OSDL DBT-1 による PostgreSQL のチューニング

(2006 年 5 月 26 日版)

日本 OSS 推進フォーラム　サーバー部会　技術評価 TF

1 概要

Web トランザクション処理モデルにおける PostgreSQL のチューニングポイントの検証を目的として，OSDL DBT-1 PostgreSQL(ODBC 版) (以下，「DBT1」) を用いて性能評価を実施した。以下に，チューニング内容と効果について，結果を報告する。なお、この評価は(株)日立製作所により実施した。

2 チューニング前提条件

本評価では，データベースに関わる項目（データベース設定項目，SQL の分析，アクセス統計情報の分析）を中心としたチューニングを実施しトランザクション性能を評価する。ただし，データベースパラメタと関係しないカーネルパラメタ，ファイルシステム，ハードウェア設定などの要素は固定とする。また，DBT1 自身には手を加えないこととしてチューニングを実施する。

3 チューニングの進め方

全体をⅠ～Ⅲの 3 フェーズに分け，順に実施する。フェーズⅠではデータベースパラメタ（postgresql.conf の各設定項目）の概略的なチューニングを行い，そのチューニング結果を反映した環境でフェーズⅡ，フェーズⅢの詳細チューニングを段階的に進める。

表 3-1 フェーズごとのチューニング方法

フェーズ	チューニング方法
Ⅰ	Postgresql.conf の各設定項目（SQL 実行プラン／統計情報分析を除く）個別のチューニング効果を把握するために，探針的な測定を実施する。特定の設定項目を単独で変更し，全ての設定項目がデフォルト値の環境の測定結果と比較することによりチューニング効果を検証する。
Ⅱ	SQL 実行プラン・統計情報の分析にもとづくチューニングを実施する。当該チューニングは，フェーズⅠの探針測定の結果，効果の確認された設定項目を反映した環境で実施する。
Ⅲ	フェーズⅠ，Ⅱのチューニング結果を反映した環境で，フェーズⅠの設定項目に従い，総合チューニングを実施する。（フェーズⅠ，Ⅱのチューニングによりスループットが向上し，デフォルト環境下では効果の現れなかった設定項目があった場合の再調整）

フェーズⅠのデータベース設定項目に関するチューニングでは，PostgreSQL のドキュメントから性能に関わりが深いと予測した項目を抽出した。DBT1 のチューニングに無関係か，通常の運用では使用しないオプションと考えられる項目についてはチューニング評価の観点から除外した（表 3-2 のアミカケで示した部分）。

表 3-2 設定ファイル(postgresql.conf)のチューニング項目と OS(カーネル)のチューニング項目

大項目	詳細項目（括弧内はデフォルト値を示す）		チューニング内容
データベースパラメタ（postgresql.conf）の設定値の変更	接続	max_connections(100)	本評価ではチューニング対象とせず，固定で測定する。
	メモリ	shared_buffers(1000)	デフォルト値 1000 面から上限値に向かって変化させながら評価する。
		work_mem(1024kb)	デフォルト値の 10 倍の値（10240KB）を指定して様子をみる。
		max_stack_depth(2048kb)	極端に複雑な SQL は実行しないので，デフォルト値固定とする。
		temp_buffers(1000) max_prepared_transactions(5) maintenance_work_mem(16384kb)	DBT-1 では使用しない機能に関するパラメタなので，チューニング対象外とする。

	解放領域マップ	max_fsm_pages(20000)	性能に影響は無いと予想されるが，デフォルト値では不足するため，30000で評価する。
		max_fsm_relations(1000)	デフォルト値のままとする。(max_fsm_pages で調整する)
	カーネル使用資源	max_files_per_process(1000)	200～300 程度に減らして測定してみる。
	バックグラウンドライタ関連	bewriter_delay(200msec)	ディスクへの書き込み頻度を調節するパラメタである。パーセンテージは固定化し，ページ数で調整する。
		8.1：bewriter_lru_percent(10%)	
		8.0：bewriter_percent(10%)	
		8.1：bewriter_lru_maxpages(5 ページ)	
		8.0：bewriter_maxpages(100 ページ)	
		8.1：bewriter_all_percent(0.333%)	
		8.1：bewriter_all_maxpages(5%)	
	WAL 関連	fsync(on)	デフォルトは on である。PostgreSQL のドキュメントでは，off にすることによりデータベースを破壊する危険性がある旨が記載されているため，参考値として測定するに留める。
		wal_sync_method (※環境により異なるが本測定環境では fdatasync であった)	指定値を変化させながら評価
		full_page_writes(on)	fsync と同様
		wal_buffers(8)	16，32 と増加させて様子を見る。
		commit_delay(0), commit_siblings(5)	commit_delay を 10，50 にして様子を見る。
	ロック管理	deadlock_timeout(1000msec)	トランザクション実行時間の最大値を上回る値（100000）を指定して様子を見る。
		max_locks_per_transaction(64)	1000 で測定を試みる。
	チェックポイント関連	checkpoint_segments(3), checkpoint_timeout(300sec)	checkpoint_segments を大きめの値に固定し，checkpoint_timeout を変化させてチェックポイント回数を調節する。
	プランナ関連のパラメタ		SQL 実行プランの分析結果をもとにチューニングする。
SQL 実行プラン／統計情報の分析	インデクス使用状況，ソート発生状況の分析など(詳細は別途)		分析結果を元にしたインデクス定義変更／追加，DB パラメタチューニング
データベース物理配置	WAL ログ（更新ログ）		データベース本体と格納先を分ける。
	データベース本体		テーブルスペース，テーブルパーティションの活用による複数ディスクへの分割格納。 今回は単純な構成での評価に留め，この項目は実施しない。
OS（カーネル）環境のチューニング	メモリ	shmmax（33554432）	shared_buffers の調整に伴い，1G バイトに増やして測定する。
	I／O	ファイルアクセス時間の更新指定	noatime オプションを指定し，更新時間を取得しないようにする。

