/mysql.sock
```

5.2.3.3. iODBC のインストール

MySQL(ODBC)版 OSDL DBT-1 をコンパイルするために、SQL 関連のヘダーファイルが必要となるため、iODBC のソースのみインストールを行う。tar コマンドを使用し、ダウンロードした tarball を/usr/local ディレクトリに展開する。展開後、シンボリックリンクを張る。

```
# cd /usr/local
```

```
# tar -xzvf 格納ディレクトリ/libiodbc-3.52.2.tar.gz
# ln -s /usr/local/libiodbc-3.52.2.tar.gz /usr/local/libiodbc
```

5.2.4. 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るが、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。

今回の測定開始時点で、使用したバージョンは、7.4.6, 8.0.0beta5, 8.0.3, 8.1beta1 であったためそれぞれ適切な tarball ファイルを、入手する。複数のバージョンを同時にインストールする場合、使用するディレクトリが競合しないように工夫する必要がある。

root ユーザで、PostgreSQL の所有者となる Linux ユーザとして、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-x.x.x
# chown pgsql /usr/local/src/postgresql-x.x.x
# 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。x.x.x は、使用する tarball のレベルである。

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

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
```

5.2.4.1. PostgreSQL 用の UnixODBC のインストール

DBT-1 PostgreSQL(ODBC)を測定する場合、UnixODBC ドライバ・マネージャをインストールする必要がある。

Linux 上で ODBC によるプログラムを使用するためには、ODBC ドライバ・マネージャと PostgreSQL 用の ODBC ドライバをインストールする必要がある。ここでは、ODBC ドライバ・マネージャとして UnixODBC を使用する。ソースコードは、以下の URL より入手した。

<http://www.unixodbc.org/>

```
$ su root
# mkdir /usr/local/src/unixODBC-2.2.11
# chown pgsq1 /usr/local/src/unixODBC-2.2.11
# su - pgsq1
$ cd /usr/local/src
$ tar xzf /tmp/unixODBC-2.2.11.tar.gz
$ cd unixODBC-2.2.11
$ ./configure
$ make
$ su root
# make install
```

5.2.4.2. psqlodbc ドライバのインストール

DBT-1 PostgreSQL(ODBC)を測定する場合、PostgreSQL 用 ODBC ドライバを unixODBC ドライバ・マネージャに対応するようにインストールする必要がある。psqlodbc ドライバは、デフォルトで unixODBC ドライバ・マネージャに対応している。最新の psqlodbc ドライバを、以下の PostgreSQL 関連プロジェクト・コミュニティ Gborg プロジェクトから入手した。

入手元 <http://gborg.postgresql.org/project/psqlodbc/projdisplay.php>

```
# mkdir /usr/local/src/psqlodbc-08.00.0102
# chown pgsq1 /usr/local/src/psqlodbc-08.00.0102
# su - pgsq1
$ cd /usr/local/src
$ tar xzf /tmp/psqlodbc-08.00.0102.tar.gz
```

```
$ cd psqlodbc-08.00.0102
$ ./configure
$ make
$ su root
# make install
```

DBT1 の C 言語のプログラムが ODBC 経由で PostgreSQL に接続を行なうための設定を行なう。pgsql ユーザで、そのホームディレクトリに下記内容の .odbc.ini ファイルを作成する。

```
[ODBC Data Sources]
DBT1 =Postgres DBT1
[DBT1]
Servername = localhost
Database = DBT1
Driver = /usr/local/lib/psqlodbc.so
Port = 5432
ReadOnly=No
```

5.3. 負荷ツール

5.3.1. DBT-1 用の事前環境設定

5.3.1.1. ユーザの作成

root ユーザでログインし、dbt 用のユーザを作成する。

MaxDB,MySQL の場合は、ここでは ユーザ ID=dbt グループ=dbt パスワード=dbt としている。

PostgreSQL では、必ず PostgreSQL のスーパー・ユーザ ID=pgsql、グループ=pgsql パスワード=pgsql で実行すること。

MaxDB, MySQL の場合

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

5.3.1.2. 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。

5.3.1.3. MaxDB,MySQL

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

5.3.1.4. PostgreSQL

```
pgsql ALL=(ALL) NOPASSWD: ALL
```

5.3.1.5. ユーザ環境設定

DBT-1 実行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。

(1) MaxDB

dbt 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
```

(2) MySQL(ODBC)

dbt 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=/usr/local/mysql/bin:. : $PATH
export LD_LIBRARY_PATH=/usr/local/MyODBC/lib:/usr/local/mysql/lib
export ODBCINI=/usr/local/etc/odbc.ini
export SID1=DBT1
```

(3) PostgreSQL(ODBC)

pgsql 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 PGDIR=/usr/local/pgsql
export PATH=$PATH:$PGDIR:$PGDIR/bin
export PGUSER=pgsql
export PGDATA=/mydata/pgsql/data
export SID1=DBT1
export LD_LIBRARY_PATH=/usr/local/lib:/usr/local/pgsql/lib
```

5.3.2. ssh の準備

5.3.2.1. MaxDB MySQL

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/dbt/.ssh/id_rsa):
Created directory '/home/dbt/.ssh'.
Enter passphrase (empty for no passphrase):dbt
Enter same passphrase again:dbt
Your identification has been saved in /home/dbt/.ssh/id_rsa.
Your public key has been saved in /home/dbt/.ssh/id_rsa.pub.
The key fingerprint is:
ac:16:14:89:01:ee:31:1c:da:1d:b3:5b:84:b8:2d:68 dbt@localhost. localdomain

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

5.3.2.2. PostgreSQL

ssh を使用しないため、準備は不要である

5.3.3. DBT-1 のインストール

現段階では、各 RDBMS 対応版は統合されていない。そのため、新規に対応した MySQL(ODBC)及び PostgreSQL(ODBC)版は独立した tarball として提供されている。以降に、各 RDBMS 版 OSDL DBT-1 モジュールの作成方法を記す。

5.3.3.1. MaxDB 版

(1) DBT-1 のダウンロード

dbt 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)

(2) DBT-1 の解凍

次のコマンドで DBT-1 を展開する。

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

(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
```

5.3.3.2. MySQL(ODBC)版

(1) DBT-1 のダウンロード

dbt ユーザでログインする。

MySQL(ODBC)用 DBT-1 ソースをダウンロードする(IPA サイトより)

(2) DBT-1 の解凍

次のコマンドで DBT-1 を展開する。

```
$ tar -xzf dbt1-v2.1-MYSQL-ODBC-1.0.tar.gz
```

(3) DBT-1 のコンパイル

コンパイルする。

```
$ cd ~/dbt1-v2.1
$ make
```

5.3.3.3. PostgreSQL(ODBC)版

(1) DBT-1 のダウンロード

pgsql ユーザでログインする。

PostgreSQL(ODBC)用 DBT-1 ソースをユーザのホームディレクトリにダウンロードする
(IPA サイトより)

(2) DBT-1 の解凍

次のコマンドで DBT-1 を展開する。

```
$ tar -xzf dbt1-v2.1-PostgreSQL-ODBC-1.0.tar.gz
```

(3) DBT-1 のコンパイル

コンパイルする。

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

5.3.4. 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。

5.3.4.1. 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/osdbt
```

PC-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
```

5.3.4.2. 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`」の出力に余分な改行が入るので、`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
```

5.4. DBT-1 評価手順

5.4.1. 前提条件

MIRACLE LINUX の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。

5.4.2. テストデータの作成

5.4.2.1. データ用ディレクトリの作成

MaxDB,MySQLは、dbt ユーザでログインする PostgreSQLは、pgsql 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
```

5.4.2.2. データの生成

次のコマンドでテスト用の基礎データ(テキストファイル)の生成を行う。

MaxDB,MySQL

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

PostgreSQL

```
$ cd ~/dbt1-v2.1/datagen
$ ./datagen -d PGSQL -i 10000 -u 1000 -p /home/pgsql/data
```

オプションの解説

- d : データベースタイプ(SAPDB または PGSQL)
- 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。

5.4.3. データベースの作成

5.4.3.1. MaxDB

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。

5.4.3.2. MySQL(ODBC)

次のコマンドで MySQL 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/mysql  
$ ./build_db.sh
```

5.4.3.3. PostgreSQL(ODBC)

root ユーザでデータベース用のディレクトリを作成する。

```
# mkdir -p /mydata/pgsql/data
# chown -R postgres /mydata/pgsql
```

次のコマンドで PostgreSQL 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。

```
# su - postgres
$ initdb --no-locale --encoding =EUC_JP
$ cd ~/dbt1-v2.1/scripts/postgresql
$ ./build_db.sh ' -c tcp_socket=on ' 0 0
```

‘(シングルコーテーション)の前後には、空白を必ず挿入し、実行すること。

5.4.4. パラメータの設定

5.4.4.1. 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、実際に計測を行う仮想ユーザ数に修正

MaxDB,MySQL の `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/dbt/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                            ← RDBMS 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/dbt/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/dbt/dbt1-v2.1/dbdriver	

PostgreSQL の場合の各行は以下のとおり

[database]	
#hostname instance username password	
localhost:DBT1:pgsql:pgsql	
[cache]	
#hostname port dbconnections items appCache_executable_directory	
localhost:9999:5:10000:/home/pgsql/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	← PostgreSQL コネクション数
20	
#transaction_queue_size	← dbdriver 通信バッファ
1500	
#transaction_array_size	← 各処理バッファ
1500	
#items	← 生成アイテム数
10000	
#appServer executable directory	← appServer 実行ディレクトリ
/home/pgsql/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>400</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
2400	
#dbdriver executable directory	← dbdriver 実行ディレクトリ
/home/pgsql/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、サーバの数だけ、全項目を":"で区切って定義を追加する。

5.4.5. 起動方法

5.4.5.1. データベースの起動

(1) MaxDB

dbt 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。

(2) MySQL

MySQL サーバを起動する。最低限必要なオプションは、"--user=mysql"のみである。例は、チューニング後のパラメータである。

```
$ sudo /usr/local/mysql/bin/mysqld_safe ¥
--user=mysql ¥
--max_connections=120 ¥
--innodb_buffer_pool_size=1024M ¥
--read_buffer_size=1M ¥
--query_cache_size=12M &
```

(3) PostgreSQL

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、\$PGDATA/postgresql.conf の関連項目を編集し、PostgreSQL を再起動すること。

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

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、以下のコマンドを実行する。

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

5.4.5.2. ssh-agent 起動

MaxDB,MySQL の場合、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/dbt/.ssh/id_rsa:
Identity added: /home/dbt/.ssh/id_rsa (/home/dbt/.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。
- ※ PostgreSQL(ODBC)では、使用しない。
- ※ 入力するパスワードは、5.3.1.7 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。

5.4.5.3. 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
```

今回の評価では、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。

5.4.5.4. 測定

次のコマンドで DBT-1 を測定する。

MaxDB, MySQL の場合

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

PostgreSQL (ODBC) の場合

```
$ cd ~/dbt1-v2.1/scripts/stats
$ ./run_dbt1.sh /home/pgsql/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/dbt/U200 または/home/pgsql/U200)に BT ファイルが作成され作成されており、最終行の「total errors」が 0 となっていれば、測定の成功となる。

以下に、BT ファイルの例を示す。

Interaction	%%	Avg. Response Time (s)
Admin Confirm	0.08	0.767
Admin Request	0.09	0.702
Best Sellers	4.95	1.928
Buy Confirm	1.21	0.853
Buy Request	2.59	0.861
Customer Registration	2.99	0.000
Home	16.75	0.821
New Products	5.00	1.921
Order Display	0.66	0.774
Order Inquiry	0.74	0.712
Product Detail	16.86	0.730
Search Request	19.79	0.000
Search Results	16.79	1.489
Shopping Cart	11.50	0.841

248.3 bogotransactions per second
49.9 minute duration
total bogotransactions 742911
total errors 0

5.4.6. グラフ化

実行時生成される BT ファイルより、BT/秒及びインタラクション平均応答時間のグラフを作成することができる。オプション詳細は、別冊マニュアル参照の事。

ex)1BT ファイルよりインタラクション平均応答時間グラフをプレビューする

```
# graph.pl -type=l -sub=BT ファイルディレクトリ -v
```

以下にインタラクション平均応答時間グラフのサンプルを示す。

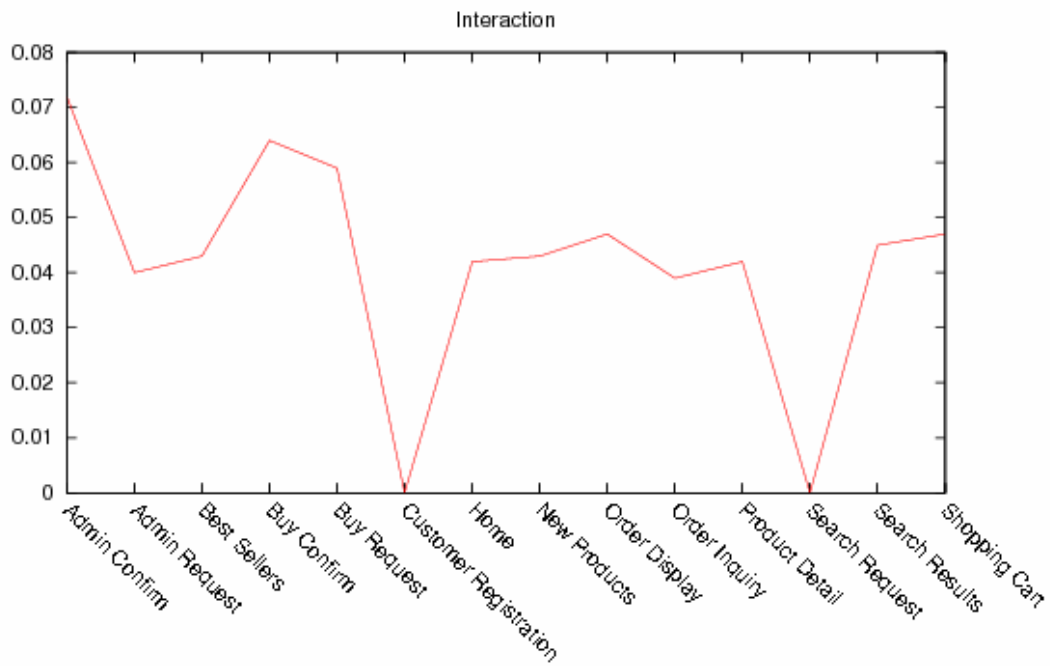


図 5.4-1 グラフツール出力例

5.5. DBT-3 評価手順

5.5.1. 前提条件

5.5.1.1. 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
```

5.5.2. 評価環境

5.5.2.1. 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 5.5-1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 5.5-1 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
<code>lineitem</code> テーブル	<code>/dbt3_1/pgtbls</code>
<code>order</code> テーブル	<code>/dbt3_2/pgtbls</code>
<code>lineitem</code> の <code>l_orderkey</code> インデックス	<code>/dbt3_3/pgtbls</code>
<code>order</code> の <code>o_orderkey</code> プライマリキー	<code>/dbt3_4/pgtbls</code>

(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。

5.5.2.2. 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 5.5-2 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 5.5-2 スケールファクターごとのストリーム数の下限

<code>scale_factor</code>	<code>num_stream</code>
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 `pgsql` 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 `pgsql` 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。

5.5.3. 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。

5.5.3.1. 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。

5.5.3.2. 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。

5.6. 出力情報

5.6.1. MaxDB のログ情報

表 5.6-1 MaxDB ログ情報一覧

情報種別	コマンド	出力ファイル名	備考
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

5.6.2. MySQL のログ情報

表 5.6-2 MySQL(ODBC)ログ情報一覧

情報種別	コマンド	出力ファイル名	備考
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 状態表示	show innodb status	show_innodb_status*.out	*:取得回数
状態確認	show status	show_status*.out	*:取得回数
テーブル状態表示	show table status	show_tabe_status*.out	*:取得回数
ステータス表示	status	status*.out	*:取得回数

5.6.3. PostgreSQL のログ情報

表 5.6-3 PostgreSQL(ODBC)ログ情報一覧

情報種別	コマンド	出力ファイル名	備考
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
接続セッション情報	select * from pg_stat_activity	db_activity*.out	*:取得回数
内部キャッシュ・ファイルシステム I/O 統計	select * from pg_stat_database where datname='DBT1'	db_load*.out	*:取得回数
ユーザインデックスに関する内部キャッシュ・ファイルシステム I/O 統計	select relid, indexrelid, relname, indexrelname, idx_blks_read, idx_blks_hit from pg_statio_user_indexes	index_info*.out	*:取得回数
ユーザインデックス問い合わせ数の統計	select * from pg_stat_user_indexes	indexes_scan*.out	*:取得回数
ロックに関する情報	select relname,pid, mode, granted from pg_locks, pg_class where relfilenode = relation	lockstats*.out	*:取得回数
ユーザテーブルに関する内部キャッシュ・ファイルシステム I/O 統計	select relid, relname, heap_blks_read,heap_blks_hit, idx_blks_read, idx_blks_hit from pg_statio_user_tables	table_info*.out	*:取得回数
ユーザテーブルの問い合わせ方法の統計	select * from pg_stat_user_tables	table_scan*.out	*:取得回数
ロックに関する情報	select * from pg_locks where transaction is not NULL	tran_lock*.out	*:取得回数

5.6.4. DBT-1 のログ情報

表 5.6-4 DBT-1 実行結果を記録するファイル

ファイル名	説明
BT	インタラクション別平均応答時間、BT/秒等 OSDL DBT-1 測定結果が格納される。
mix.log	dbdriver から appServer へのリクエストインタラクション、応答

	時間(msec)が記録される。BT ファイルを計算するための、元情報。
--	-------------------------------------

5.6.5. DBT-3 のログ情報

表 5.6-5 DBT-3 実行結果を記録するファイル

ファイル名	説明
report.html	OSDL DBT-3 実行結果レポートファイル

6. OSDL DBT-1 MySQL(ODBC)版の評価

6.1. 概要

第3章にてポータリングを行った、OSDL DBT-1 MySQL(ODBC)版(以下、ツールを指す場合はDBT-1と記す。特にMySQLを指して言及する場合は、DBT-1 MySQL(ODBC)版と記す)を利用してMySQLの性能測定を行う。