		I／0 スケジューラの指定	elevator オプションに deadline を指定して測定してみる。

4 環境定義

4.1 システム構成

4.1.1 ハードウェア

本評価で使用したハードウェアの構成を，**表 4.1-1**に示す。

表 4.1-1 ハードウェア構成

製品名	HITACHI HA8000 110W HB
プロセッサ	インテル Xeon プロセッサ 3.2GHz，2CPU HT off
メモリ	2Gbyte
ハードディスク	内蔵ドライブベイに 73GB×3(Ultara320 SCSI)

4.1.2 ディスク構成

本評価で使用したディスクの構成を，**表 4.1-1**2 に示す。

表 4.1-2 本評価のディスク構成

DISK 容量	容量	マウントポイント
73GB	60GB	/
	100MB	/boot
73GB	70GB	/data
73GB	40GB	/data2
	30GB	/home

- データベースは/data ディレクトリ下に格納し，容量は約 7GB。
- wal ログの格納先を変更する測定ケースでは，/data2 ディレクトリを使用。

4.1.3 ソフトウェア

本評価で使用したソフトウェアの構成を，**表 4.1-1**3 に示す。

表 4.1-3 本評価のソフトウェア

OS	Red Hat Enterprise Linux AS release 4 (Nahant) Update1 2.6.9-11.Elsmp
RDB	PostgreSQL 8.0.6 PostgreSQL 8.1.2
ベンチマークツール	OSDL DBT-1 PostgreSQL(ODBC)版 （dbt1-v2.1-PostgreSQL-ODBC-1.2.1.tar.gz）
odbc ドライバ	iODBC 3.51.2 psqlodbc8.00.0102

4.2 環境構築と測定手順

2005 度の成果報告書の「DB 層～OSDL DBT-1/3 による DBMS 評価編～」報告書の第 5 章（以下,「前報告書第 5 章」）からの変更点を以下に示す。

- Linuxディストリビューションは**表 4.1-3**のOSの欄に示すものを使用した。インストールは当該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に従いインストールした。

- DBT1 用オプション設定
  /etc/security/limits.conf に以下の行を追加。

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

- SELinux の無効化設定
  /etc/selinux/config の"SELINUX"オプションを"enforcing"から"disabled"
  に変更する。

```
SELINUX=disabled
```

- ODBC ドライバは iODBC を以下の手順によりインストールした。

```
$ su root                  (pgsql でログインした状態)
# mkdir /usr/local/src/libiodbc-3.51.2
# chown pgsql.pgsql /usr/local/src/libiodbc-3.51.2
# exit
$ cd /usr/local/src
$ tar xfzv /home/pgsql/libiodbc-3.51.2.tar.gz
$ cd libiodbc-3.51.2/
$ ./configure
$ make
$ make check   <-- libiodbc-3.51.2/配下のディレクトリ全てで,
                   "Nothing to be done for 'check'"と"Nothing to be done
                   for 'check-am'"になれば良い。
$ su root
# make install
```

dbt1.configの設定値については, 前報告書第 5 章と異なる設定をした場合は, 5章以下の各項で説明する。

5 探針測定(フェーズⅠ)

5.1 概要

各チューニング項目の効果を概略的に把握することを目的として，探針測定を実施した。

5.2 測定手順

以下の順序で測定を推進した。

- デフォルトの設定項目値のままの環境を基礎環境として，総エミュレーションユーザ数（以下,「eu」）を変化させながら DBT1 を実行させ，bogotranzactions per second(以下,「BT／s」)の変化が横ばいとなる eu 値を比較基準値とする。
- 当該基準点で，基礎環境に対して特定チューニング項目値を変化させて DBT1 を実行する（表 3-2 参照)。当該実行結果の BT/s と，基礎環境の BT/s と比較し，チューニング効果を検証する。
- 最終段階では，チューニング効果のあった設定項目を全て組み合わせた環境で eu を変化させて DBT1

を実行する。

前報告書第５章の DBT1 実行スクリプト・ファイル（run_dbt1.sh）を使用して測定を行う。測定結果は上記スクリプト・ファイルの引数に指定した出力用ディレクトリに出力される。

5.3 前準備

チューニング前後のBT/sの差異を認識する比較基準点を決定するために，デフォルト値の基礎環境でDBT1 を実行した。euは 400 から 200 単位で 2600 まで変化させて測定した結果を図 5.3-1に示す。

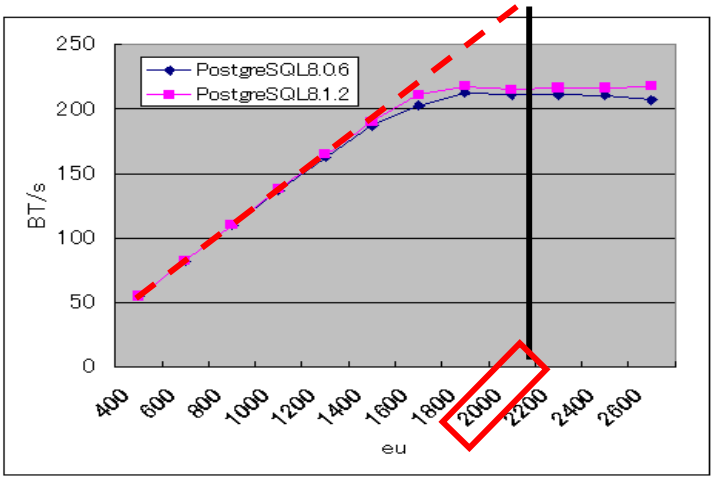

図 5.3-1 デフォルト値による BT／s と評価する eu 値

BT/s は，PostgreSQL 8.0.6，PostgreSQL 8.1.2 とも eu＝1600 付近までほぼ線形に伸びているが，それ以上では変化が鈍り，eu=2000 以上では横ばいとなった。そこで，eu＝2000 を測定ポイントと定め，探針測定を実施することとした。

5.4 測定結果

探針測定の結果を表 5.4-1に示す。アミカケ部がチューニング効果があった項目である。

表 5.4-1 探針測定結果

設定項目	オプション値	Ver8.0.6		Ver8.1.2	
		BT/s	ﾃﾞﾌｫﾙﾄ比(※1)	BT/s	ﾃﾞﾌｫﾙﾄ比
shared_buffers(※2)	500	178.2	0.87	200.6	0.95
	1000	207.3	1.01	213.7	1.02
	5000	229.1	1.11	269.4	1.28
	10000	226.9	1.10	270.0	1.28
	50000	222.9	1.08	268.3	1.28
	100000	222.9	1.08	270.2	1.28
work_mem	64	199.8	0.95	207.3	0.96
	10240	212.1	1.00	220.1	1.02
max_fsm_pages	30000(※3)	211.7	1.00	217.9	1.01
max_files_per_process	50	212.8	1.01	216.5	1.01
	5000	210.9	1.00	218.6	1.02
8.0 bgwriter_delay,-① bgwriter_percent,-②	①200, ②0, ③0	213.4	1.01		
	①200, ②100, ③5	211.2	1.00		

	bgwriter_maxpages,-③	①200, ②100, ③50	213.5	1.01		
8.1	bgwriter_delay, ,-① bgwriter_lru_percent,-② ,bgwriter_lru_maxpages,-③ bgwriter_all_percent,-④ bgwriter_all_maxpages-⑤	①200,②0,③0,④0,⑤0 ①200,②100,③5,④0,⑤0 ①200,②100,③50,④0,⑤0			215.4 218.1 218.2	1.00 1.01 1.02
fsync		off	209.2	0.99	271.3	1.26
full_page_writes		Off			210.0	0.98
wal_sync_method		fsync	212.0	1.00	216.6	1.01
		open_sync	207.6	0.98	261.4	1.22
wal_buffers		16	212.2	1.00	217.8	1.01
		32	213.4	1.01	217.8	1.01
		64	212.1	1.00	214.7	1.00
commit_delay, ,,-①		①10, ②5	212.8	1.01	213.1	0.99
commit_siblings,-②		①50, ②5	211.1	1.00	217.6	1.01
checkpoint_segments,,-①		①30, ②100	211.4	1.00	217.6	1.01
		①30, ②,600	212.3	1.00	214.8	1.00
checkpoint_timeout,-②		①30, ②,900	211.9	1.00	218.1	1.01
deadlock_timeout		100000	211.6	1.00	215.8	1.00
max_locks_per_transaction		300	211.9	1.00	213.8	0.99
wal ログ格納ディスク変更			206.4	0.98	268.7	1.25
noatime オプション			211.8	1.03	210.5	0.98
Elevator=deadline			210.5	1.00	214.7	1.00

※1 shared_buffers はPostgreSQL の統計情報を取得しながらの評価のため，同様にデフォルト環境の値も統計情報を取得しながら評価した値（eu=2000）で比較した。

PostgreSQL 8.0.6 211.3 (205.6)

PostgreSQL 8.1.2 214.9 (210.3)

※2 shmmax は，1073741824 に固定して測定した。

※3 DB 初期ロードのシェル実行時に"NOTICE: number of page slots needed (28352) exceeds max_fsm_pages (20000)”のメッセージが出力されていたため，30000 で試行した。

以上の結果から，チューニング効果の高かった項目値を組み合わせた環境（表 5.5-1）で，euを変化させDBT1を実行した。PostgreSQL 8.0.6 は，wal_sync_method，walログの格納先を変更しても顕著な効果が出なかったが，PostgreSQL 8.1.2 の結果から有効と判断した。

チューニング効果の高かった設定項目値を組み合わせたチューニング後環境の DBT1 実行結果を，デフォルト環境の実行結果と比較して図 5.4-1 に示す。なお，設定項目チューニング後は BT／s が向上したため，eu を 3400 まで拡大し，dbt1.config の run_duration を 2400 から 4100 に延長して，有効な DBT1 実行時間を確保した。