6.1.1. 環境定義書

DBT-1の評価構成を図6.1-1に示す。今回の測定では、全てのプロセスを一台のサーバ上で稼働させる。

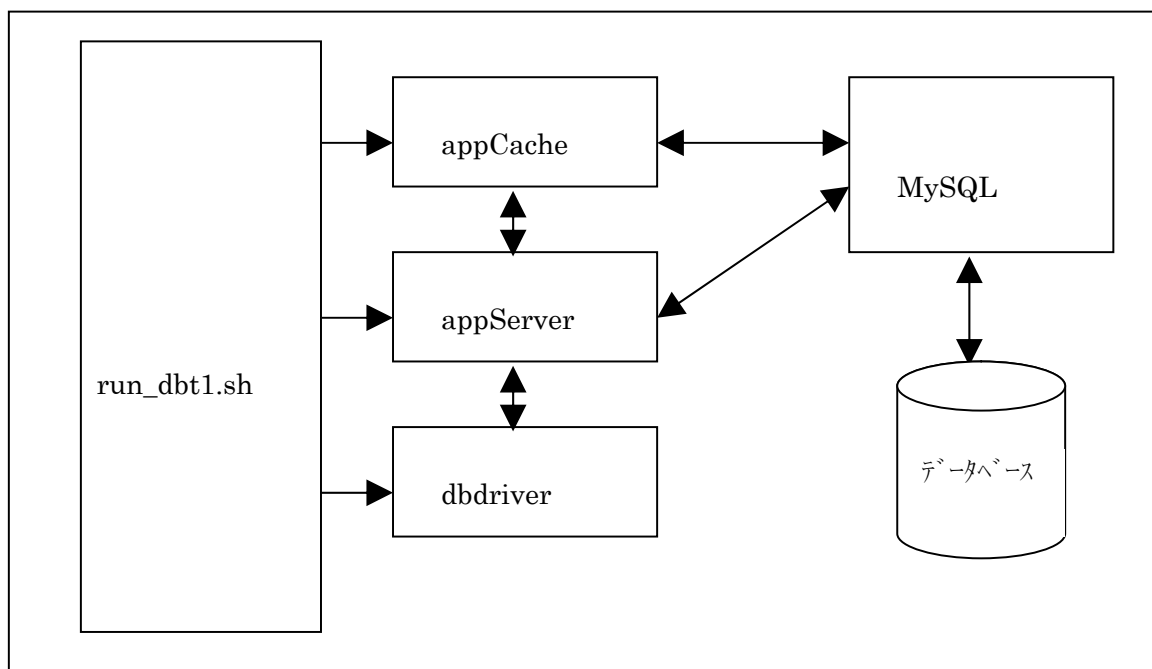


図 6.1-1 システム構成図

6.1.2. システム構成

6.1.2.1. ハードウェア

DELL PowerEdge 1850

Xeon 3.6GHz × 2 (ハイパースレッド ON)

MEMORY 4GB

L2 CACHE 1MB

DISK 72GB(15000rpm) × 2

6.1.2.2. DISK 構成

表 6.1-1 DISK 構成

DISK 容量	マウントポイント	備考
72GB	/boot	100MB
	swap	1GB
	/	RAID0
72GB	swap	swap
	/	RAID0

6.1.2.3. ソフトウェア

- 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
- MySQL 5.0.7
- MyODBC 3.51.10
- iODBC 3.52.2
- OSDL DBT-1 MySQL(ODBC)版 (dbt1-v2.1-MySQL-ODBC-1.0.tar.gz)

6.1.3. 環境構築

第 5 章 OSDL DBT-1/3 評価手順書を参照のこと。

6.2. 測定結果と考察

6.2.1. チューニング前測定結果

DBT-1 側にて、パフォーマンスに影響を与えるパラメータとして、DB 接続数がある。今回は、拡張後の値を使用しこの値は 100 に固定した。理由として、デフォルトの 20 では DB に対して十分な負荷を与えられず、RDBMS の性能信頼性評価という観点より、不要なケースと判断したためである。今回の仮想ユーザ数(EmulateUser 以下 EU)は、高速と言われる MySQL を考慮して、2004 年度 DBT-1 MaxDB(ストアド)版にて頭打ちとなった 1200 より測定を開始し、以降 400 刻みで 2800 まで測定した。図 6.2-1~3 は、その際の BT 値・インタラクション平均応答時間・リソース(CPU)消費量それぞれのグラフである。

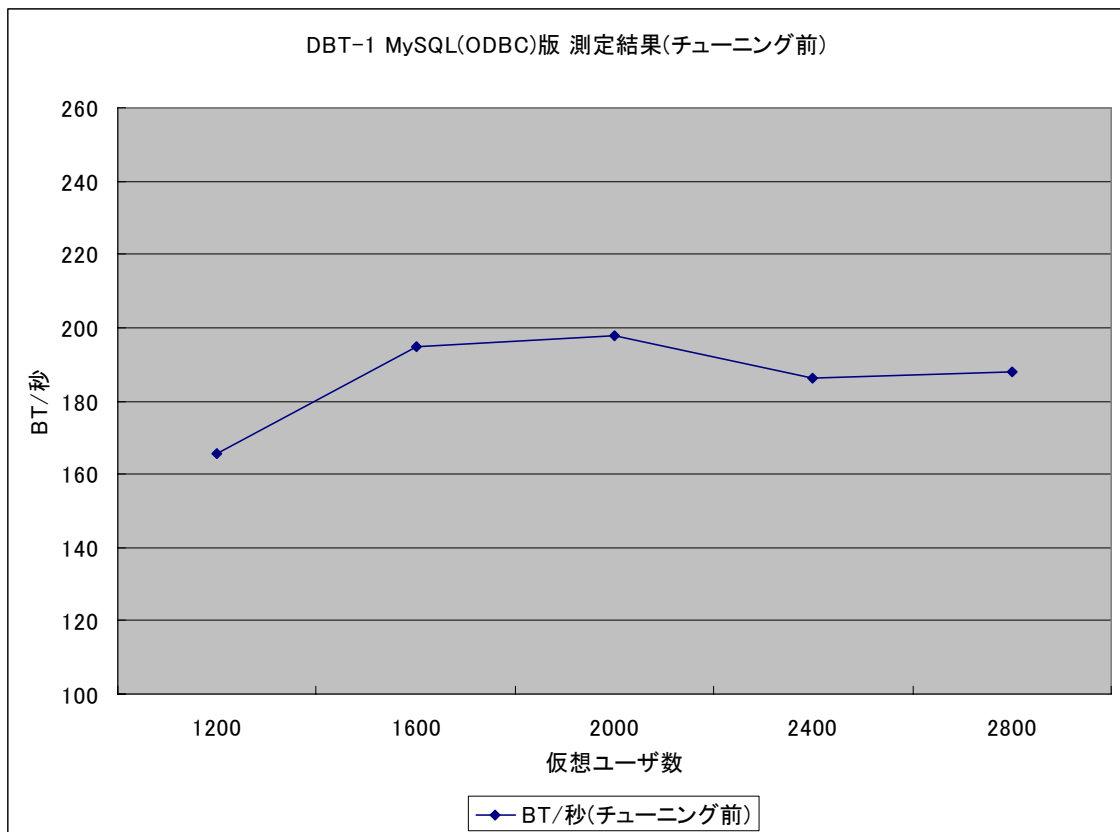


図 6.2-1 DBT-1 MySQL(ODBC)版 BT/秒推移(チューニング前)

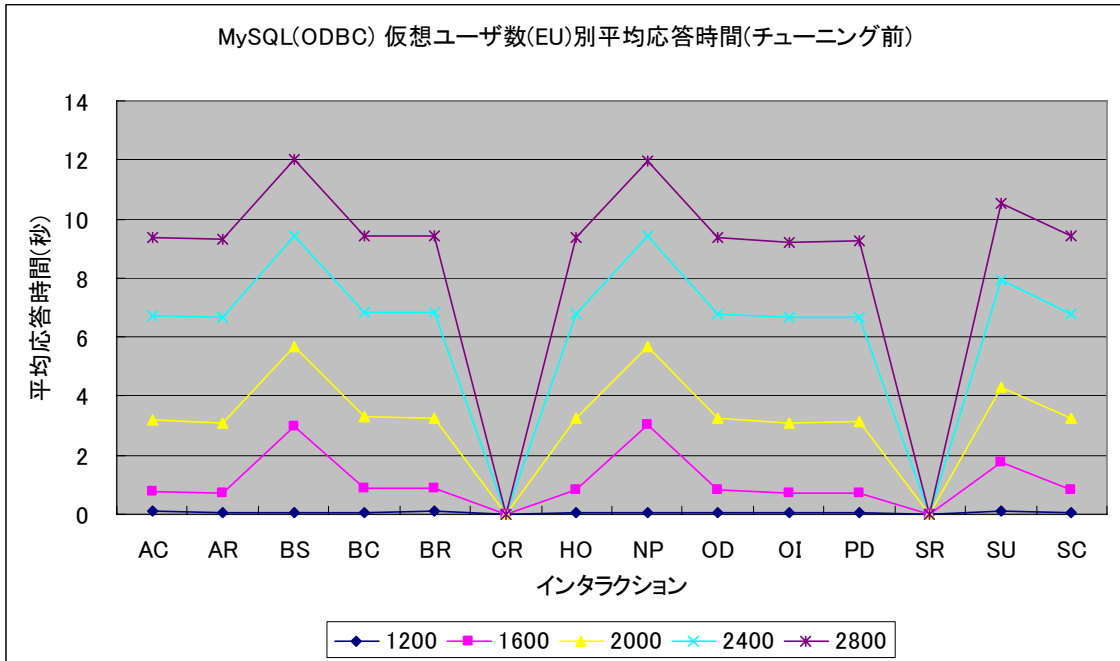


図 6.2-2 DBT-1 MySQL(ODBC)版 平均応答時間推移(チューニング前)

※略号は、第1章 表 1.4-2 インタラクション略号を参照のこと

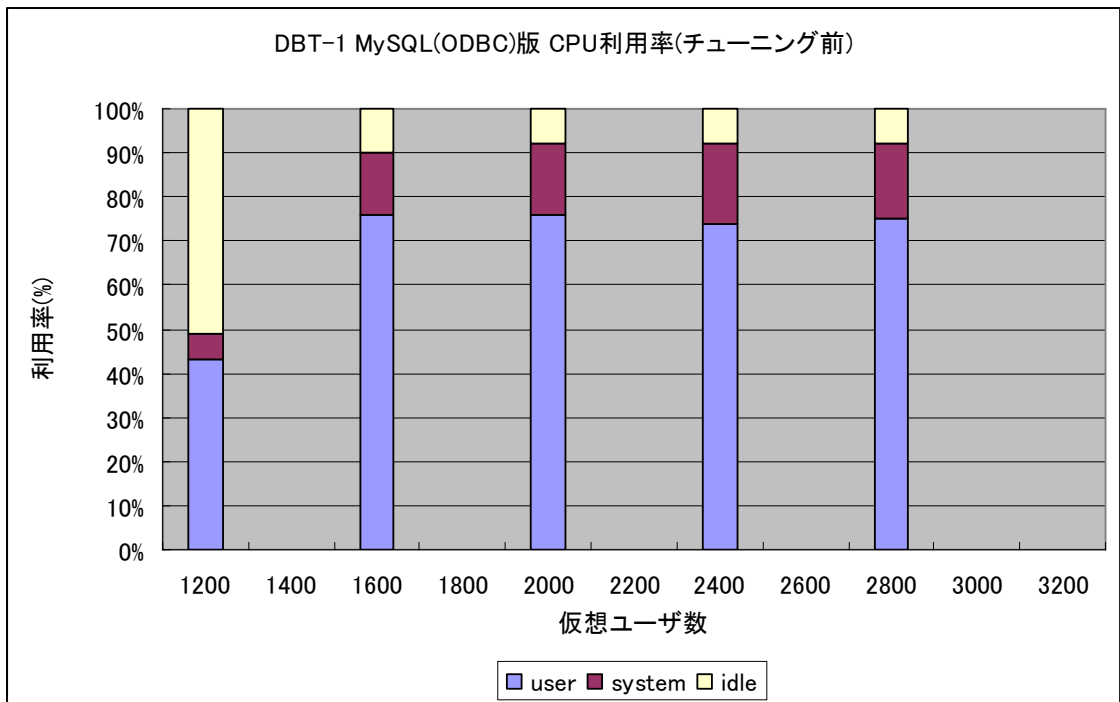


図 6.2-3 DBT-1 MySQL(ODBC)版 CPU 利用率推移(チューニング前)

6.2.2. チューニング前の結果の考察

図 6.2-1DBT-1 MySQL(ODBC)版チューニング前 BT/秒推移より、MySQL のチューニ

ング前動作環境では、より多くの環境で先ず動作するよう、バッファサイズ等パフォーマンスに影響を与えるパラメータの値を小さく設定しているが、EU=1600での理論BT/秒が222.2であることを考えると、チューニング前のBT/秒が195.0というのは、かなり高いパフォーマンスを出しているといえる。但し、EUをこれ以上増やしてもBT/秒が伸びることは無かった。図 6.2-2DBT-1 MySQL(ODBC)版チューニング前平均応答時間推移からは、BS,NP,SUの3インタラクションが他のインタラクションに比べ、時間を要していることが分かる。この傾向は、2004年度で測定したPostgreSQL(ストアド)版に良く似ている。

リソース消費量は、図 6.2-3 DBT-1 MySQL(ODBC)版デフォルトパラメータによるCPU利用率推移より、BT/秒のピークであるEU=1600で急激に増加し、EU=1600以降フラットな利用率となっているが、8%前後のidleがあることから、BT/秒が伸びない原因がCPUパフォーマンス不足に起因するものではない事が伺える。MySQLのチューニング前では、バッファサイズが小さく抑えられていることより、メモリに載らない情報が多く、不足分をDISKアクセスで補っている事によるものと推測し、次節以降でチューニングのポイントを検証する。

6.2.3. チューニングのポイント

6.2.3.1. バッファプールサイズ

チューニング前の結果の考察に記述した通り、まずバッファサイズを確認してみる。今回使用したストレージエンジンInnoDBのデータとインデックスを扱う領域のサイズは、innodb_buffer_pool_sizeパラメータで与える事ができ、使用量を含めた状況は、コマンド”show innodb status;”で確認する事が出来る。出力の該当部分を図 6.2-4 に記す。

```
-----  
BUFFER POOL AND MEMORY  
-----  
Total memory allocated 18885848; in additional pool allocated 1044096  
Buffer pool size 512  
Free buffers 0  
Database pages 498  
Modified db pages 15  
Pending reads 0  
Pending writes: LRU 0, flush list 0, single page 0  
Pages read 508875, created 569, written 55512  
425.46 reads/s, 0.61 creates/s, 49.95 writes/s  
Buffer pool hit rate 1000 / 1000
```

図 6.2-4 バッファサイズ

デフォルトの場合、”Buffer pool size”は8MB(1ブロックは16KB)確保されているが、”Free buffers”の値が0となっている。この状態は余裕が無いことを示しているため、

拡張する事が望ましい。マニュアルでは、データベースサーバ専用機の場合、メモリの 80% まで確保しても良いとある。そのため、4GB のメモリを実装した今回のハードウェアでは、3.2GB まで確保しても良いことになる。今回測定した EU の最大値(3200)に関連するデータに必要な量、及び、アイテム関連の情報量より、時間制限を設けて計測する DBT-1 では、1GB で必要な量は足りると計算し、この値を設定する。

6.2.3.2. クエリキャッシュ

MySQL では、リクエストされたクエリとその結果をメモリ上にキャッシュしておき、同じクエリに対しては、キャッシュよりデータを返すクエリキャッシュ機能がある。デフォルトキャッシュサイズは、0 で機能自体無効としているが、item テーブル等参照専用テーブルがあり、効果が期待できるため、12MB に拡張する。尚、この値の推奨値は定義されていない。表 6.2-1 は、クエリキャッシュの効果を判定するステータスで、"show status;" コマンドで取得可能。

表 6.2-1 クエリキャッシュ関連ステータス

ステータス	説明
Qcache_lowmem_prunes	メモリ不足のためにキャッシュから削除されたクエリ数
Qcache_free_memory	未使用のメモリ領域 (単位: バイト)
Qcache_hits	キャッシュがヒットしたクエリの数(Com_select の値に近いほど良い)
Com_select	SELECT 総数

※今回使用した MySQL のバージョンでは、"Com_select"がカウントアップされなかった。

6.2.3.3. スロークエリ対策

DBT-1 の計測結果から比較的処理に時間を要している 3 つのインタラクション「Best Sellers」、「New Products」、「Search Results」に着目した。それらの SQL 文を EXPLAIN にて解析してみると、何れもクエリプラン上好ましくない filesort が使用されている事が分かる。図 6.2-5 に「Best Sellers」で呼び出す SQL 文の EXPLAIN による解析結果を記す。

```
mysql> explain SELECT i_id, i_title, a_fname, a_lname
          FROM item , author
          WHERE i_subject = 'HOME'  AND i_a_id = a_id
          ORDER BY i_pub_date DESC, i_title ASC¥G
***** 1. row *****
          id: 1
```

```

select_type: SIMPLE
  table: item
  type: ref
possible_keys: i_i_subject,i_i_a_id
  key: i_i_subject
  key_len: 63
  ref: const
  rows: 404
  Extra: Using where; Using filesort
***** 2. row *****
  id: 1
  select_type: SIMPLE
  table: author
  type: eq_ref
possible_keys: PRIMARY
  key: PRIMARY
  key_len: 5
  ref: DBT1.item.i_a_id
  rows: 1
  Extra:
2 rows in set (0.00 sec)

```

図 6.2-5 EXPLAIN による解析結果

このクエリでは、ソートを行う際にインデックスを使用していない。まずは、インデックスを付ける事を検討するのが一般手順ではあるが MySQL では、

- ・ ORDER BY 句の中で異なるキーで昇順と降順を混在させた場合
- ・ 先頭がワイルドカードの LIKE を使用している場合
- ・ 検索時の条件に使用するインデックスとソートに使用するインデックスが異なる場合

等の場合にインデックスを使用しない仕様であることから、パフォーマンスを向上させるためには、別なアプローチが必要となる。今回は、filesort を行う際使用されるバッファのサイズを大きくして計測する。この値には具体的な指針が無いため、数パターンの結果を比較し最適な値を求める。

6.2.4. チューニング後の測定結果

チューニングのポイントによって決定した MySQL パラメータを使用し、測定した結果をチューニング前の値と併せてグラフ化したものを図 6.2-6 に示す。

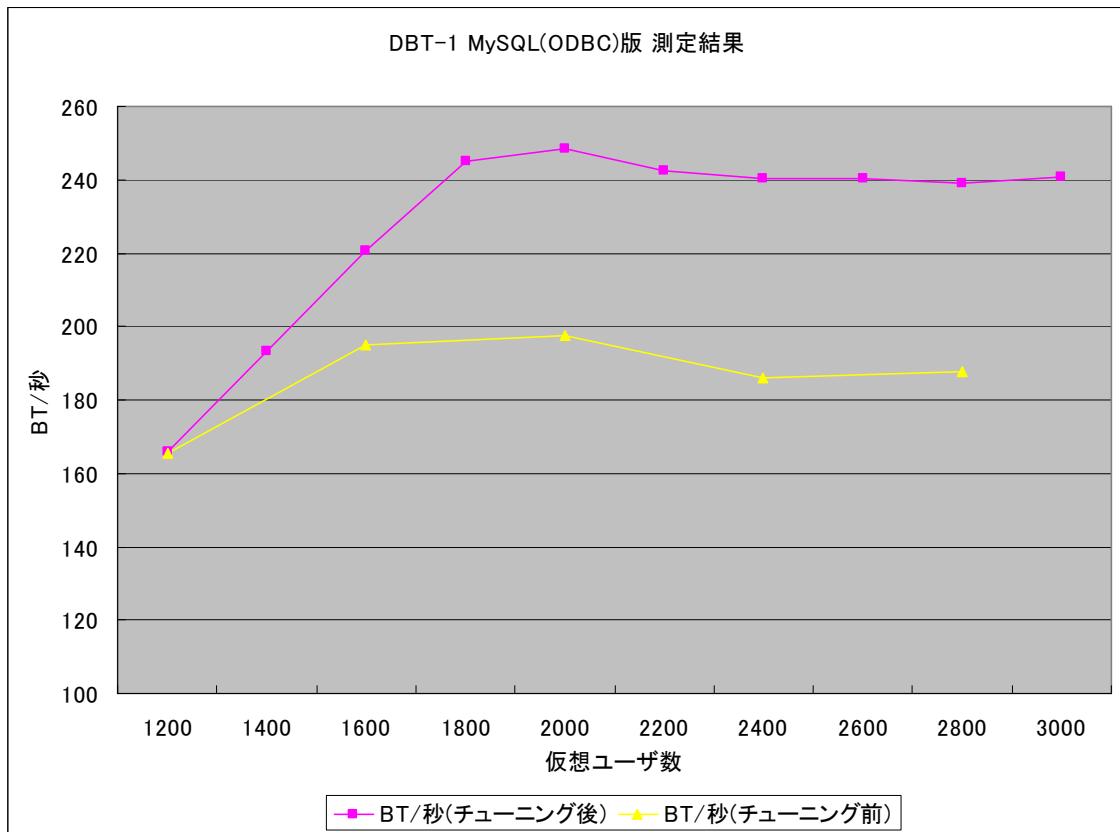


図 6.2-6 DBT-1 MySQL(ODBC)版 BT/秒推移

図 6.2-7 は、チューニング後における CPU リソース利用率をグラフ化した物である。

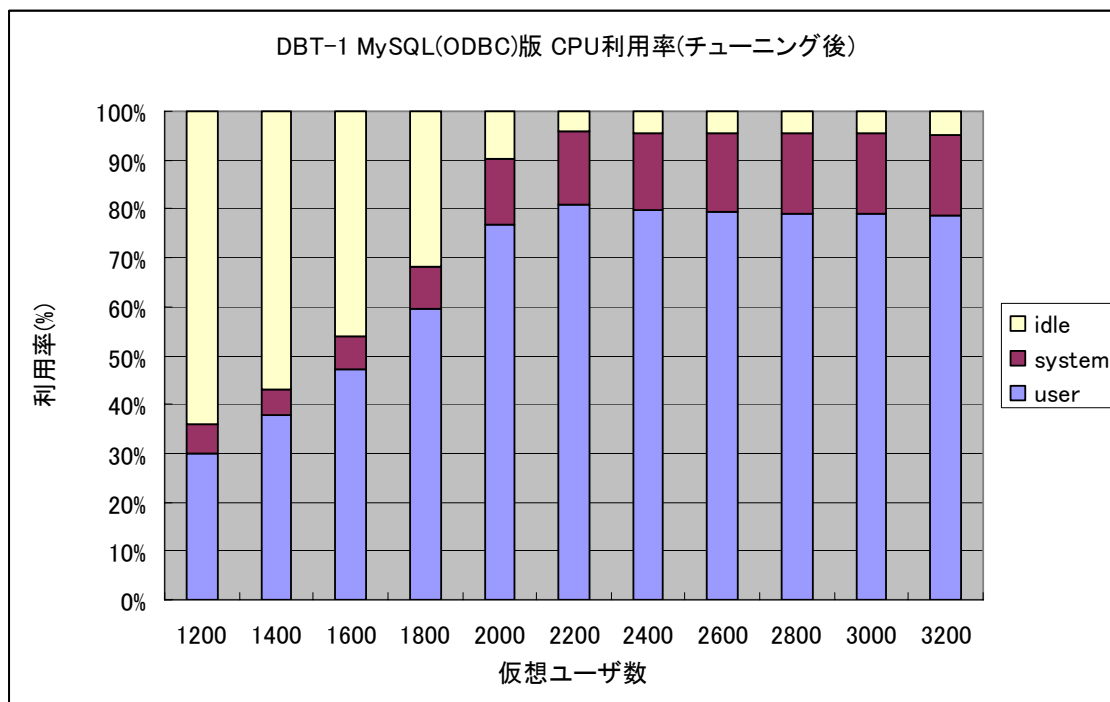


図 6.2-7 DBT-1 MySQL(ODBC)版 CPU 利用率推移

図 6.2-8/9 は、インタラクション毎の平均応答時間を、チューニング前後での傾向の変化を読み取れるよう、Y 軸のスケールを同じにし、EU も合わせたグラフである。

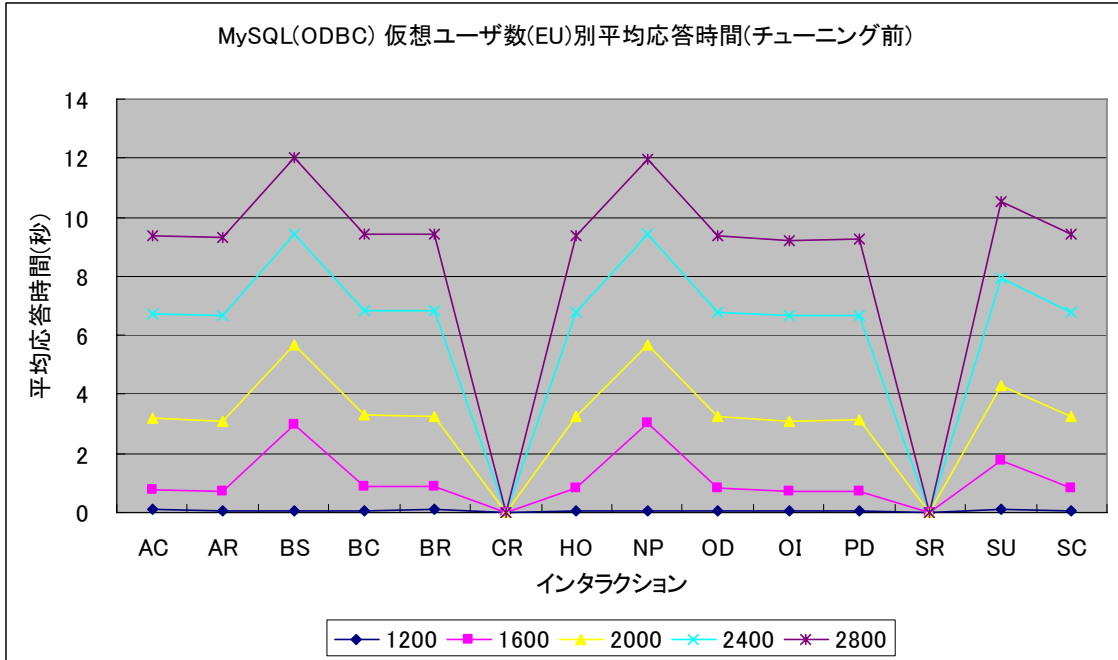


図 6.2-8 チューニング前 平均応答時間

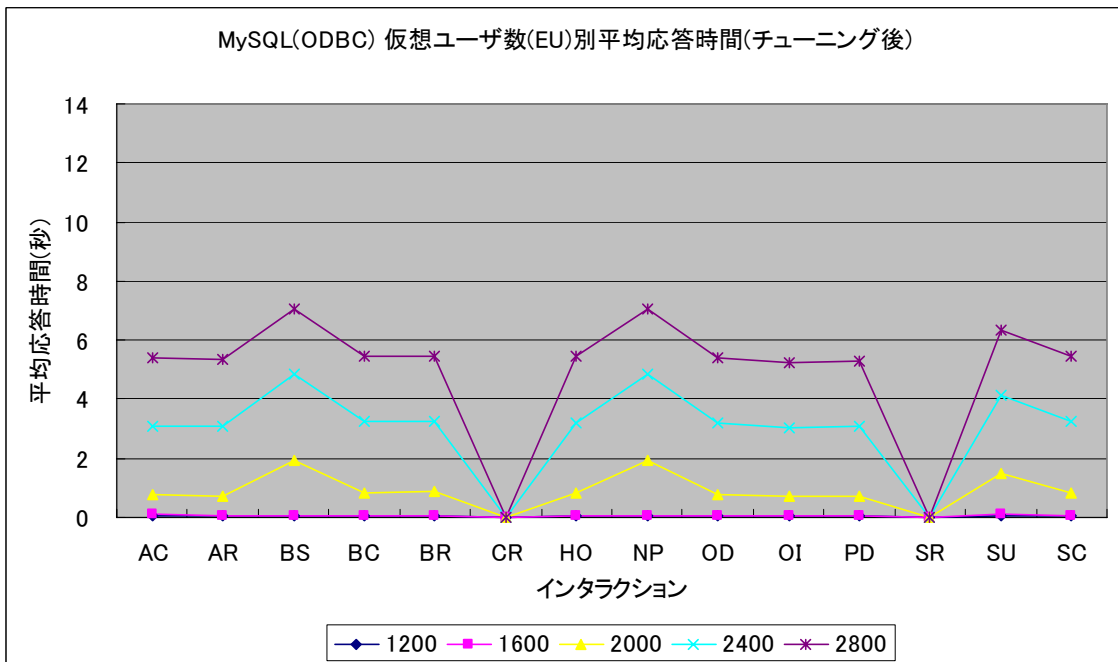


図 6.2-9 チューニング後 平均応答時間

※略号は、第 1 章 表 1.4-2 インタラクション略号を参照のこと

図 6.2-10/11 は、測定開始から 1000 秒間に DISK オペレーションが実施された回数である。この値は、MySQL のログより取得した。

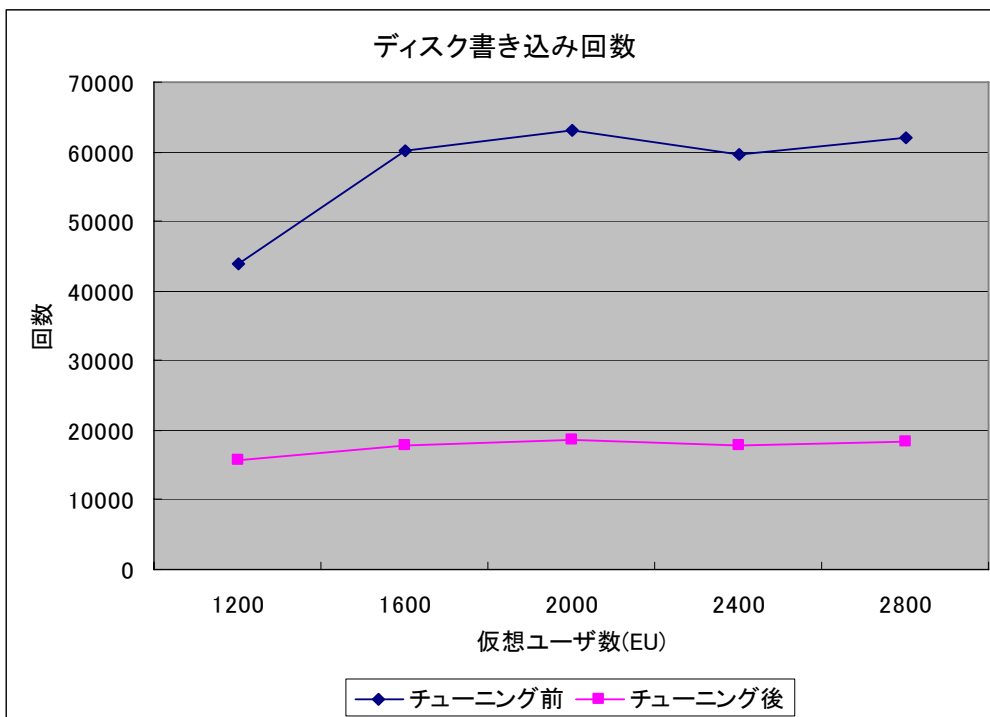


図 6.2-10 DISK 書き込み回数

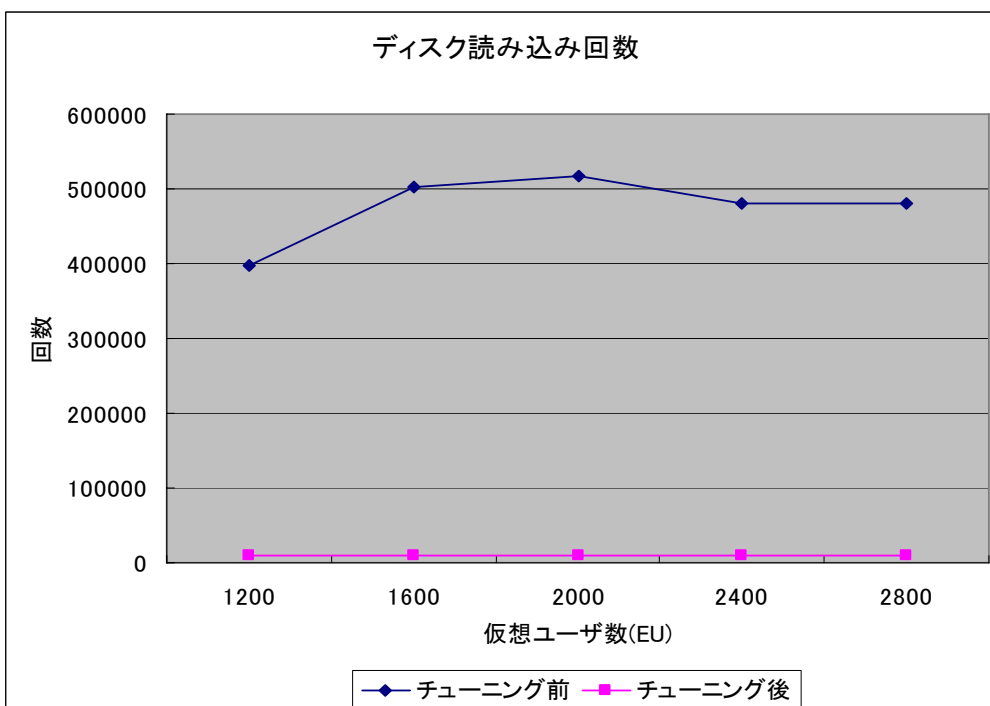


図 6.2-11 DISK 読み込み回数

6.2.5. チューニング後の結果の考察

BT/秒では、EU=1200 までは、理論値*1通りの結果が得られたが、チューニング後では、EU=1800 まで理論値を維持し続けた(図 6.2-6 参照)。平均応答時間のチューニング前後を比較すると、図 6.2-8/9 に見られるとおり、半分ほどの時間に短縮された。また、バッファ拡張の効果として、図 6.2-10/11 にDISKオペレーションの回数をグラフ化した。書き込みは一度バッファに格納され、一定のタイミングでDISKへと書き出されるが、バッファ領域があふれた時点で書き出されるため、小さい領域に置いては、書き出し処理が頻発する。また、読み込みに関しては殆どがバッファに格納されるためか、チューニング後の読み込み回数は非常に小さい値で推移することとなった。

*1 : DBT-1 における BT/秒の理論値は、 $EU \div \text{ユーザオペレーション間隔(ThinkTime)}$ で算出される

6.2.3 チューニングのポイントに挙げたパラメータの利用状況を、表 6.2-2~4 に記す。

表 6.2-2 innodb_buffer_pool の利用状況の変化

	設定値	使用領域	空領域
チューニング前	8M	8M	0
チューニング後	1024M	555.3M	468.4M

innodb_buffer_pool_size は、DBT-1 においては表 6.2-2 にある通り、555.3MB の利用に留まった。これは、DBT-1 では、ある程度アクセスが固定される事に起因するもので、連続稼動且つ不特定多数のアクセスが発生する実システムにおいては、可能な限り大きなサイズに設定して運用し、実際の資料領域のサイズを見て、調整を行うべきであろう。

表 6.2-3 クエリキャッシュの利用状況の変化

	設定値	使用領域	空領域
チューニング前	0	0	0
チューニング後	12M	4.5M	7.5M

表 6.2-4 ソート関連のパラメータ

パラメータ	設定内容	デフォルト値	チューニング後
read_buffer_size	テーブルを順次スキャンするための領域(1 接続毎)	128K	1M

MySQL のチューニングにおいて、一般的とされるパラメータのうち幾つかは、DBT-1 での有用性を実証できなかった。表 6.2-5 はその一覧である。

表 6.2-5 チューニング不要パラメータ一覧

パラメータ	設定内容
innodb_log_file_size	ログのファイルサイズ
innodb_log_buffer_size	ログのメモリ領域
innodb_thread_concurrency	InnoDB 用の R/W 用のスレッド数を指定する。
sort_buffer_size	ソート用の領域 (1 接続ごと)
read_rnd_buffer_size	order by によって 2 度ソートを行う時に使用する領域(1 接続ごと)

innodb_log_file_size / innodb_log_buffer_size

これらの数値を増加させることにより、MySQL が Commit 前のトランザクションのデータをより多く扱うことができるようになるのだが、DBT-1 の場合では変更しても、パフォーマンスの変化は見られなかった。これは、DBT-1 が TPC-W 準拠の Web ベースのトランザクションであり、更新系トランザクションの絶対数が少ない(14 インタラクション中 4 インタラクションが更新系。更新処理が占める割合は全処理の 15%)ことと、トランザクションが開始されてからコミットされるまでの間であまり多くのデータを扱わないため、ログ領域を多くとっても効果がないと考えられる。

innodb_thread_concurrency

このパラメータを増やすことでパフォーマンスが低下した。一度に操作できるデータ量を増やすことで CPU 利用率は増えたのだが、そのかわりに、MySQL でロックが頻繁に発生する結果になり、逆にスループット自体は低下してしまった。

sort_buffer_size

パフォーマンスが低下したわけではないが、sort_merge_passes の値が変化しないことや、初めから十分大きな値が割り当てられていることから、あえて設定する必要が無いことが判明した。

read_rnd_buffer_size

ソートを行う際に、ソートキーを 2 つ以上使用してソートをする場合は、このパラメータを大きくすることを推奨しているが、DBT-1 の場合は、取り出す際のデータ量が少なかったためデフォルトの値で十分であった。

6.2.6. 計測データ

今回の計測で得られたデータを、表 6.2-6 に記す。

表 6.2-6 DBT-1 MySQL(ODBC)版計測データ (参考)