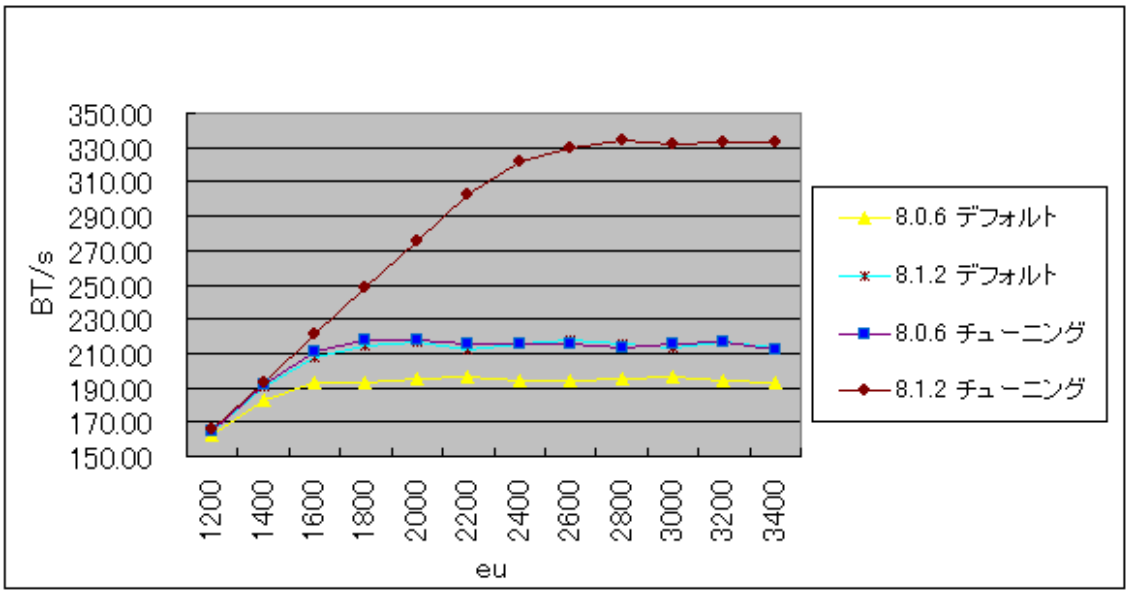

図　5.4-1 チューニング環境とデフォルト環境の比較

5.5　まとめ

探針測定（フェーズⅠ）ではshared_buffers，wal_sync_method，walログの格納先の 3 項目についてチューニング効果があった。探針測定の結果を表 5.5-1に示す。

表 5.5-1　探針測定（フェーズⅠ）の結果

	チューニング項目	PostgreSQL 8.0.6	PostgreSQL 8.1.2
探針測定によるチューニング内容	shared_buffers	10000	100000
	wal_sync_method	open_sync	open_sync
	wal ログ格納先	DB 本体と異なるディスク	DB 本体と異なるディスク
	shmmax	33554432	1073741824
BT 値 (eu＝3200)	デフォルト	193.80	216.20
	探針測定チューニング後	216.20	333.50
	向上率	11%	54%

6　SQL 実行プランの分析に基づくチューニング（フェーズⅡ）

6.1　概要

SQL 実行プランの分析による性能チューニングを，以下の手順で実施する。

- SQL 実行プラン，統計情報の分析による問題点抽出
- 分析結果にもとづくチューニング方法の検討
- チューニング実施，および効果の検証

実施環境は，フェーズI完了時点の表 5.5-1に示す環境とする。

6.2 SQL 実行プランの分析

6.2.1 DBT1 の SQL 分類

DBT1(PostgreSQL用ODBC版)のSQL文の分類を表 6.2-1に示す。

表 6.2-1 SQL 文種別の内訳

SQL 種別	該当 SQL 数	備考
SELECT	39	表 6.2-2に詳細に分類する。
UPDATE	9	全 SQL について WHERE 句にプライマリキーの全構成列に対する＝条件が指定されている。
DELETE	1	WHERE 句にプライマリキーの全構成列に対する＝条件が指定されている。
INSERT	9	全て INSERT VALUES 指定。 SQL 実行プランは単純で分析余地が無いと考える。
計	58	

SELECT 文で指定されていた詳細な分類を表 6.2-2 に示す。単一テーブルを WHERE 句の＝条件で絞り込んで検索するパターンが最も多かった。また，DISTINCT，GROUP BY 句，集合演算，SQL 関数や複雑な副問合せの指定は無く，比較的単純なパターンが多かった。

表 6.2-2 SELECT 文の詳細分類

項番	FROM 句表数	WHERE 句の条件	グループ化	ORDER BY	該当 SQL 数	備考
1	0(FROM 句無し)	WHERE 句無し	無し	無し	5	シーケンス値取得
2	1	＝条件	有り	無し	3	何れも暗黙的グループ化
3	1	＝条件＋OR 演算	無し	無し	1	副問合せ 5 個有り
4	1	無し	無し	有り	1	"LIMIT 1"の指定あり
5	1	＝条件（＋AND 演算）	無し	無し	14	
6	2	＝条件（＋AND 演算）	無し	無し	8	
7	2	＝条件（＋AND 演算）	無し	有り	5	LIKE 部分一致条件を含む SQL2 個有り
8	3	＝条件＋AND 演算	無し	無し	1	
9	3	＝条件＋AND 演算	無し	有り	1	
	計				39	

SQL 文の形式で示すと次のようになる。

SELECT 選択項目[, …]
[FROM テーブル名称[,…]]
[WHERE 条件 1(※) [{AND｜OR} 条件 2 …]
[HAVING 条件]
[ORDER BY ソート列 1[,…]] [LIMIT 1]

※ WHERE 句の条件は＝条件または LIKE 条件の何れか。

使用されていたSQL文から抽出したSQL実行プランのチューニングに関する分析観点を表 6.2-3に示す。

表 6.2-3 分析観点

項番	SQL の使用状況		分析観点
1	FROM 句	テーブル指定のみ。単一テーブルの検索が最も多かったが，ジョイン（最高 3 テーブル）も指定されているものもあった。	ジョイン方式 ソート発生状況（Merge Join）
2	WHERE 句	＝条件，LIKE 条件，及びそれらの AND／OR 演算	インデクス使用状況（テーブル検索方式）
3	グループ化（集合関数，HAVING 句）	暗黙的グループ化のみ (GROUP BY 句の指定無し)	特になし。（実行プランにチューニング余地が無いと判断した。）
4	ORDER BY 句	列によるソート指定のみ。ソート方向の指定（ASC/DESC）の組み合わせ有り。	ソート発生状況
5	LIMIT 句	1 SQL のみ LIMIT 1 の指定がある。	特になし。