	略号	1200	1400	1600	1800	2000	2200	2400	2600	2800	3000	3200
	チューニング前	AC	0.084	—	0.757	—	3.191	—	6.725	—	9.382	—
AR		0.042	—	0.713	—	3.094	—	6.648	—	9.312	—	—
BS		0.061	—	2.995	—	5.669	—	9.411	—	12.023	—	—
BC		0.077	—	0.896	—	3.298	—	6.846	—	9.436	—	—
BR		0.090	—	0.879	—	3.272	—	6.825	—	9.434	—	—
CR		0.000	—	0.000	—	0.000	—	0.000	—	0.000	—	—
HO		0.039	—	0.825	—	3.230	—	6.785	—	9.384	—	—
NP		0.061	—	3.005	—	5.666	—	9.403	—	11.982	—	—
OD		0.066	—	0.830	—	3.231	—	6.767	—	9.392	—	—
OI		0.043	—	0.722	—	3.101	—	6.645	—	9.229	—	—
PD		0.031	—	0.730	—	3.127	—	6.680	—	9.284	—	—
SR		0.000	—	0.000	—	0.000	—	0.000	—	0.000	—	—
SU		0.100	—	1.785	—	4.297	—	7.919	—	10.510	—	—
SC		0.056	—	0.842	—	3.239	—	6.797	—	9.400	—	—
BT/秒	165.5	—	195.0	—	197.7	—	186.1	—	187.9	—	—	
チューニング後		1200	1400	1600	1800	2000	2200	2400	2600	2800	3000	3200
	AC	0.080	0.080	0.085	0.127	0.767	1.931	3.111	4.213	5.406	6.353	7.400
	AR	0.042	0.042	0.043	0.070	0.702	1.897	3.063	4.189	5.343	6.223	7.373
	BS	0.045	0.046	0.048	0.259	1.928	3.645	4.846	5.944	7.076	8.003	9.122
	BC	0.067	0.066	0.066	0.120	0.853	2.073	3.252	4.341	5.478	6.448	7.524
	BR	0.075	0.076	0.077	0.129	0.861	2.056	3.250	4.325	5.475	6.432	7.526
	CR	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	HO	0.037	0.037	0.037	0.086	0.821	2.024	3.209	4.284	5.431	6.400	7.490
	NP	0.045	0.046	0.048	0.262	1.921	3.645	4.829	5.959	7.081	8.026	9.144
	OD	0.049	0.050	0.051	0.099	0.774	2.008	3.171	4.239	5.379	6.379	7.454
	OI	0.041	0.041	0.042	0.075	0.712	1.901	3.057	4.128	5.255	6.242	7.307
	PD	0.033	0.033	0.034	0.068	0.730	1.901	3.087	4.167	5.311	6.283	7.367
	SR	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	SU	0.076	0.082	0.094	0.249	1.489	2.925	4.120	5.188	6.347	7.310	8.392
SC	0.051	0.052	0.053	0.104	0.841	2.048	3.233	4.306	5.461	6.420	7.513	
BT/秒	165.8	193.3	220.5	245.2	248.3	242.6	240.4	240.3	239.2	240.8	240.4	

※略号は、第 1 章 表 1.4-2 インタラクション略号を参照のこと

6.3. まとめ

3章で改変を実施した DBT-1 MySQL(ODBC)版を使用した評価では、MySQL はデフォルト設定で動作させても、DBT-1 のようなトランザクションパターンであれば、十分なパフォーマンスを発揮することが確認できた。

チューニング後の平均応答時間は、EU=2000 で約 1 秒であった。BT/秒は EU=1800 の時点で頭打ちとなっているが、応答時間は実用的な数値と言える。今回使用したハードウェアと DBT-1 のようなトランザクションパターンでは、ピーク時 250BT/秒、応答時間 1 秒を目安としたシステム設計が可能となる。

このことより、オープンソースの RDBMS でも、用途次第ではチューニングする事で商用 RDBMS に匹敵するパフォーマンスを出せる。

今回の評価は、単体サーバ内で完結させたものであったが、今後は、より実運用を想定したシステム構成、特に高可用性を持たせた環境で、どのような測定結果が出てくるか、検証することが必要となるであろう。

7. PostgreSQL (ODBC) 版 OSDL DBT-1 の評価

7.1. 概要

本章では、ポーティング向上の確認のため改変された DBT-1 PostgreSQL(ODBC)版を使用して、PostgreSQL バージョン 8.0.3 の評価を実施する。ODBC 接続で直接 SQL を発行することによる基本的な測定と DBT-1 PostgreSQL(ストアド)版との比較、簡単なチューニングを実施しての測定効果を検討する。また、本稿執筆時にリリースされた PostgreSQL8.1beta1 を使用して、効率改善機能の効果を検証する。

7.2. 環境定義

7.2.1. システム構成

今回の評価で使用したシステム構成は、以下の表 4.2-1の通りである。

表 7.2-1 システム構成

製品名	Dell PowerEdge 2600
プロセッサ	インテル Xeon プロセッサ 3.06GHz、2CPU HT on
メモリ	6GB
ハードディスク	内蔵ドライブベイに、UltraSCSI320 72GB 2 台

DBT-1 PostgreSQL(ODBC)版としての基本性能を検証するため、今回は、表 7.2-2のような単純な構成をとった。

表 7.2-2 ディスク構成

72GB * 2	/	69GB
	/mydata	72GB
	Swap	3GB

※ /mydata は、後述の DBT-1 データベース用のディレクトリ

7.2.2. ソフトウェア

以下のソフトウェアを使用した。

- MIRACLE LINUX V3.0 – Asianux Inside
Linux 2.4.21-9.30AXsmp #1 SMP
- PostgreSQL 8.0.3/8.1beta1
- unixODBC 2.2.113.51.10
- psqlODBC08.00.0102

- OSDL DBT-1 V2.1 改訂版 (dbt1-v2.1-PostgreSQL-ODBC-1.0.tar.gz)

7.3. 環境構築と測定

第5章「OSDL DBT-1/3 評価手順」を参照のこと。

7.4. 測定と考察

7.4.1. 測定結果の収集

実行結果は 実行スクリプト・ファイル `run_dbt1.sh` の引数に指定した出力用ディレクトリに出力される。主に参照されるべき測定値は、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: ポゴ (擬似) トランザクション/秒 = web 経由のリクエスト数/秒) である。

7.4.2. 測定の観点

改変した DBT-1 による測定は、以下のような観点で測定・実行を行った。

- ODBC 接続で直接 SQL を実行することによる基本的な性能の変化
仮想ユーザ数 (システムに同時に接続する Web エンドユーザ数) を変化させることによるスループットがどのように変化するかを測定する。
- DBT-1 PostgreSQL(ODBC) 版と DBT-1 PostgreSQL(ストアド) 版の比較
今回ポーティング作業を実施した、DBT-1 PostgreSQL(ODBC) 版を DBT-1 PostgreSQL(ストアド) 版と比較し、インターフェースの違いによる性能の差異を検証する。
- 簡単なチューニングによる性能の変化と ODBC 接続の特性の有無
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。前年度の報告では、DBT-1 PostgreSQL(ストアド)版で、“一般的なチューニング”による性能の変化を考察したが、DBT-1 PostgreSQL(ODBC)版でも“一般的なチューニング”があたえる効果を測定する。なお、本章で述べるチューニング項目は、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。
- PostgreSQL のバージョンの違いによる性能の変化
マルチ CPU 環境での共有ファイル・メモリの効率改善機能が含まれた、8.1 ベータと性能を比較する。

7.4.3. ODBC 接続のダイレクト接続による基本的な性能の変化

ODBC 接続で直接 SQL を発行することによる基本的な性能の変化を検証するため、仮想ユーザ数 (システムに同時に接続する Web エンドユーザ数) を変化させ、スループットがどのように変化するかを測定する。初期設定値であるデータベースに対する同時接続数 20 に対して仮想ユーザ数を 400 から 1200 まで 200 ごとに変化させ、スループットを測定する。

7.4.3.1. 測定結果

測定した結果を示す。

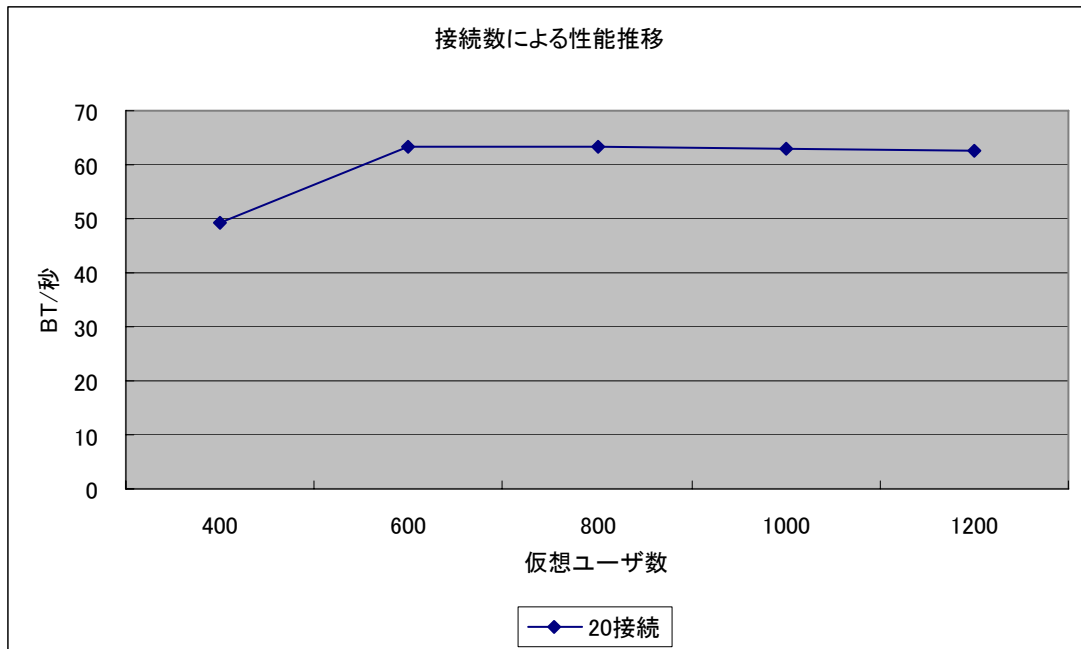


図 7.4-1 接続数による性能推移

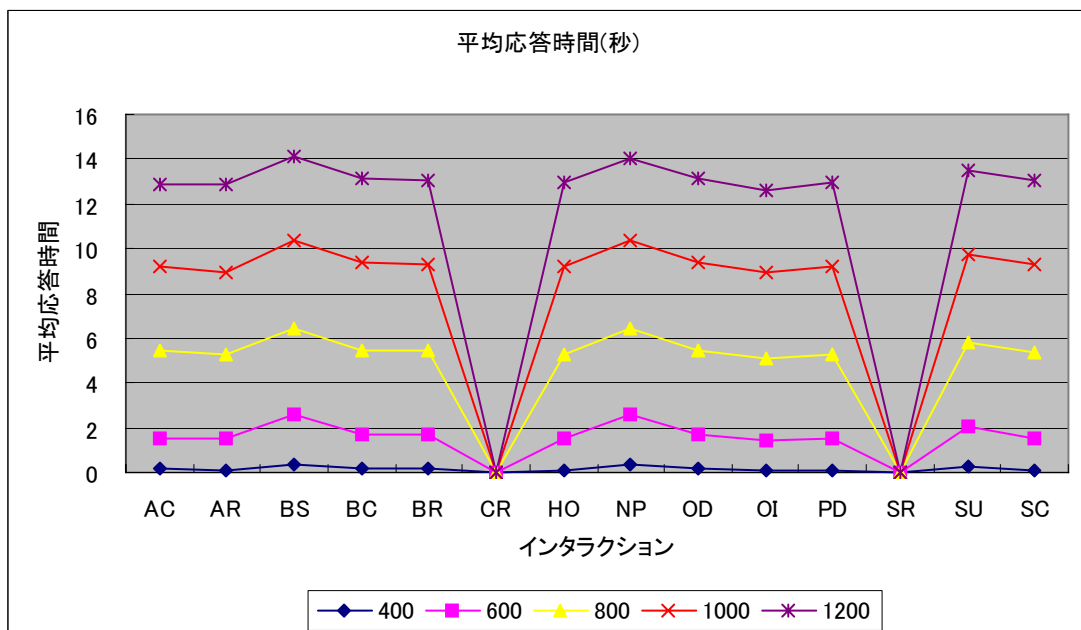


図 7.4-2 平均応答時間推移

表 7.4-1 インタラクシヨン略号

インタラクシヨン	略号	インタラクシヨン	略号
Admin Confirm	AC	New Products	NP
Admin Request	AR	Order Display	OD
Best Sellers	BS	Order Inquiry	OI
Buy Confirm	BC	Product Detail	PD
Buy Request	BR	Search Request	SR
Customer Registration	CR	Search Results	SU
Home	HO	Shopping Cart	SC

スループットの BT/秒は、ユーザ数を増加させても、仮想ユーザ数=600 でスループットが頭打ちとなり、しだいに少しずつ落ちていった。仮想ユーザ数 600 での理論値は、83BT/秒ではあるが、デフォルト値で、63BT/秒を出せたことになる。仮想ユーザ数を増やした場合、インタラクシヨンあたりの平均応答時間が延びてしまうが、処理件数が増えるため、スループットとしては、やや右肩下がりの傾向となった。

7.4.3.2. 考察

仮想ユーザ数 600 で出力されたvmstat等のログをみるとI/Owaitが発生しており、システムにおけるidle状態の割合が多く、CPUのパフォーマンスとしては、余裕がある状態であった。一方、トランザクシヨン同士のロック待ちには問題がなかった。結果的に、サーバのCPU リソースを 100%使用していない状態と言える。(図 4.2-1参照)。すなわち、BT/秒が伸びない原因としては、ディスクアクセスが考えられる。

今回の基本的な測定では、ディスク構成を単純にしたため、I/Owait が発生しているが、各種設定値を変えずチューニングを行わない状態で、システムに対する負荷を強化するためには、データベースをクラスタ化し複数のハードディスクに割り当てるなどディスクアクセスを分散させるなどの考慮が必要である。

続いて、DBT-1 PostgreSQL(ODBC)版での基本性能に ODBC インターフェースに特化した事象があるかどうかを検証するため、次節にて DBT-1 PostgreSQL(ストアド)版での測定結果と比較する。

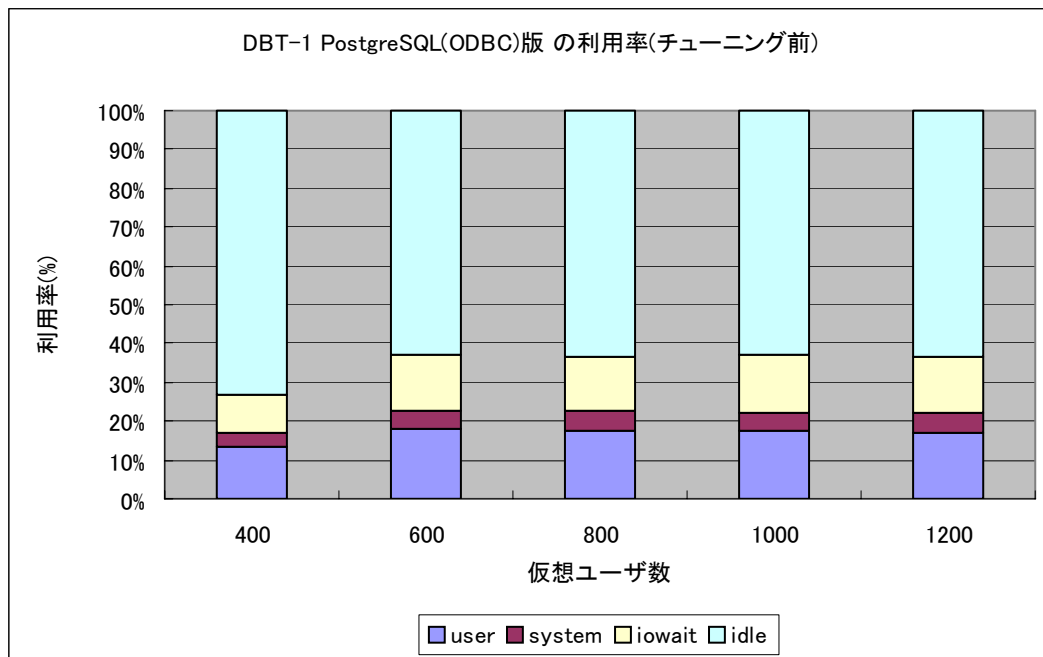


図 7.4-3 DBT-1 PostgreSQL(ODBC)版のリソース状況

7.4.4. DBT-1 PostgreSQL(ODBC)版と DBT-1 PostgreSQL(ストアド)版の性能比較

DBT-1 PostgreSQL(ODBC)版にポータリングしたことにより、PostgreSQL データベースの下での DBT-1 は、ネイティブ関数である libpq インターフェースによるストアドプロシージャ (DBT-1 PostgreSQL(ストアド)版) と ODBC 接続による直接の SQL 実行 (DBT-1 PostgreSQL(ODBC)版) という 2 種類のモデルでの測定が可能になった。それぞれのモデルで DBT-1 を同じ設定値条件で実行し、インターフェースの違いによる性能の差異を検証する。

7.4.4.1. 測定結果

仮想ユーザ数 600 以上でスループットが頭打ちになるが、DBT-1 PostgreSQL(ODBC)版の方が、DBT-1 PostgreSQL(ストアド)版より、若干スループットは良い。傾向としては、いずれも同様の傾向を示している。

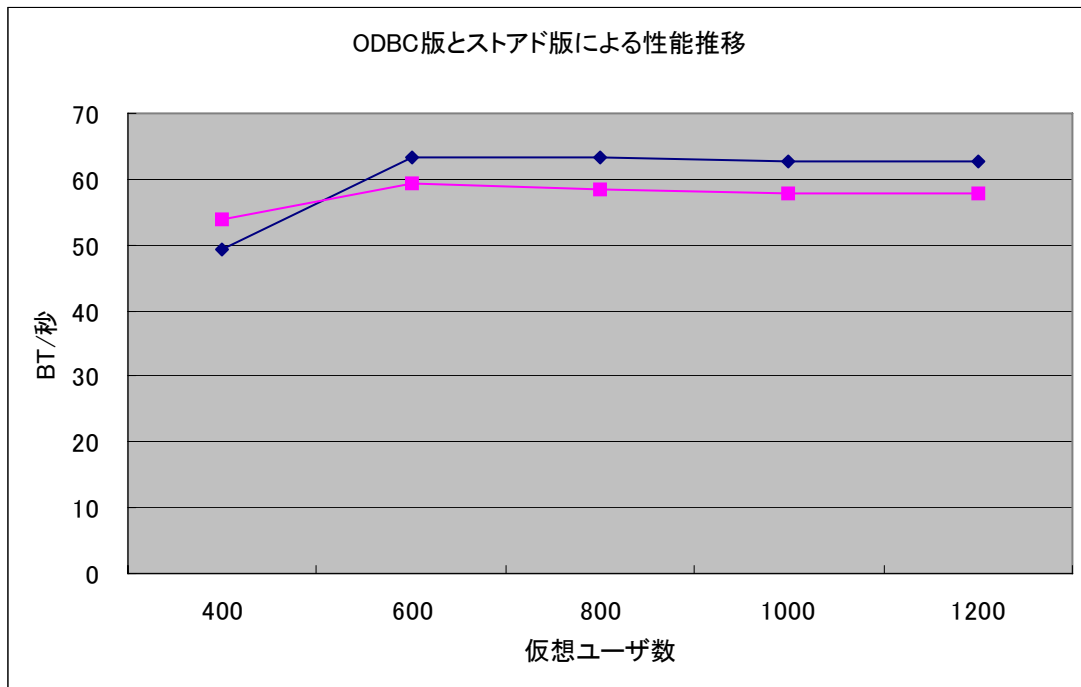


図 7.4-4 DBT-1 モデルの違いと性能推移

7.4.4.2. 考察

ODBC接続による直接のSQL処理のほうが若干スループットが高くなった理由を、測定時のリソースの使用状況 (図 7.4-5) から考察してみる。スループットのピークである仮想ユーザ数=600 で比較すると、CPUの利用率は、DBT-1 PostgreSQL(ストアド)版の方が高い。すなわち、ストアドプロシージャの方が、同じ仕事に対してより多くのCPUを消費すると言える。また、IOWaitも増えていることから、ディスクへの書き込みも多い。システムのリソースを多く消費することがスループットの差となっている。