58 個の SQL（付録 A 参照）中で INSERT 文，FROM 句の無い SELECT 文は，SQL 実行プランが単純で分析の余地が無いため解析対象から除外。更に，重複した SQL を 1 つにまとめ，最終的に 41 個の SQL を分析対象とした。

6.2.2 分析観点と対策方針

表6.2.1の分析観点に対し，対策の対象とすべきSQL文の絞込みに使用するキーワードとその対策方針を表6.2-4にまとめる。

表 6.2-4 分析観点と対策方針

項番	分析観点	探索キーワード	検討方針	検討内容
1	インデクス使用状況（テーブル検索方式）	Seq scan	Seq scan を Index scan または Bitmap Index scan にできないか	①使用させたいインデクスが定義済みの場合は random_page_cost を調整する。 ②適切なインデクスが無い場合は，インデクス定義の改善を検討する。
2		Index scan	使用されているインデクスは妥当か	①使用させたいインデクスが定義済みの場合は，データベース統計情報を見直す。 ②適切なインデクスが無い場合は，インデクス定義の改善を検討する。
3	ソート発生状況	Sort	Sort は回避可能か	① Merge Join に伴う Sort は enable_mergejoin を on から off にして，Nested Loop または Hash Join にすることにより回避できないか検討する。 ②order by 指定に伴う Sort は enable_sort を on から off にすることにより回避することを試みる。（※①の対策により自然に回避される可能性もある） ③上記①②でも回避できない Sort は，インデクス定義の見直しを試みる。
4	ジョイン方式	Nested Loop Merge Join Hash Join	他のジョイン方式と比較して性能はよいか	enable_nestjoin，enable_mergejoin，enable_hashjoin，work_mem を調整してジョイン方式の変更を試みる。

Seq scan 及び Sort は，I／O コスト，CPU コストに対する影響が大きいと判断し，これらの観点での分析を優先した。そして，その後に Index scan，ジョイン方式での分析を行った。

6.2.3 解析手順

以下の手順でSQL実行プランを解析した。

① DBT1のソースプログラム中からSQL文を取り出す。

② PostgreSQLのログ情報を参考にして変数部分に妥当な定数値を当てはめる。
postgresql.confのlog_min_duration_statementを-1から0に変更し，
SQL文とSQL実行時間をPostgreSQLのログファイルに取得する。

③ ログに取得されたSQL文を参照し，定数値を取り出して①のSQLの変数部に埋め込み，EXPLAIN文で解析する。

6.2.4 解析結果

DBT1をSeq scanで検索した結果, PostgreSQL8.0.6は6件, PostgreSQL8.1.2は4件, Sortで検索した結果, PostgreSQL8.0.6，PostgreSQL8.1.2とも5件のSQL文が該当した。そのうち，テーブル容量，条件のヒット件数からみて，実行コストが小さいと予測できるものを除外し，チューニング対象として絞り込んだSQL文を表6.2-5に示す。当該4つのSQL文以外に関しては，Index scan，ジョイン方式の観点でも分析を行ったが，特に問題は無かった。

なお，Q20，Q35については，インデクスの使用状況にバージョン間の差異があった。（PostgreSQL 8.0.6ではSeq scanとなるが，PostgreSQL 8.1.2ではIndex scanとなっていた。）

表 6.2-5 チューニング対象SQL

SQL番号	SQLテキスト	SQL実行プラン分析		
		Seq scan	Sort	ジョイン方式
Q20	SELECT i_id, i_title, a_fname, a_lname FROM item, author WHERE i_subject = '%s' AND i_a_id = a_id ORDER BY i_pub_date DESC, i_title ASC;	無(8.0) 有(8.1)	有	Merge Join
Q34	SELECT i_id, i_title, a_fname, a_lname FROM item, author WHERE i_a_id=a_id AND a_lname LIKE '%%%s%%' ORDER BY i_title ASC;	無	有	Nested Loop
Q35	SELECT i_id, i_title, a_fname, a_lname FROM item, author WHERE i_subject = '%s' AND i_a_id = a_id ORDER BY i_title ASC;	無(8.0) 有(8.1)	有	Merge Join
Q36	SELECT i_id, i_title, a_fname, a_lname FROM item, author WHERE i_title LIKE '%%%s%%' AND i_a_id = a_id ORDER BY i_title ASC;	無	無	Nested Loop