リソースを多く消費する原因を PostgreSQL 内で実行される SQL のログから調べてみた結果、次のようなことがわかった。

DBT-1 PostgreSQL(ストアド)版では、一部のストアドプロシージャ内で複数回 SELECT 指令を実行し、リザルトセットをレコード型に横に展開した形式で、データを得るような定義となっているインタラクションがあった。一方、同じインタラクションが、DBT-1 PostgreSQL(ODBC)版では、検索した結果をすべて取得するためには、複数回実行している SELECT 指令を副問合わせで定義することができる。そして、そのカーソルに対して、単純に ODBC の SQLFetch 関数で必要なデータ件数だけループして、C 言語のプログラム側で逐次データを処理するようにプログラミングすることができる。このある特定のインタラクションが実行された場合、実行される SQL の指令数が、DBT-1 PostgreSQL(ストアド)版の方が多くなる場合があるため、結果としてシステムのリソースをより多く消費する結果となっている。

PostgreSQL バージョン 8.0.3 は、ストアドプロシージャの結果として、リザルトセット (SETOF 型) を返すことが可能であるが、DBT-1 では、リザルトセットを返せない

RDBMS(MaxDB)も考慮した結果、そのようなストアドプロシージャの定義を行なったと考えられる。

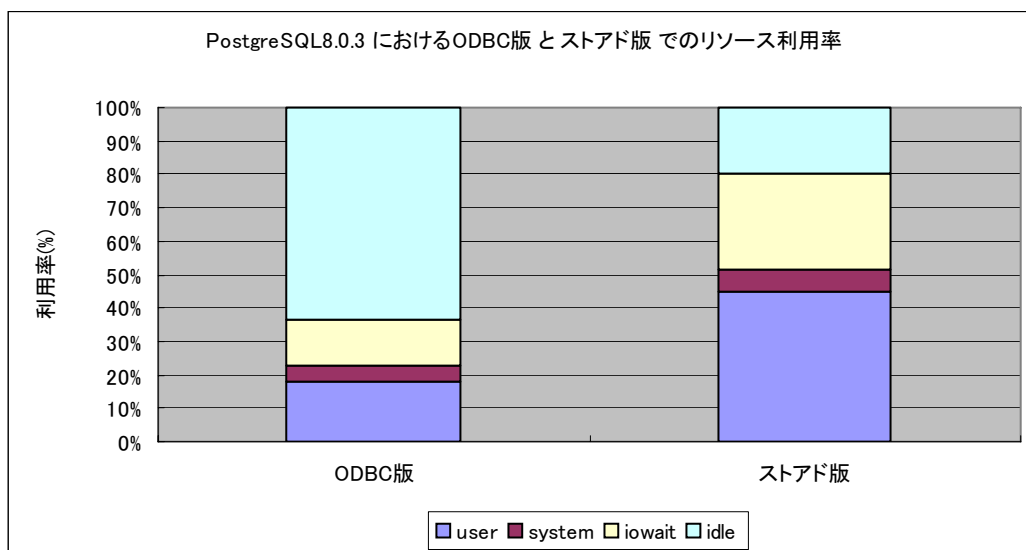


図 7.4-5 リソースの使用状況比較 (仮想ユーザ数 600)

ストアドプロシージャが使用される目的の一つに、本来、クライアントとサーバの通信コストを削減するために定義されることがある。今回のワークロードのテスト環境は、DBT-1 のクライアントプログラムとデータベース・サーバが同一マシンにあるため、ネットワーク負荷を軽減することのできるストアドプロシージャの優位性がいかされていない。また、今回使用した設定値では、システムのリソースに余裕を持たせていることもあり、スループットの BT/秒としてはあまり変わらない性能差となった。

クライアントとサーバを別マシンにしてネットワークで結んだ環境での測定であれば、ネットワークのオーバーヘッドの有無がスループットの差として現れる結果となる可能性がある。

また Linux の下では、現実には ODBC 接続より、ネイティブな関数 libpq ライブラリや JDBC 接続を使用してプログラムを作成するケースが一般的と考えられる。より現実的なワークロードにするためには、DBT-1 PostgreSQL(ODBC)版をネイティブな関数 libpq インターフェースまたは JDBC でさらに改変し、計測することも今後の課題と言える。

7.4.5. 簡単なチューニングによる性能の変化

7.4.5.1. 目的

DBT-1 PostgreSQL(ODBC)版に対して、簡単なチューニングを施し、性能特性の変化を調べる。一般的なチューニングの手法が、DBT-1 PostgreSQL(ODBC)版にも有効かどうかを検証する。今回は 7.4.2.3 の考察より、特にディスクの IO に着目し、PostgreSQL の設定パラメタとデータベース接続数をチューニングし、その効果を検証する。

7.4.5.2. チューニングのポイント

前年度の DBT-1 PostgreSQL(ストアド)版で実施した PostgreSQL の設定パラメタに対する一般的なチューニングと同様なチューニングに加え、さらにテスト環境のディスク構成を意識したチューニングを実施し、その効果を測定する。ただし、チューニングは、ハードウェアリソース、ソフトウェアリソース、コネクション数、ユーザ数などの違いなどにより、その効果は異なるため、今回の設定値が最適値とは言えないことをお断りしておく。

仮想ユーザ数に対してデータベース側の処理能力を向上させるため、同時接続数を 100 に増やし、同じ仮想ユーザ数仮想ユーザ数 400 からの変化を測定する。ただし、今回使用した、psqlODBC ドライバは、最大接続数が内部的に 128 接続で設定されているので、それ以上の同時接続数を増やすことはできない。

DBT-1 PostgreSQL(ODBC)版で、チューニングの効果がどのように想定結果に変化をもたらすかを計測する。同時接続数 20 で仮想ユーザ数 400 から 1200 まで変化させたときの性能の変化を測定し、7.4.3項での結果と比較する。表 7.4-2 にチューニングの前と後の値を以下に示す。チューニングは、バッファークャッシュファイルとワークメモリを増加（それに対応するカーネル共有メモリの変更）させ、ログの書き込み頻度を減らし、トランザクションが持つ最大ロック数も増やした。

表 7.4-2 チューニングのパラメータ

postgresql.conf のパラメタ名	チューニング前	チューニング後
max_connections	100	200
shared_buffers(8KB each)	1000	131072(注 1)
work_mem (KB)	1024	393216
maintenance_work_mem(注 2)	16384	39216
bgwriter_delay	200	10000
bgwriter_percent	1	3
checkpoint_segments	3	30
effective_cache_size	1000	102400
max_locks_per_transaction	64	124

注 1) shared_buffers を増やすと同時にカーネルの共有メモリ
/proc/sys/kernel/shmmax を “3134728” に変更した。

注 2) DBT-1 の測定時にこのパラメタが直接影響する SQL 指令は、実行されない。

7.4.5.3. チューニング後の測定結果

チューニング後の測定結果を図 7.4-6 と図 7.4-7 に示す。チューニングしたことにより、仮想ユーザ数の増加とともにスループットが増加し、PostgreSQL のリソース管理が良くなっていることを示している。トランザクションの処理負荷が減少したことで、ユーザ数の増加に対応し、スループットの最高値も仮想ユーザ数=600 から 1200 へと改善されている。

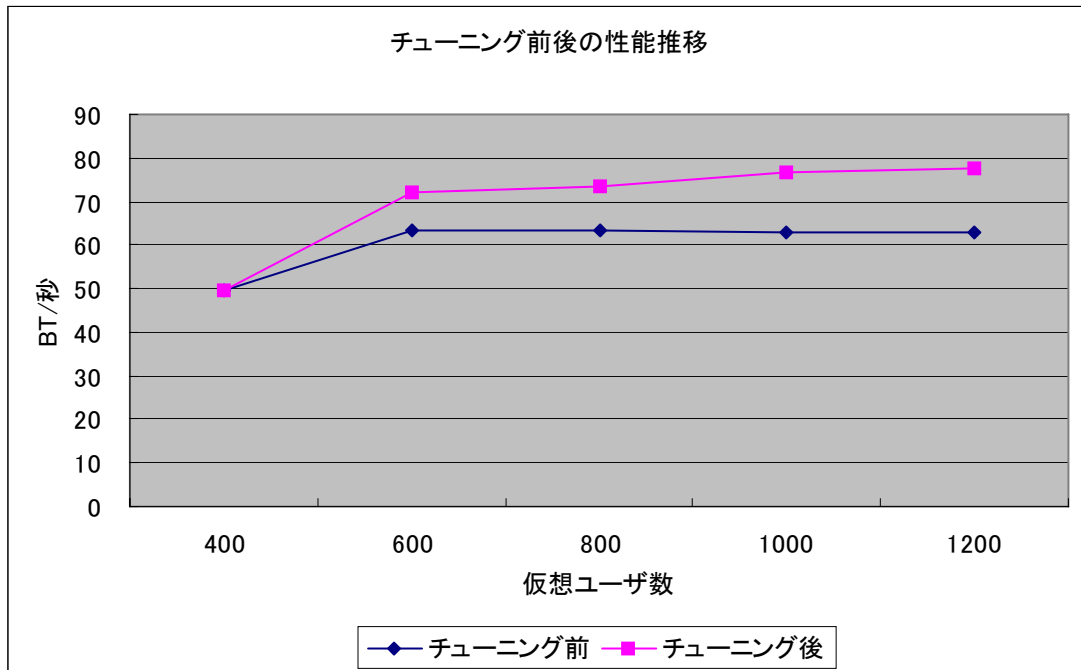


図 7.4-6 チューニング前後の性能推移

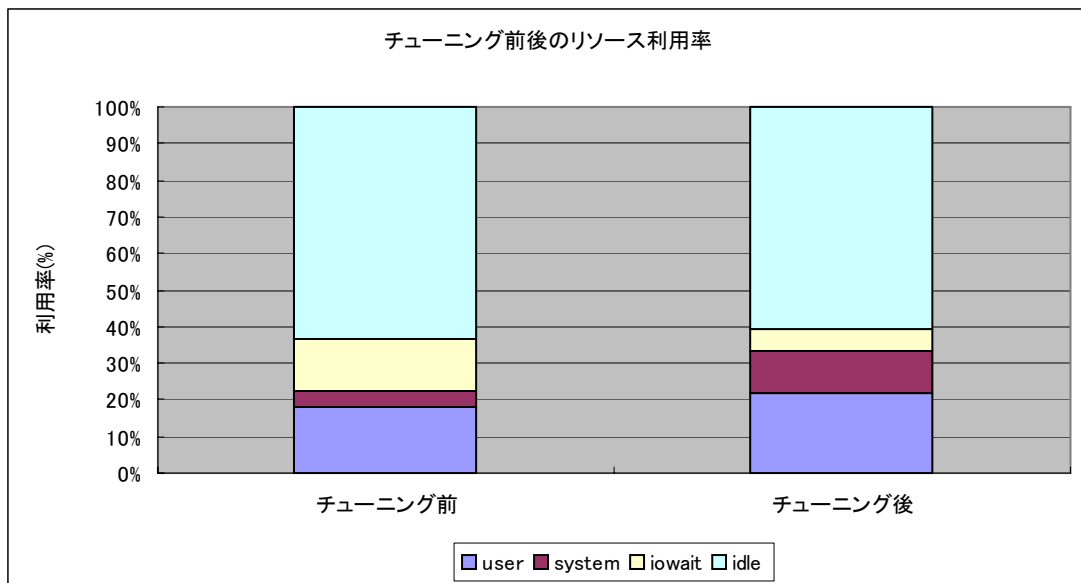


図 7.4-7 チューニング前後のリソース利用率比較 (仮想ユーザ数 600)

7.4.5.4. チューニング後の考察

チューニングを施した結果、今回の DBT-1 PostgreSQL(ODBC)版も、前年度の DBT-1 PostgreSQL(ストアド)版でおこなったチューニングの効果と同様に仮想ユーザ数の変化に対して効率向上が見られた。チューニングしたパラメタを使用することでメモリ使用の最

適化とディスクアクセスの改善がなされたためと思われる。仮想ユーザ数 600 において、63BT/秒から 72BT/秒と約 15%のスループット向上が見られた。

SQL を直接発行する DBT-1 PostgreSQL(ODBC)版に対しても一般的なチューニング対策項目は、有効であるといえる。

7.4.6. DBT-1 PostgreSQL(ODBC)版での PostgreSQL バージョン間の性能比較

PostgreSQL バージョン 8.1beta1 では、マルチ CPU のハードウェアに対する効率改善がなされているとの記述がある。今回のシステム環境は、マルチ CPU (2CPU HT on) であったため、PostgreSQL8.1beta1 で測定を実施し、PostgreSQL バージョン 8.0.3 とバージョン 8.1beta1 での測定結果を比較する。データベース接続数 (dbconnections=100) で仮想ユーザ数を変化させる。postgresql.conf の設定パラメタに対するチューニングは実施せず、デフォルト値を使用した。

7.4.6.1. 測定結果

図 7.4-8で示す結果によるとスループットが約 1.5 倍に向上しかつ仮想ユーザ数の頭打ちが 800 まで増えている。明らかにバージョン 8.1beta1 では、効率が向上しているといえる。併せて図 7.4-9に利用ユーザ数 600 でのCPU利用率を示す。

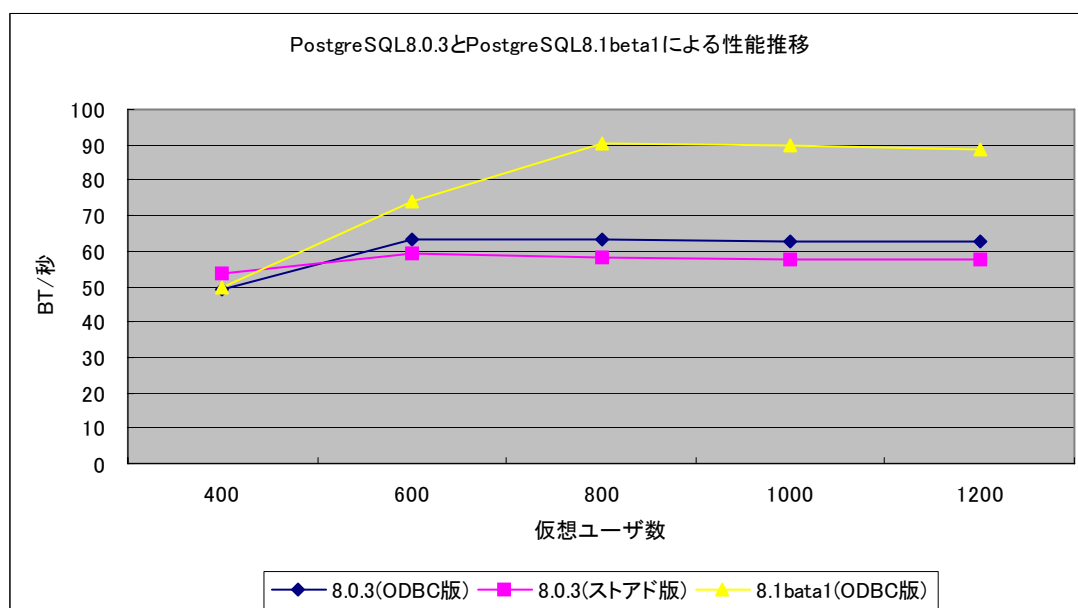


図 7.4-8 PostgreSQL のバージョンの違いと性能推移

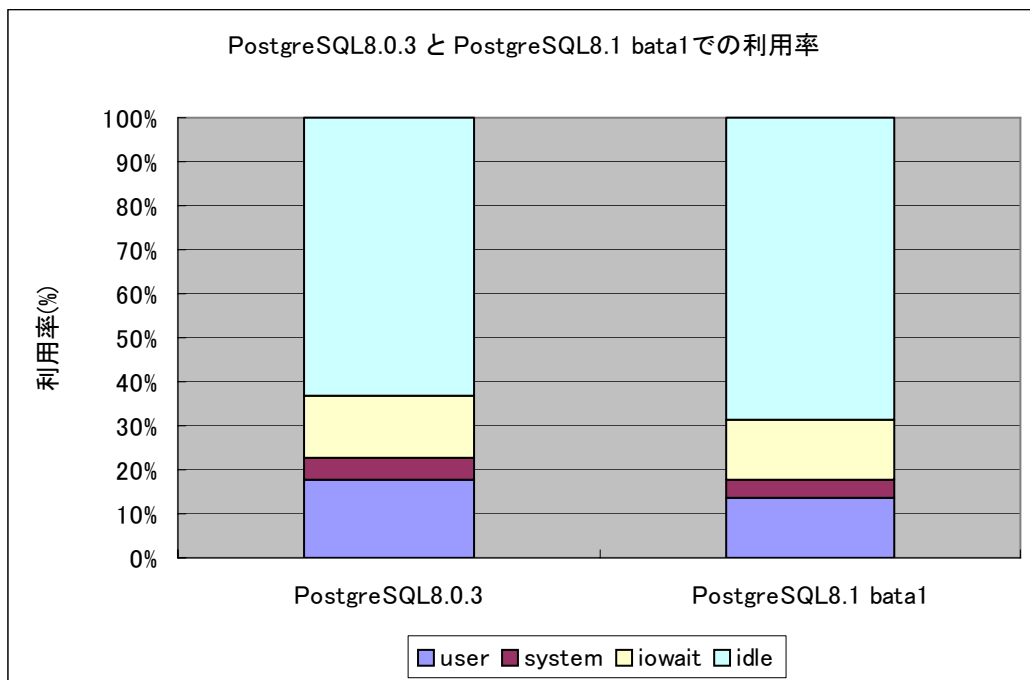


図 7.4-9 バージョン間の利用率の比較 (仮想ユーザ 600)

7.4.6.2. 考察

PostgreSQLバージョン 8.1beta1 は、仮想ユーザ数 600 での理論値 83BT/秒に対して、73BT/秒を記録している。構成が単純なことや設定値がデフォルトであることを考えると PostgreSQLバージョン 8.1beta1 の基本性能が向上していることが伺える。図 7.4-10のように、1 トランザクションあたりの平均応答時間を比較をしてみても、PostgreSQLバージョン 8.1beta1 の方が明らかに短くなっている。スループットの向上は、この平均応答時間が短縮されたためと言える。

PostgreSQLバージョン 8.1beta1 のリリースノートによると、マルチ CPU のハードウェアに対する効率改善やバッファキャッシュのロック競合の改善がおこなわれているとのことである。今回の結果は、その効果を直接的に裏付けているわけではないが、バージョン 8.1beta1 は、内部的な効率改善がなされたため、基本性能の向上と CPU のスケールアップに対応されたバージョンであると言える。ただし、今回はベータバージョンによる測定であるため、正式リリースでの再評価が必要である。

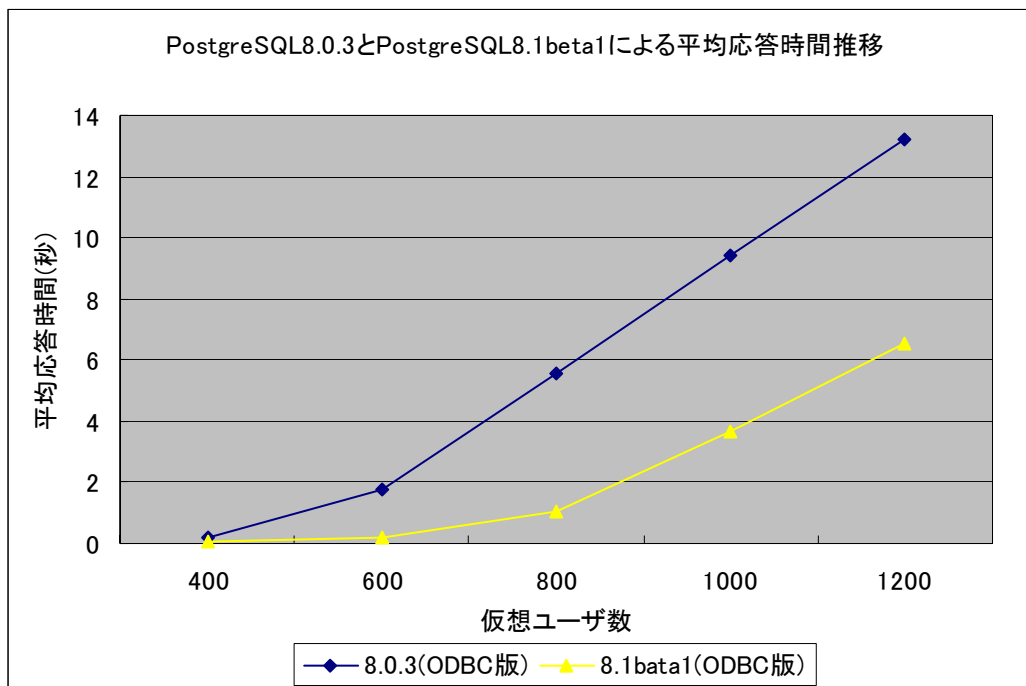


図 7.4-10 バージョン間の平均応答時間の比較推移

7.5. 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。

- PostgreSQL の ODBC 接続による性能測定ができ、評価ツールの正当性が確認できた。これにより、適用システムに応じた DBT-1 のモデルを選択し、性能評価をすることが可能になった。
- ODBC 接続で直接 SQL を実行する処理であっても、一般的なチューニングは有効である。
- 今回のモデルにおいては、ODBC 接続による直接的な SQL の実行は、ネイティブな関数を使用したストアードプロシージャよりは、システム・リソースを消費しない。ただし、ネットワーク環境下での再評価が必要である。
- マルチ CPU のハードウェアにおいては、PostgreSQL バージョン 8.1 を使用することで効率が改善される可能性がある。

8. DBT-1 による MaxDB の評価 -32bit と 64bit 環境の比較-

8.1. 概要

本評価においては、2004 年度に作成した ODBC 接続の DBT-1 を利用して、MaxDB7.5 の性能評価を行う。

8.1.1. 目的

近年、OSS のソフトウェアに対して性能面での期待を集まっている中で、どの環境下において OSS が効率よく性能を発揮するのかを検証する必要性が高まっている。

それを受けて、本章では、64bit 対応の OS やミドルウェアを使うことによって、32bit 版の物を使った場合とどのくらい性能差が出るかを検証する。

具体的には、OS としては Asianux1.0 を使用し、32bit 環境用の Asianux1.0 と EM64T 対応の Asianux1.0 for x86-64 を利用し、性能測定対象として MaxDB を DBT-1 で測定することで、どれくらいの性能差が出るかを評価する。

なお、ハードウェア構成は、全く同じものを使用し、ソフトウェア (OS および MaxDB) が 64bit 対応となるだけで出るの性能差を測定する物とする。

8.2. 環境定義書

8.2.1. 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、全てのプロセスを一台のサーバ上で稼働させる。

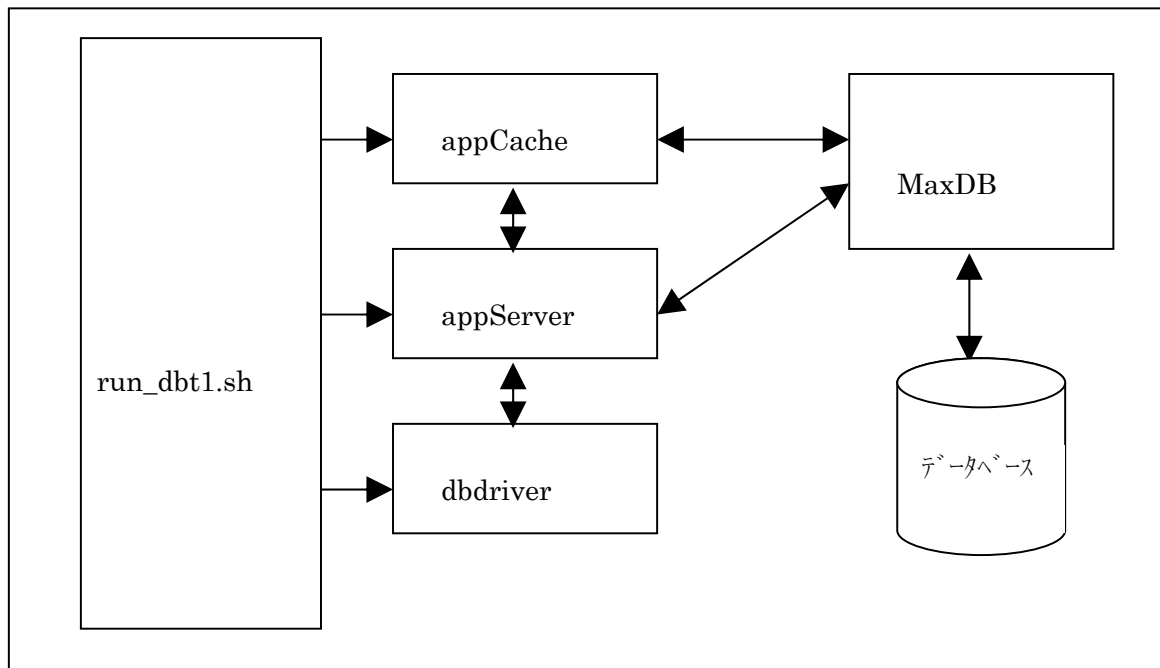


図 8.2-1 システム構成図

8.2.1.1. ハードウェア

NEC Express5800 Rg-2
Xeon 3.4GHz × 2 (ハイパースレッド有効)
MEMORY 9GB
L2 CACHE 1MB
DISK 72GB(10000rpm) × 3 (RAID5)

8.2.1.2. DISK 構成

表 8.2-1 DISK 構成

144GB	/boot	100MB
	swap	1GB
	/	残りすべて

8.2.1.3. ソフトウェア

32bitOS 環境

- Asianux v1.0 download
Linux 2.4.21-9.30AXsmp #1 SMP
Thu May 27 00:03:41 EDT 2004 i686 i686 i386 GNU/Linux
- MaxDB 7.5.0.19
- DBT-1 V2.1 改訂版(dbt1-v2.1-maxdb-1.0.tar.gz)

64bitOS 環境

- Asianux v1.0 for x86-64 download
Linux 2.4.21-9.19AX.x86_64smp #1 SMP
Thu May 27 00:03:41 EDT 2004 i686 i686 i386 GNU/Linux
- MaxDB 7.5.0.21
- DBT-1 V2.1 改訂版(dbt1-v2.1-maxdb-1.0.tar.gz)

各環境ともOSは、www.asianux.comからダウンロードしたものを利用するものとし、デフォルトインストールでパッチの適用は行わないものとする。

8.2.2. 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。

注) ssh、gcc、sysstat、libstdc++6.2.2 環境は必ずインストールする。また、シェルで測定するために、ssh 及び sudo の実行に際しては、パスワード入力が必要になるよう事前に設定する必要がある。

8.2.3. MaxDB のインストール

MaxDB のインストールおよび、環境構築に関しては、第 5 章と同様の手順にて行う。

8.2.4. パラメータの設定

8.2.4.1. OS パラメータ

今回の評価では、単純にカーネルの違いによる性能差を評価するため、OS パラメータはすべての環境において全く同じものを設定した。

OS パラメータには、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が存在する。

MaxDB を DBT-1 で測定する場合、この数が少ないと、DB 接続に失敗してしまう可能性があるので事前にチューニングを行う必要がある。

今回の評価にあたって、OS パラメータがデフォルト値の場合 DBT-1 で 1000 ユーザ以上の設定をした場合、エラーとなった。

それを受けて、以下のコマンドにて OS パラメータの”open files”を変更した。

```
# ulimit -n 1024
```

表 8.2-2 環境別 ulimit -a 状況

	32bit OS 環境	64bit OS 環境
core file size	0	0
data seg size	unlimited	unlimited
file size	unlimited	unlimited
max locked memory	4	4
max memory size	unlimited	unlimited
open files	1024	1024
pipe size	8	8
stack size	10240	10240
cpu time	unlimited	unlimited
max user processes	7168	7168
virtual memory	unlimited	unlimited

8.2.4.2. DB パラメータ

MaxDB パラメータは、2004 年度 OSS 推進フォーラム公開文書「OSS 性能・信頼性評価 / 障害解析ツール開発 DB 層の評価」の第 6 章の内容に基づいてチューニング後の環境である、以下のパラメータで実施した。

表 8.2-3 MaxDB チューニングパラメータ

パラメータ名	32bit OS 環境	64bit OS 環境
MAXCPU	4	4
CACHE_SIZE	300000	300000
CAT_CACHE_SUPPLY	250000	250000

8.3. 評価手順書

評価手順に関しては、第5章と同様の手順にて行うものとする。

8.4. 測定結果と考察

8.4.1. 測定結果

8.4.1.1. BT 値

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すべてデフォルトのまま DBT-1 を実行する。

計測は、200 ユーザから計測を始め、1000 ユーザまで 100 ユーザ単位で計測を行う。1000 ユーザから 2000 ユーザまでの間は 200 ユーザ間隔にて計測を行うものとする。

評価対象のデータは、各条件にて 4 回測定し、BT 値の平均値を算出する。パラメータの設定方法に関しては、本文書の第5章を参照のこと。

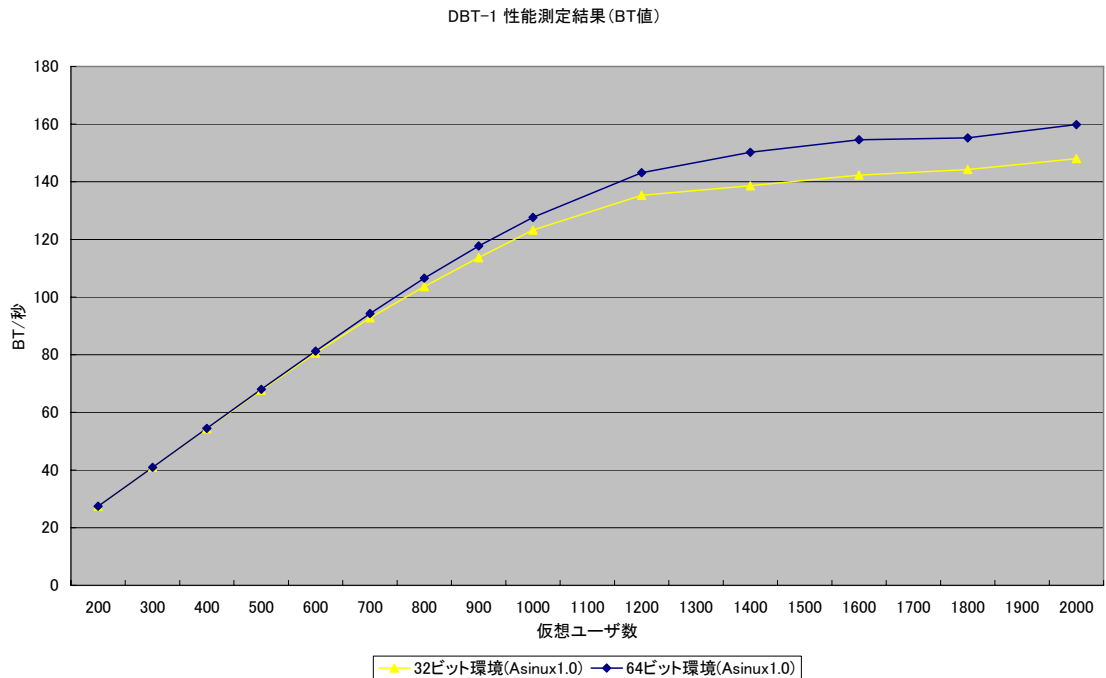


図 8.4-1 32bitOS 環境と 64bitOS 環境における BT 値の推移

図 8.4-1 は、仮想ユーザ数 200～2000 まで、測定を実施した結果のグラフである。500 ユーザまではほとんど結果に違いはみられないが、600 ユーザから BT 値に違いが出てくるのがわかる。

両者の BT 値の差は、600 ユーザから 1400 ユーザにかけて、仮想ユーザ数が増えるほど大きくなる傾向にある。この原因として検証の結果、32bitOS 環境のリソースが 64bitOS 環境に比べて不足しやすいことに起因しているためと思われることがわかった。リソースの検証結果においては、8.4.1.2 リソース消費量にて述べる。

1400 ユーザ以降に関しては、64bitOS 環境における BT 値の伸びが衰え始め、32bitOS 環境との性能差が一定となる。これは、64bitOS 環境のスループットが限界に達していることに起因しているためと思われる。

表 8.4-1 BT 値の平均値(参考)

ユーザ数	32ビット環境	64ビット環境	性能比率
200	27.400	27.450	1.002
300	40.975	40.975	1.000
400	54.325	54.425	1.002
500	67.675	67.975	1.004
600	80.600	81.225	1.008
700	92.775	94.275	1.016
800	103.60	106.525	1.028
900	113.675	117.675	1.035
1000	123.200	127.650	1.036
1200	135.275	143.150	1.058
1400	138.575	150.225	1.084
1600	142.300	154.575	1.086
1800	144.225	155.225	1.076
2000	148.000	159.850	1.080

8.4.1.2. リソース消費量

次は、同様の条件での CPU 利用率の推移である。

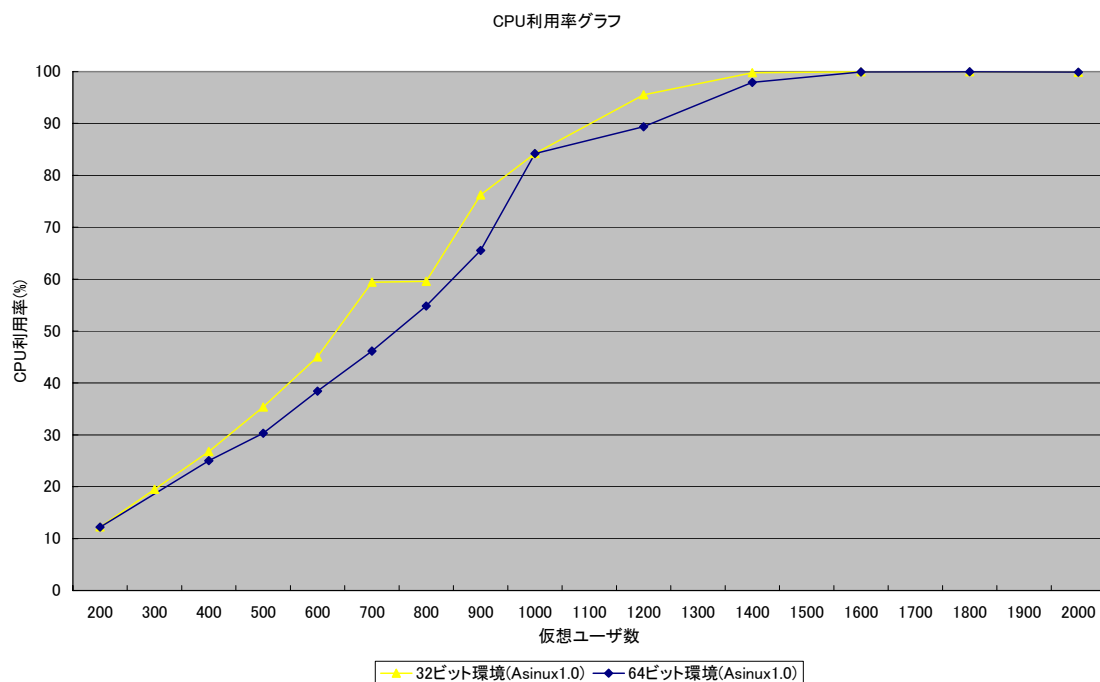


図 8.4-2 CPU 利用率グラフ

結果から、ユーザ数が少ない場合は、64bitOS 環境の方が CPU 利用率は低いですが 500 ユーザから 32bitOS 環境の CPU 利用率が急激に増え、64bitOS 環境より高くなっている。

その後も、32bitOS 環境の方が CPU 利用率が高い状態が続き、1400 ユーザでは、ほぼ 100%の状態になっていることがわかる。

次は、図 8.4-2 での CPU 利用率の結果をユーザモード利用率、システムモード利用率を表示させたものである。

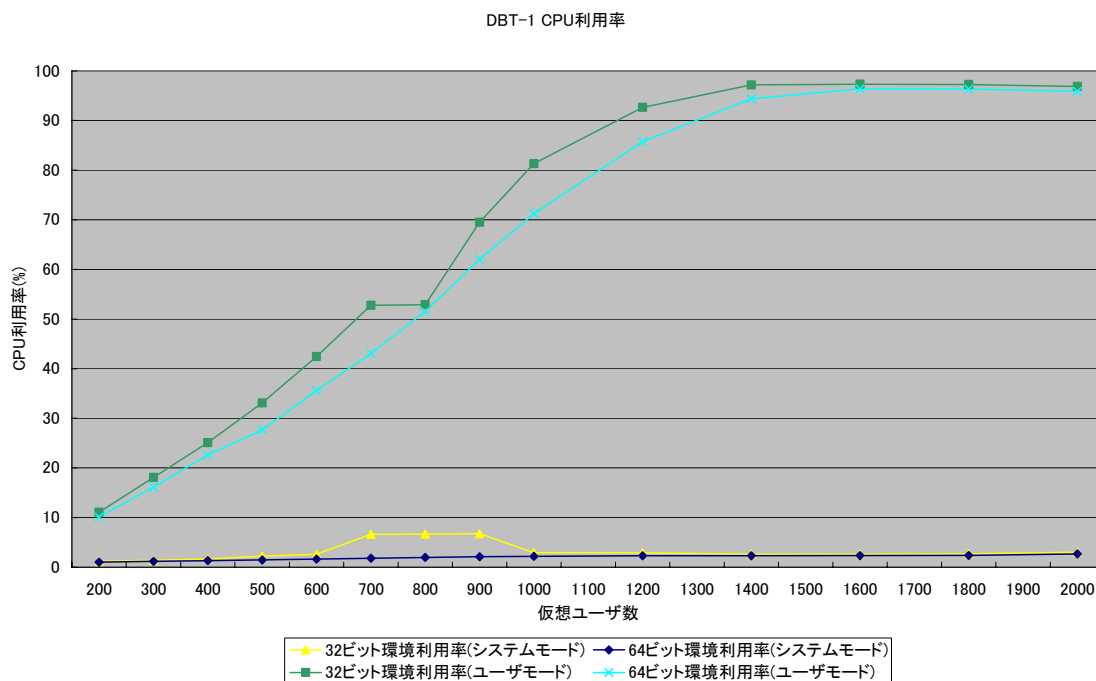


図 8.4-3 モード別 CPU 消費率

結果から、ユーザモードの CPU 利用率は、32bitOS、64bitOS 環境共にユーザ数の増加に伴い増加することがわかった。性能の比較という観点から、300 ユーザから利用率に差がみられ、ユーザ数の増加に伴って差が大きくなると考えられる。