次に，これら4SQLについて問題点の明確化と対策内容の検討を行った。結果を

表 6.2-6に示す。

表 6.2-6　問題点と対策検討結果

SQL 番号	問題点と対策
Q20	＜PostgreSQL 8.0.6＞ 問題点：item テーブルの検索にインデクスが使用されない。 対策：2004 年度の成果報告書のチューニング方法を参考にして，PostgreSQL のパラメタを以下のように変更し，インデクスが使用されやすくする。 random_page_cost を 4（デフォルト）から 2 に変更。 ※　PostgreSQL 8.1 ではインデクスが使用されていたため問題なし。 ＜PostgreSQL 8.0.6，PostgreSQL 8.1.2＞ 問題点：ソートが発生している。 対策：ソート回避の検討。
Q34	＜PostgreSQL 8.0.6，PostgreSQL 8.1.2＞ 問題点：author 表の検索にインデクスが使用されない。 対策：LIKE 述語の部分一致検索を使用しているため，対策不可。 ＜PostgreSQL 8.0.6，PostgreSQL 8.1.2＞ 問題点：ソート(Sort)が発生している。（LIKE 述語のヒット率が不明のため件数が予測できず） 対策：ソート回避の検討。
Q35	＜PostgreSQL 8.0.6＞ 問題点：item テーブルの検索にインデクスが使用されない。 対策：2004 年度の成果報告書のチューニング方法を参考にして，PostgreSQL のパラメタを次のように変更し，インデクスが使用されやすくする。 random_page_cost を 4（デフォルト）から 2 に変更 ※　PostgreSQL8.1.2 ではインデクスが使用されていたため、問題なし。 ＜PostgreSQL 8.0.6，PostgreSQL 8.1.2＞ 問題点：ソートが発生している。 対策：ソート回避の検討。
Q36	＜PostgreSQL 8.0.6，PostgreSQL 8.1.2＞ 問題点：item 表の検索にインデクスが使用されない。 対策：LIKE 述語の部分一致検索を使用しているため，対策不可。 ＜PostgreSQL 8.0.6，PostgreSQL 8.1.2＞ 問題点：ソートが発生している。（LIKE 述語のヒット率が不明のため件数が予測できず） 対策：ソート回避の検討。

6.2.5　ログの分析による裏づけ

DBT1 の rampup 時間終了直後から約 10 分間 postgresql.conf の log_min_duration_statement をデフォルトの−1（ログを取得しない）から 0（実行時間が 0ms 以上の SQL について SQL 実行時間と SQL 文をログに取得する）に変更し，得られたログを集計したところ，チューニング対象の 4SQL の実行時間が全体に占める割合は PostgreSQL 8.0.6 で 82%，PostgreSQL 8.1.2 で 65%となることが分かった。

表 6.2-7　ログ集計結果

	PostgreSQL 8.0.6(ミリ秒)	PostgreSQL 8.1.2(ミリ秒)
①4SQL 計	47886099	1482021
②全 SQL	58540409	958441
比率（②／①）	0.82	0.65

6.3 チューニング内容および効果の検証

PostgreSQLのバージョン別に，チューニング内容を経過時間に沿ってケース化し，その効果とを表 6.3-1，表 6.3-2に示す。euは 3200 を測定ポイントとし，各ケースについてBT/sを測定した。太線で囲ったケースがBT/sが最も向上した場合である。

表 6.3-1　PostgreSQL 8.0.6 チューニング内容とその効果

測定ケース名	チューニング内容	チューニングの効果	
		BT／s	SQL 実行プランの変化，他
チューン前		216.2	
CASE1_8.0	Seq scan を Index scan にすることを目的に，random_page_cost 4 を 2 に変更	284.1	Q20，Q35 において Seq scan は Index scan に改善された。 （Q34,Q36 は LIKE 部分一致であるため Seq scan のまま改善はされず。）
CASE2_8.0	CASE1_8.0 の状態でジョイン方式を変更することを目的に，更に enable_nestloop on を off に変更	10.1	Q34，Q36 で　Nested Loop は Merge Join になった。
CASE3_8.0	CASE1_8.0 の状態で，ジョイン方式の変更，及び Merge Join に伴うソートの回避を目的に，更に enable_mergejoin on を off に変更	276.0	Q20，Q35 で　Merge Join は Nested Loop になった。
CASE4_8.0	CASE3_8.0 の状態で，ORDER BY のソート回避を目的に，更に enable_sort on を off に変更	120.8	Q34，Q35，Q36 について，order by 指定に伴う Sort が回避されたが，条件判定が行われず。
CASE5_8.0	CASE1_8.0 の状態で，HashJoin の性能検証を目的に enable_mergejoin on を off，enable_nestloop on を off，work_mem 1024 を 102400 に変更	118.3	HashJoin にならず。（全て Nested Loop になり，Q20 のみソートが発生した。）

CASE4'_8.0	CASE4_8.0 の状態で，Q20，Q35 の「条件評価＋order by のソート回避」，を同時に行わせることを目的に，インデクス定義を改善（※）した。 また，Q34 では i_i_title インデクスを使用すると性能が劣化したため，このインデクスを削除した。（Q34，Q36 以外の SQL では i_i_title インデクスを使用しておらず，i_i_title インデクスを削除しても Q36 の単体性能に悪影響は無し）	207.3	Q20，Q35 については，条件判定のみにインデクスが使用され，order by のソートは回避されず。
CASE6_8.0	CASE2_8.0 の状態で，ORDER BY のソート回避を目的に，更に enable_sort on を off に変更	16.0	

（※）インデクス改善内容を以下に示す。

```
CREATE OR REPLACE FUNCTION datecmprev(DATE,DATE) RETURNS INTEGER AS $$ SELECT
date_cmp($2,$1);$$ LANGUAGE sql;
CREATE OPERATOR CLASS date_desc_ops FOR TYPE date USING BTREE AS OPERATOR 1 >, OPERATOR 2 >=,
OPERATOR 3 =, OPERATOR 4 <=, OPERATOR 5 <,FUNCTION 1 datecmprev(date,date);
drop index i_i_subject;
create index i_i_subject1 on item (i_subject,i_pub_date date_desc_ops,i_title);
create index i_i_subject2 on item (i_subject,i_title);
drop index i_i_title;
```