最終的には、約 95%にて飽和状態となり、1600 ユーザにおいては利用率の差がみられなくなる。

システムモードの利用率に関しては、ユーザ数が増加してもあまり変化が見られない。

また、32bitOS、64bitOS 環境で利用率に差があまりないと言える。

8.4.2. 考察

計測を行った結果から、サーバの CPU 利用率が限界に達するまでは、32bitOS 環境、64bitOS 環境共に、ユーザ数の増加に伴って BT 値が増加する結果となった。

ただし、32bitOS 環境と、64bitOS 環境では CPU 利用率の上昇率に違いがあり、32bitOS 環境の方が CPU 利用率が上がりやすい。

そのため、ユーザ数が増加するに伴って性能差が広がり、最大で約 8%の性能差が見られた。

ただし、1400 ユーザ以上では、64bitOS 環境においても CPU 利用率が限界値に達しており、32bitOS との性能差は約 8%以上に広がらなかった。

高負荷時におけるリソースの状態に関して、メモリの使用には十分な余裕があった。スワップも使用していないことから、32bitOS 環境においてボトルネックとなっているのは、CPU であるといえる。

8.4.3. インタラクションにおける性能の違い

32bitOS,64bitOS 両環境における、各インタラクションの平均応答時間をグラフ化し違いを評価する。

仮想ユーザが 1000 ユーザの時の、インタラクション別所要平均時間を図 8.4-4 に記述する。

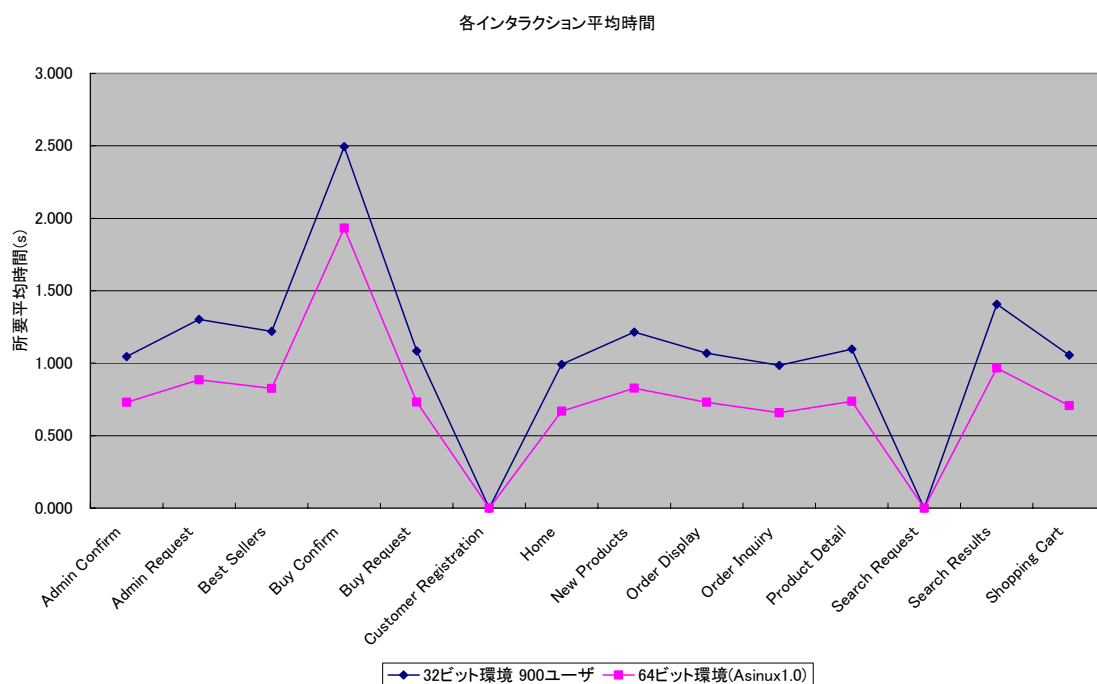


図 8.4-4 各インタラクション平均時間

32bitOS、64bitOS 両環境の性能差に関する比率は、インタラクションに関係なくほぼ一定している。

これは、インタラクションによる性能差の違いが少ないことを示している。

8.4.4. 計測データ

8.4.4.1. インタラクション平均時間

表 8.4-2 インタラクション別所要時間平均値(参考)

	32ビット環境(s)	64ビット環境(s)	性能比率
Admin Confirm	1.046	0.731	1.430
Admin Request	1.302	0.886	1.471
Best Sellers	1.220	0.827	1.476
Buy Confirm	2.494	1.933	1.290
Buy Request	1.084	0.733	1.480
Customer Registration	0.000	0.000	0.000
Home	0.992	0.669	1.482
New Products	1.215	0.827	1.469
Order Display	1.068	0.731	1.461
Order Inquiry	0.985	0.658	1.496
Product Detail	1.098	0.737	1.489
Search Request	0.000	0.000	0.000
Search Results	1.407	0.967	1.455
Shopping Cart	1.056	0.708	1.492

8.4.5. MaxDB ログから見る 32bit と 64bit の違い

MaxDB のログを解析した結果、キャッシュ領域を十分に確保したこともあり、どちらのアーキテクチャにおいても、100%のヒット率を示した。IOWait やロックステータスも内容に特徴的な差分を見つけることは出来なかった。MaxDB の m_load 出力結果より、測定開始 1000 秒後の'SQL Commands'の結果をプロットしたものを、図 8.4-5 SQLCommand 数に記す。

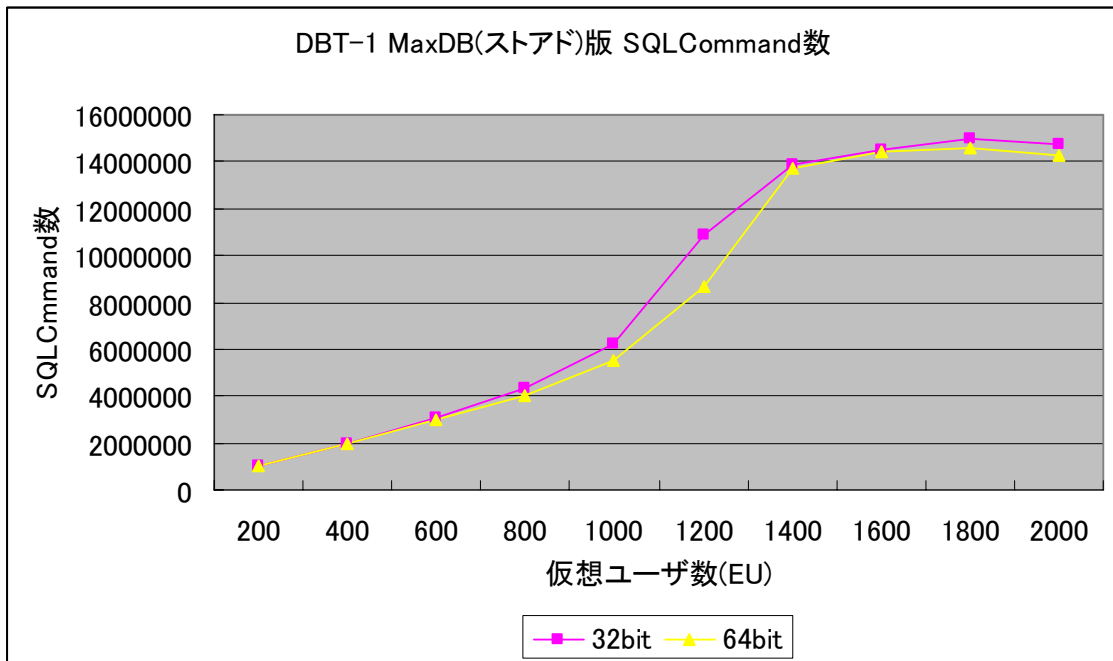


図 8.4-5 SQLCommand 数

どちらのアーキテクチャも、EU=1000 までは一定の割合で増加しているが、それ以降は急な伸びを見せている。しかし、EU=1400 以降は SQL コマンド数が伸びることはなかった。EU=1200 の時点を除けば傾向に差はないと言える。EU=1200 では大きく差が付いているが、測定開始後 1000 秒時点の値であり、若干ではあるが 2000 秒後には値は逆転していた。時間経過と共に 64bit 環境の方がより多くの SQL コマンドを処理出来ているようである。

8.5. まとめ

今回の評価結果から、今回の環境における 32bitOS 使用時と 64bitOS 使用時は、BT 値から最大約 8%ほどの性能差が出た。

この原因として、32bitOS 環境では、CPU の利用率が上昇しやすいことが、原因と考えられ、特にユーザモードの CPU 利用率において、環境の違いが出ていると言える。

なお、評価の結果として、OS の 64bit 対応の有無と、MaxDB の 64bit 対応の有無による環境の差異が今回の性能の差異となる。なお、MaxDB が 32bit 版と 64bit 版でバージョンに違いが存在するが、リリースノートから性能能力への影響はないと言える。

結果として、EM64T 対応の CPU の場合、OS と MaxDB が 64bit 対応をしているだけで 32bit のものを使用した時に比べて性能の向上が見込めることがわかった。

今回、MaxDB におけるキャッシュサイズ(CACHE_SIZE)の違いについても評価を行ったが、32bitOS 環境下においては設定値 400000 以上において、MemoryAllocateError が表示され、計測ができなかった。

今回、64bitOS 環境において、キャッシュサイズをチューニングすることによる性能向上は限定的であった。

インタラクション別の比較においては、32bitOS 環境と、64bitOS 環境との性能比率は、インタラクションが変わっても変化しなかった。

これは、両環境とも処理の内容によって性能の差に変化がないことを示していると言える。

9. DBT-3 による PostgreSQL の評価 -32bit と 64bit 環境の比較-

1.1. 概要

9.1.1. 評価概要

本評価では OSDL-DBT-3 を利用し、CPU に IA32 を使ったマシンと EM64T を使ったマシンとの性能測定を行う。

評価は、以下の 2 種類を行う。

- (1) 同じパラメータにおいて計測を複数回行い、その性能平均値とクエリ毎の特徴について考察を行う (以下、評価 1)。
- (2) 同時に SQL を実行するストリーム数の変化による影響を、IA32 と EM64T を使ったマシンとで比較する (以下、評価 2)。

9.1.2. 目的

昨年度実施した OSDL-DBT-3 評価は 32bit OS によるもののみであった。

今年度は更なるプラットフォーム拡大を目的として、64bit CPU 環境下で PostgreSQL に対して OSDL-DBT-3 テストキットが正常に動作するかを評価する。

動作確認後、32bit OS と 64bit OS による差異を見極めることを目的として、同じスペックのハードウェア上で IA32 用の OS と EM64T 用の OS の違いのみで性能測定を行う。

また、64bit OS が有効となる場合を見極めることを目的として、クエリや SQL の同時実行ストリーム数の違いによる影響の違いについても評価を行う。

9.2. 環境定義書

9.2.1. システム構成

今回、2つの環境を使って評価を行った。

表 9.2-1 評価環境（評価1）

製品	BladeSymphony(日立製作所) IA32/EM64T 環境毎にブレードを設置
CPU	Intel(R) Xeon(TM) CPU 3.40GHz × 2 [一方は EM64T 対応版 / 一方は IA32 版]
メモリ	DDR-1 8GB
ハードディスク	160GB[SATA,7200r/min] × 2 [ブレード毎独立して設置]
OS	Red Hat Enterprise Linux AS 4
RDB	PostgreSQL 8.0.3 データベースクラスタを OS と分離し専用単一ディスクに配置 クラスタとトランザクションログのディスク分割なし 統計情報収集あり、設定値チューニングなし
ODSL-DBT-3	dbt-v1.5 + pgsql 用パッチ

表 9.2-2 評価環境（評価2）

製品	NEC Express5800 Rg-2
CPU	Intel(R) Xeon(TM) CPU 3.40GHz × 2
メモリ	DDR-1 9GB
ハードディスク	72GB[SATA,10000r/min] × 3 (RAID5)
OS	Red Hat Enterprise Linux AS 4 Update 1
RDB	PostgreSQL 7.4.7 設定値チューニングなし
ODSL-DBT-3	dbt-v1.5 + pgsql 用パッチ

9.2.2. PostgreSQL のインストール

PostgreSQL のインストールおよび、環境構築に関しては、第 5 章と同様の手順にて行う。

9.2.3. ワークロード実施手順（初期セットアップ作業）

ブレードホスト上のノードを3つ使用し、EM64T であるもの、IA32 であるものを一つずつワークロード対象ノードとし、残りをコントロール用のノードとした。テスト対象ノードには OS を標準状態でインストールし、DBT-3 に必要なソフトウェアをその後でインストールした。

テスト対象ノードで `cron`, `anacron` を停止し、それ以外のプロセスは標準の起動状態のままとした（したがって X サーバ等の直接必要でないプロセスも存在している状態である）。

ODSL-DBT-3 に PostgreSQL 用のパッチを当てた後、さらに昨年度実施したときとの OS バージョン差異に対応するため、以下の修正を実施（そのままでは `sysstat` のバージョン番号が変わっているとき動作しない可能性がある）。

```
$ cd $DBT_HOME/data_collect/resulttools
$ cp sar.5.0.1.key sar.5.0.2.key
$ cp sar.5.0.1.key sar.5.0.3.key
$ cp sar.5.0.1.key sar.5.0.4.key
$ cp sar.5.0.1.key sar.5.0.5.key
```

PostgreSQL8.0.3 をソースコードからインストールし、`file` コマンドでそれぞれ 64 ビット、32 ビットの実行ファイルとしてビルドされているかを確認した。

9.2.4. ワークロード実施手順

評価手順に関しては、第5章と同様の手順にて行うものとする。

9.3. 評価結果(評価1)

スケールファクタ=1として IA32 環境、EM64T 環境に対してそれぞれ8回ワークロードを実施した。スケールファクタ=2として1回ワークロードを実施した。

実施毎に次のことを行った。

- PostgreSQL のデータベースクラスタを削除、再初期化
- DBT-3 が生成したファイル群を（必要なデータを採取した後）削除
- 残存する sadc、iostat、vmstat、sar のプロセスを KILL
 - コマンド「killall sadc iostat vmstat sar」を実行した

9.3.1. 実施状況

ワークロード実施においては、全体で5回～6回、dbt-3プログラムが停止したり、必要な処理を行わずに先に進んでしまい不正なワークロード結果を報告するケースが発生した。問題は IA32 マシン、EM64T マシンの両方で発生しており、EM64T マシン固有の動作不良は発生しなかった。

スケールファクタ=1の8回の計測のうち IA32、EM64T の各1回分は不正な結果であり有効なデータとしては7回ずつ採取できた。スケールファクタ=2で実施した場合も不正な結果で、再度行えば正常に動作することが予想できたが今回は実施しなかった。

9.3.2. 指標値

DBT-3では Composite、Power、Throughput の3つの指標値が生成される。図1はスケールファクタ=1 のときの IA32、EM64T ごと指標値の平均値である。値の次元は1時間あたりのトランザクション数である。

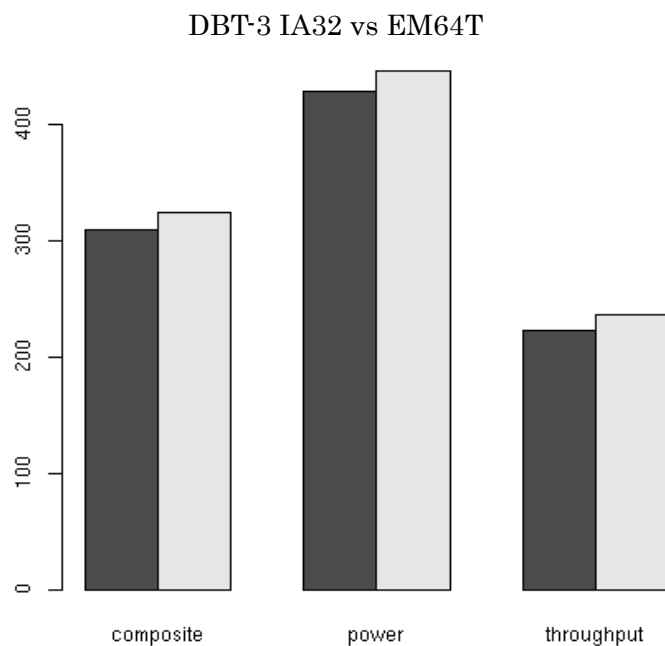


図 9.3-1 指標値の比較

表 9.3-1 指標値の比較

指標値	CPU	平均値	標準誤差	標準誤差率
Composite	IA32	308.3600	4.283394	1.39%
	EM64T	323.9943	1.1176929	0.34%
Power	IA32	427.4814	10.603251	2.48%
	EM64T	445.2143	1.9861195	0.45%
Throughput	IA32	222.6000	0.744139	0.33%
	EM64T	235.7843	0.8746615	0.37%

なお、ここで標準誤差とは $\text{標準偏差} \div (\sqrt{\text{標本数}})$ で定義されるもので、標準誤差率はその平均値に対する比率である。