表 6.3-2　PostgreSQL 8.1.2 チューニング内容とその効果

測定ケース名	チューニング内容	チューニングの効果	
		BT／s	SQL 実行プランの変化
チューン前		333.5	
CASE1_8.1	ジョイン方式の変更，Merge Join に伴うソートの回避を目的に，enable_mergejoin on を off に変更	346.0	Q20，Q35 で Merge Join は Nested Loop になった。
CASE2_8.1	CASE1_8.1 の状態で random_page_cost 4 を 2 に変更	340.2	変化なし
CASE3_8.1	ジョイン方式の変更を目的に，enable_nestloop on を off に変更	2.1	Q34，Q36 で，Nested Loop は Merge Join になった。
CASE4_8.1	CASE1_8.1 の状態で，ORDER BY 指定に伴うソートの回避を目的に，enable_sort on を off に変更	216.3	Q34，Q35，Q36 について，order by 指定に伴う Sort が回避されたが，条件判定が行われなくなった。
CASE5_8.1	HashJoin の性能検証を目的として，enable_mergejoin on を off，enable_nestloop on を off，work_mem 1024 を 102400 に	217.7	HashJoin にならず。（4SQL とも Nested Loop になった。）

	変更		
CASE4'_8.1	CASE4_8.1 において，Q20，Q35 の「条件評価＋order by のソート回避」，を同時に行わせることを目的に，インデクス定義を改善(※)した。また，Q34 では i_i_title インデクスを使用すると性能が劣化したため，このインデクスを削除した。（Q34，Q36 以外の SQL 文は i_i_title インデクスを不使用だったため，i_i_title インデクスを削除しても Q36 の単体性能に負の影響は無かった）	354.0	Q20，Q35 において，インデクスが条件判定と order by のソート回避に使用された。
CASE5_8.1	CASE3_8.1 の状態で，ORDER BY のソート回避を目的に，更に enable_sort on を off に変更	28.9	

（※）改善内容は CASE4'_8.0 と同じ。

SQL単体性能の変化をBT／sが一番高い値を示したCASE1_8.0，CASE4'_8.1 とチューニング前との比較で表6.3-3に示す。

表 6.3-3　チューニング前後の SQL 単体性能の変化

	PostgreSQL 8.0.6			PostgreSQL 8.1.2		
	①チューニング前(ms)	②チューニング後(ms)(CASE1_8.0)	比率(①／②)	①チューニング前(ms)	②チューニング後(ms)(CASE4'_8.1)	比率(①／②)
Q20	25.176	13.435	1.87	14.018	6.541	2.14
Q34	6.513	6.857	0.95	5.531	5.680	0.97
Q35	24.492	12.802	1.91	13.640	6.287	2.17
Q36	40.102	39.726	1.01	32.652	31.208	1.05

最後に，上記 4 SQL 文以外の SQL 文に対するチューニングの影響を調査するため，CASE1_8.0，CASE4'_8.1 の環境で，41SQL 文について SQL 実行プランを再確認し，チューニング前の実行プランと比較した。

その結果，PostgreSQL8.0.6 は差異が無かった。また，PostgreSQL8.1.2 は下記 SQL（付録 A の Q30）において orders テーブルの検索方法が Bitmap Index scan から Index scan へと変化していた（使用インデクスは同じ）。PostgreSQL のログで下記 SQL 文の実行時間を確認したところ，SQL 実行時間全体に占める割合は約１％であったため，当該 SQL 文の実行プランが変化しても，BT/s への影響は小さいと考える。

```
SELECT c_fname, c_lname, c_phone, c_email,o_id, o_date, o_sub_total, o_tax,o_total, o_ship_type,
o_ship_date,o_status, o_bill_addr_id, o_ship_addr_id,cx_type, cx_auth_id FROM customer, orders,
cc_xacts WHERE c_uname ='ULULRINGATBA' AND c_passwd='ululringatba' AND o_c_id =c_id AND cx_o_id
= o_id ORDER BY o_date DESC;
```

その他の SQL についても，SQL 中の定数値が変わると SQL 実行プランが変化する可能性はあるが，SQL の条件に指定されている列についてはデータベース統計情報（pg_stats テーブル）のデータ分布情報に大きな片寄りが見られなかったことと，上記 4SQL 文以外の SQL に指定されている条件は，＝条件の組み合わせのみであったことから，データベース統計情報から算出される条件ヒット率は SQL 中の定数値が変わっても極端に変動しないと考えられること，の２点から，BT/s への影響は小さいと考える。

但し，今回の評価ではアプリケーションを変更せず，静的に postgresql.conf の設定項目を変更する前提でチューニングを実施したため，全く BT/s に影響しないわけではない。確実な方法は，PostgreSQL の SET 文によりチューニング対象 SQL 文の前後で，当該設定項目値の変更及びその戻しを実行することである。また，enable_mergejoin，enable_sort などの設定項目をサーバ全体で off にすることは，PostgreSQL のプランナが SQL 実行プランを選択する際の選択肢を狭めていることになるので，実運用では設定項目に応じて SET 文を用いた局所的なチューニングを行うことが好ましい。

6.4 考察

探針測定とSQL実行プラン分析によるチューニングの結果を全て反映した環境（表 6.4-1）において，euを変化させてBT／sを測定した結果を，探針後の環境，デフォルト環境での結果と共に以下に示す。探針測定の結果を反映したチューニング状態から，eu＝3200 において，PostgreSQL 8.0.6 で 31%，PostgreSQL 8.1.2 で 6%向上した。

両バージョンを比較すると，デフォルト状態でインデクスが使用されるようになった SQL があったことや，マルチカラムインデクスの指定で性能が改善された状況から，PostgreSQL 8.1.2 でプランナが改良されたと言える。