9.3.3. クエリごと所要時間

DBT-3 を構成するクエリ 24 種の実行時間は、以下の図のような値となった。

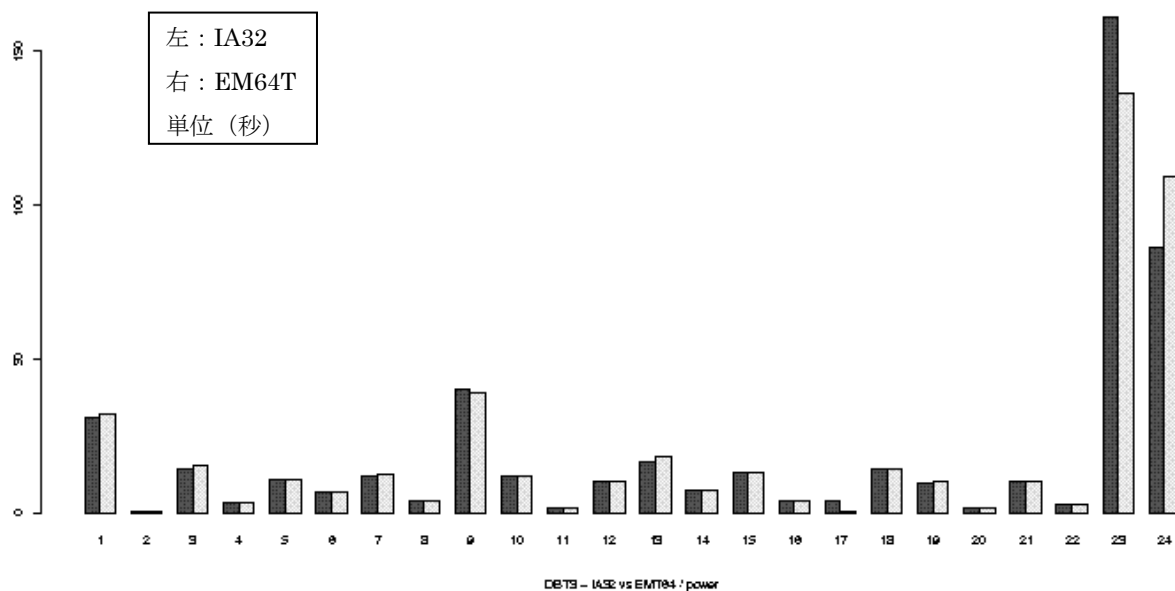


図 9.3-2 クエリ実行所要時間平均 Power テスト

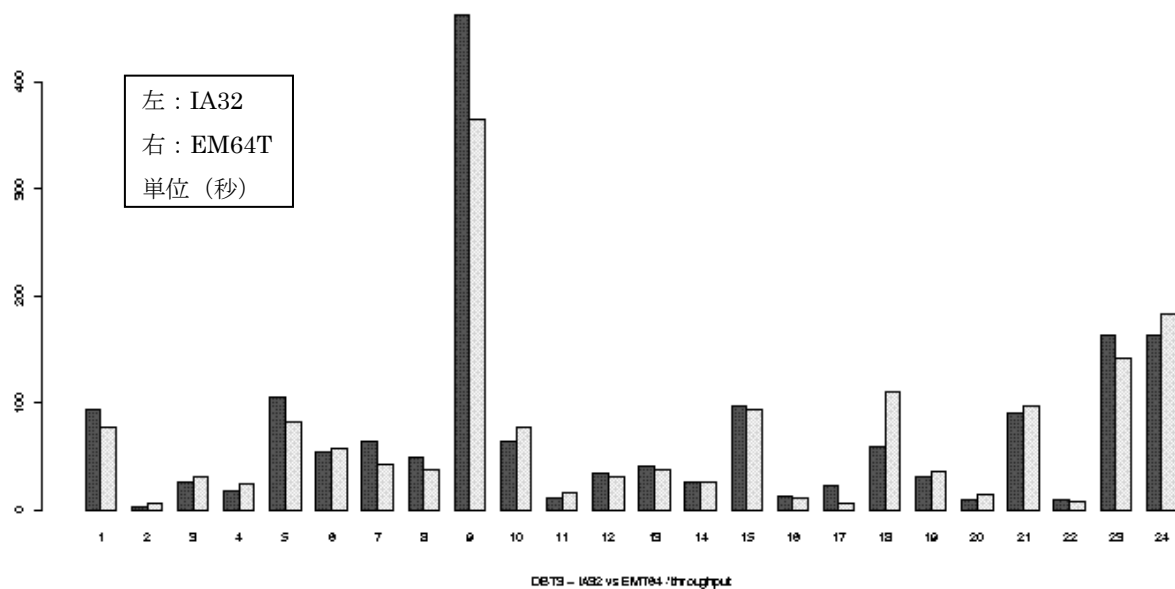


図 9.3-3 クエリ実行所要時間平均 Throughput テスト

以下にクエリ実行所要時間の標準誤差率のグラフを掲載する。スループットテストは全般にばらつきが大きく、IA32 と EM64T の違いを議論できる精度ではない。

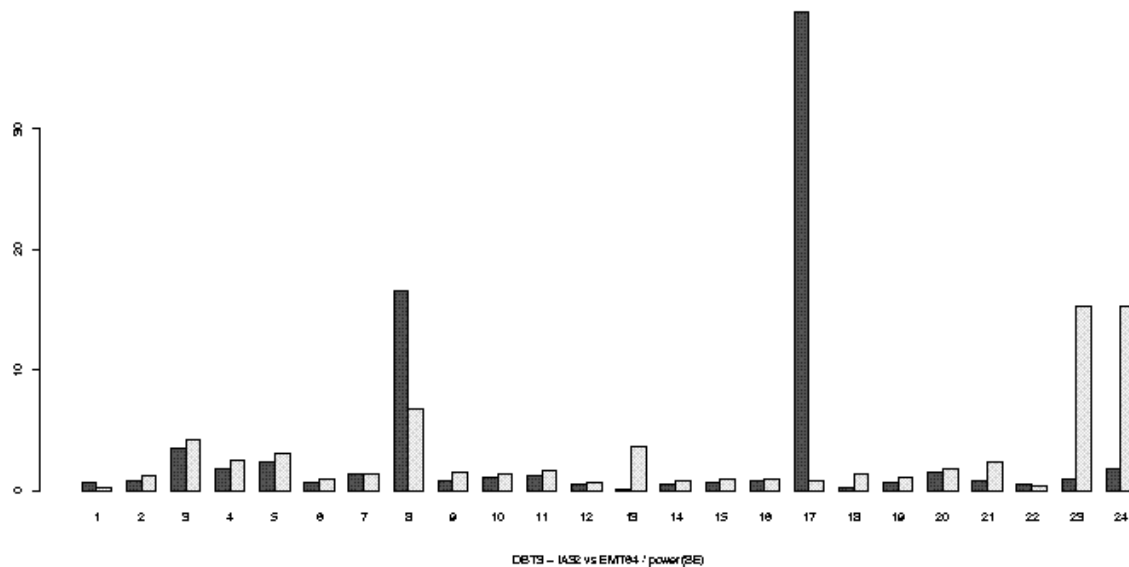


図 9.3-4 クエリ実行所要時間の標準誤差率(%) Power テスト

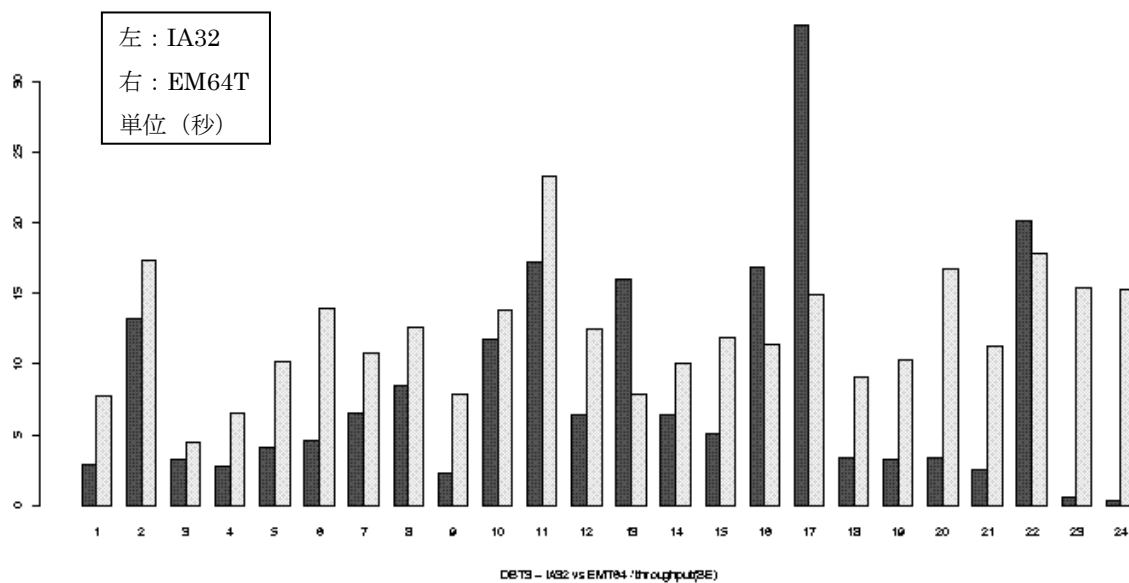


図 9.3-5 クエリごと所要時間の標準誤差率(%) Throughput テスト

パワーテストでは Q8、Q17、Q23、Q24 以外が、スループットテストでは Q3、Q4 がぶれの少ない値と考えられる。

9.3.4. 実行時システム負荷

9.3.4.1. CPU

IA32/EM64T、スループットテスト/パワーテストを問わずほぼ同じような結果が得られた。EM64Tの方が5～10%程度、ユーザプログラム処理時間の割合が大きくそれ以外が全般に小さい傾向があった。

9.3.4.2. ディスクI/O

IA32とEM64Tとで挙動に大きな違いは無い。僅かではあるがEM64Tの方が短い時間に多量のI/O処理を行い、I/O量が少ない状態に戻るのが早いという傾向が全般に見られた。

9.3.4.3. メモリ使用量

IA32でも8GBのメモリが利用できるカーネルを使用した。全テストでスワップが発生することは無く、OSのキャッシュバッファとして4GB程度が使用された。IA32とEM64Tではっきりとわかるようなメモリ使用量の違いはなかった。

9.3.5. 考察

9.3.5.1. IA32 と EM64T の差異

EM64T 対応の CPU と非対応の IA32CPU を比べたとき前者のメリットは以下の 2 点に集約できる。

1. 64 ビットレジスタが利用できる
2. 大きなメモリ空間が利用できる

このうち 1. はコンパイラやアプリケーションのソースコードに影響される。今回は PostgreSQL を IA32 での手順書通りにインストールして評価したので Red Hat Enterprise Linux AS 4 のパッケージに含まれる gcc でデフォルト状態でビルドした。コンパイラにより良いものに換える、最適化指定を強くする、などで EM64T のメリットを引き出せる可能性がある。

また、2. は EM64T 非対応の IA32 プロセッサであっても 36 ビット (64GB) まではアドレスバスが拡張されており、それほど大きな差が出ない。今回の評価においても両マシンともメモリ 8GB を利用できる状態であった。したがってメリットは以下に集約される。

- OS 上の 4GB 超メモリ扱いのオーバーヘッドが軽減
- 1 プロセスにおいて 4GB 超メモリがハンドリングできる
- 64GB 超の物理メモリを扱える

PostgreSQL は「個々の postgres プロセスの使うメモリ」「SysV 共有メモリ」「OS のバッファキャッシュ」の 3 種類のメモリを使用する。このうち前者 2 つが制御可能であるが「SysV 共有メモリ」は大きくしすぎても遅くなることが知られている。「個々の postgres プロセスが使用するメモリ」を大きくとるようになれば DBT-3 のパワーテスト（およびそれに類する複雑巨大なクエリを 1 接続だけで実行する類）でより良いパフォーマンスが出ることは期待できる。

なお、メモリ容量を増やさずにスケールファクタを大きくした場合、データベース内容全体がメモリに載らなくなり、ディスク I/O が律速になるため、IA32 と EM64T との差異はより小さくなると考えられる。逆に 64GB 超のメモリが必要な程度にスケールファクタを大きくしてやれば IA32 だけがディスク I/O 律速になり大きな差異が予想される。

9.3.5.2. 差異のある処理

DBT-3 を構成するいくつかのクエリで比較的大きな差異が表れた。それらはどのようなものか。以下は IA32 と EM64T で 20% 以上の差異があるクエリである。これらは Q9 を除き全て誤差率が 10% 以上のデータに該当し、大きな差異は偶発的な要因による結果の可能性も考慮しなければいけない。

Q9 Product Type Profit Measure Query (スループットテストで EM64T 優越)
集計ソート (結果行数大きい)。

Q17 Small-Quantity-Order Revenue Query 多段集計。	(両テストで EM64T が優越)
Q18 Large Volume Customer Query 多段集計とソート (結果行数少ない)。	(スループットテストで IA32 が優越)
Q23 Refresh Function 1 レコード追加処理。	(両テストで EM64T が優越)
Q24 Refresh Function 2 レコード削除処理。	(両テストで IA32 が優越)

これらクエリの特徴と IA32、EM64T の違いとを結びつける論拠は得られていない。

9.3.6. スクリプトの問題点

OSDL-DBT3 v1.5 (for PostgreSQL のパッチをあてたもの) の実行スクリプトにはいくつかの問題点がみられた。

1. 実行中に止まってしまうことがある
2. 工程をスキップして異常に高パフォーマンス値になることがある

1.については具体的な原因が判明している。エラーメッセージとして `postmaster` に接続できないという旨が出ている。これは `scripts/pgsql/start_db.sh` 内で「`pg_ctl start`」コマンドで `postmaster` を起動して、その後「`sleep 10`」とするだけで、`postmaster` を起動したとみなしていることが原因で、未だ起動していない状態で `psql` を実行することでエラーになっていると考えられる。代わりに `pg_ctl` コマンドに `-w` オプションをつけることで処理の完了を 60 秒待つようになるので、ある程度は回避可能である。

2.についてはリフレッシュ処理がほぼ即時に完了して実際には処理が行われないという現象に遭遇している。こちらは `/tmp/` 以下のファイルが開けない、というエラーが出ている。スクリプトは `/tmp` 以下にファイルを生成した直後に `SQL` としてそのファイルを読みに行くという部分に問題がある可能性がある。

これらの修正には DBT-3 としての実行結果に影響がでるかなどの考慮をしたうえでの評価作業が必要と考えられる。

9.4. 評価結果(評価2)

9.4.1. 測定結果

各 IA32、EM64T 環境にて DBT-3 の計測を行った。

計測にあたっては、スケールファクタを1、ストリーム数を変化させることによって、両環境での違いを比較した。

評価結果としては、性能評価中、特に問題なく動作した。

EM64T 環境下においても、IA32 と同様の手順にて動作することを確認できた。

9.4.1.1. 指標値

次は、DBT-3 の指標値の推移である。

Processing Power はパワーテストの結果、Throughput Numerical Quantity はスループットテストの結果である。

両者とも、ストリーム数の増加に伴い値が低下する傾向にあるが、最大で 10%程度の差がみられたが、ストリーム数との関連性はみいだせなかった。

傾向に関しては、両環境ともに同じ傾向を示しており、環境に特化した特徴は見られなかった。

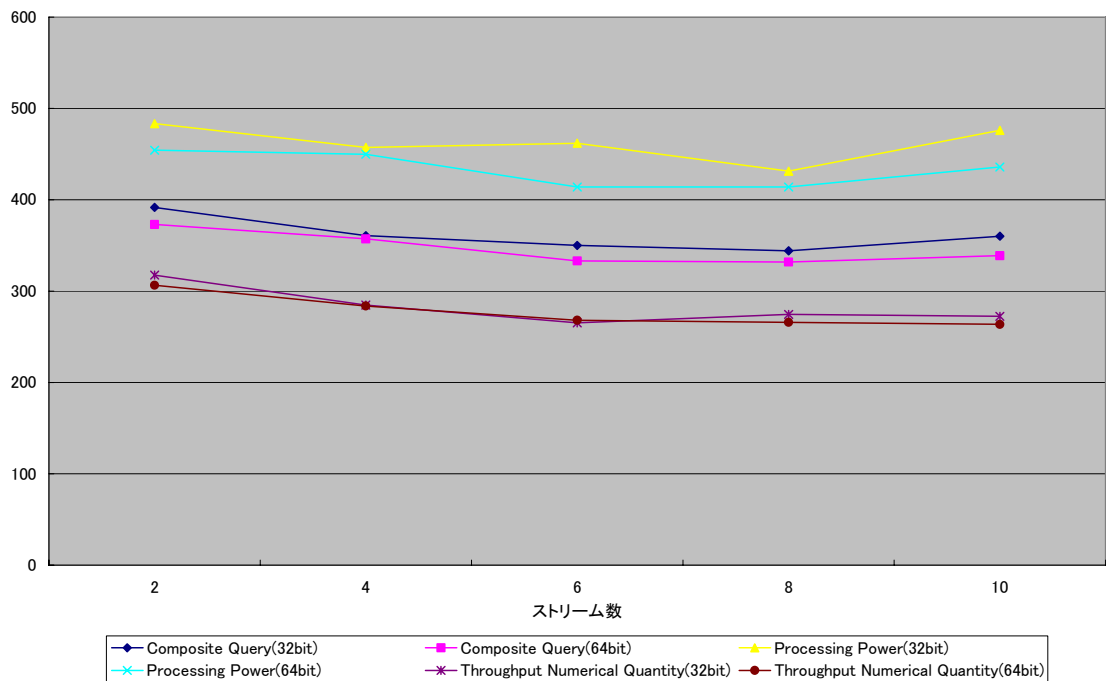


図 9.4-1 32bit OS 環境と 64bit OS 環境における BT 値の推移

表 9.4-1 Composite Query

	32ビット環境(s)	64ビット環境(s)	性能比率
2	391.70	373.10	1.050
4	360.80	357.19	1.010
6	350.01	333.16	1.051
8	344.07	331.86	1.037
10	360.08	338.92	1.062

表 9.4-2 Processing Power

	32ビット環境(s)	64ビット環境(s)	性能比率
2	483.34	454.35	1.064
4	457.34	449.85	1.017
6	461.97	413.90	1.116
8	431.43	414.04	1.042
10	476.06	435.82	1.092

表 9.4-3 Throughput Numerical Quantity

	32ビット環境(s)	64ビット環境(s)	性能比率
2	317.43	306.38	1.036
4	284.64	283.62	1.004
6	265.18	268.17	0.989
8	274.40	265.99	1.032
10	272.35	263.56	1.033

9.4.1.2. IA64 での評価

今回、同様の評価を IA64 環境においても行った。

インテル® Itanium® 2 プロセッサ 1.3 GHz×8

Memory 9GB

Red Hat Enterprise Linux AS release 4 Update 1

PostgreSQL 7.4.7

dbt-v1.5 + pgsq1 用パッチ

基本的に評価 2 の環境とは、ハードウェアスペックが違うため、性能値については比較することができないが、PostgreSQL+DBT-3 の評価は正常に動作することを確認すること

ができた。

9.4.2. 考察

結果として、今回の結果としては、IA32 と EM64T の性能値について、最大 10%の差がみられた。ただ、ストリーム数と性能比率に関連性はみられなく、ほとんど差がない計測結果の時もあった。

今後、EM64T がどのような場面で性能を発揮することが可能であるのか、またどの程度の性能値までを出すことが可能なのかについては、今回の評価とは違った視点でのチューニングを含め、評価を行うことが必要であるといえる。

9.5. まとめ

本評価の結果、同じスペックのハードウェア環境下において、OS の 64bit 対応の有無による性能の差異について、PostgreSQL+DBT-3 の場合、最大約 5%ほど 64bit に対応している環境の方が優位であることが確認できた。

これは、チューニングを行わない状態でも EM64T 対応の CPU の場合、EM64T に対応した OS を使うだけで、性能の向上が期待できるということを示しているといえる。

よって、今後 PostgreSQL のチューニング含めて、今回とは別の観点での検証を行うことによって、更なる優位性を期待することができ、その点において検証していく必要がある。

10. 付録資料一覧

10.1. DBT-1 の MySQL 用改変

- ・ DBT-1 グラフ作成ツール (graph.pl)
- ・ 同 マニュアル
- ・ DBT-1 MySQL (ODBC) 版

10.2. DBT-1 の PostgreSQL 用改変

- ・ DBT-1 PostgreSQL (ODBC) 版