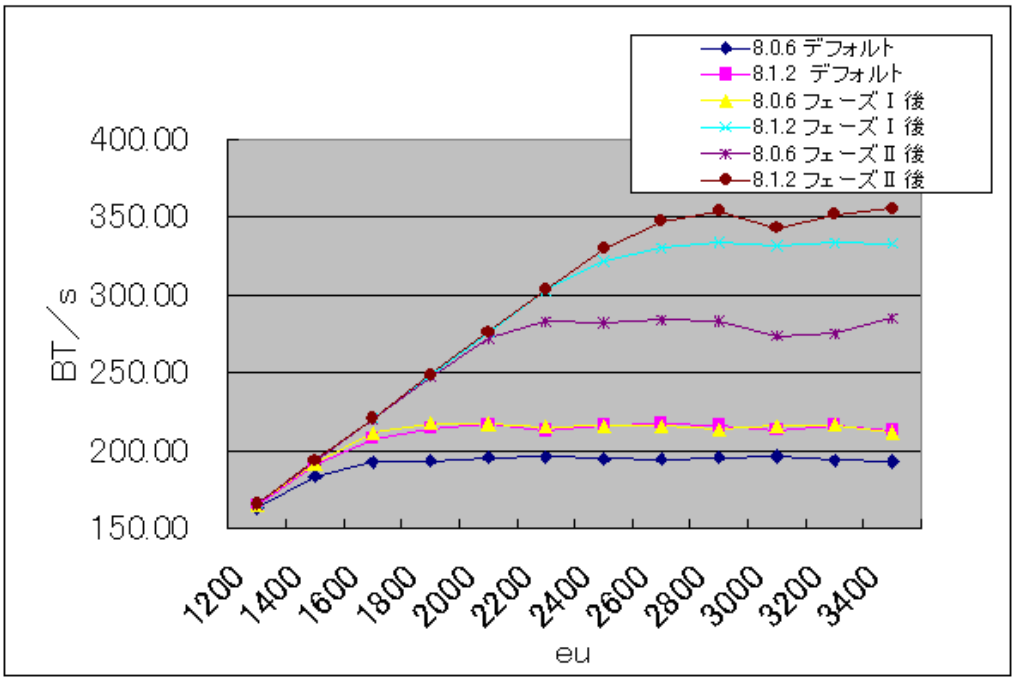

表 6.4-1　チューニング後測定環境

6.5 まとめ

SQL実行プランの分析に基づくチューニング（フェーズⅡ）ではrandom_page_cost ， enable_mergejoin ， enable_sort の 3 項目を調整することでてチューニング効果を出すことができた。探針測定の結果を表 5.5-1に示す。

	項番	チューニング観点（設定項目）	改善内容（設定項目値）	
			PostgreSQL 8.0.6	PostgreSQL 8.1.2
SQL 実行プラン分析によるチューニング内容	1	random_page_cost	2	4（デフォルト）
	2	enable_mergejoin	on（デフォルト）	Off
	3	enable_sort	on（デフォルト）	Off
	4	インデクス定義改善	デフォルトのまま(改善点なし)	条件判定とソートを同時に行えるインデクスを追加
BT／s (eu＝3200)		フェーズⅠ後	216.2	333.5
		フェーズⅡ後	284.1	354.0
		向上率	31%	6%

図 6.5-1　探針（フェーズⅠ）後の環境と SQL 実行プランの分析に基づくチューニング（フェーズⅡ）後環境の比較

7　総合チューニング(フェーズⅢ)

7.1　測定方法

SQL 実行プランの分析に基づくチューニング結果を反映させた環境において，探針測定（フェーズⅠ）で評価した各設定項目を再度微調整しながら測定し，BT/s の向上効果を検証する。チューニング効果が明確になるよう検証のポイントを eu＝3600 として測定した。また，DBT1 の実行時間（run_duration）は SQL 実行プラン分析によるチューニング時と同様 4100 とした。

7.2　測定結果

本チューニング（フェーズⅢ）の測定結果を表 7.2-1に示す。各設定項目を微調整したが変化した設定項目は無かった。

なお，max_fsm_pages，max_files_per_process，max_locks_per_transaction は，探針測定の結果とパラメタの性質から，さらなる性能向上には寄与しないと判断し，また，wal_sync_method＝open_sync，wal ログ格納先の変更は，探針測定でチューニング効果が既明のため，それぞれ固定値として測定した。

表 7.2-1　本チューニング測定結果

パラメタ	オプション値	8.0.6		8.1.2	
		BT/s	向上率	BT/s	向上率
shared_buffers	1000	225.80	0.79	256.10	0.72
	3000	285.10	1.00	—	—
	5000	285.80	1.00	357.50	1.00
	8000	285.60	1.00	—	—
	10000	—	—	358.50	1.00
	50000	—	—	357.00	1.00
work_mem	10240	283.70	1.00	360.50	1.01
max_fsm_pages	50000 固定				
max_files_per_process	デフォルト固定				
8.0 bgwriter_delay-①	①200②0　③0	279.20	0.98		

	bgwriter_percent-②	①200②100③5	272.00	0.95		
	bgwriter_maxpages-③	①200②100③50	278.70	0.98		
8.1	bgwriter_delay-①	①200, ②0, ③0, ④0, ⑤0			358.30	1.00
	bgwriter_lru_percent-②	①200, ②100, ③5, ④0, ⑤0			349.50	0.98
	bgwriter_lru_maxpages-③ bgwriter_all_percent-④ bgwriter_all_maxpages-⑤	①200, ②100, ③50, ④0, ⑤0			351.30	0.98
wal_sync_method		open_sync 固定				
wal_buffers		16	282.60	0.99	358.60	1.00
commit_delay-①		①10　②5	284.10	1.00	352.40	0.99
commit_siblings-②		①50　②5	279.60	0.98	349.10	0.98
checkpoint_segments-① checkpoint_timeout-②		①30　②600	284.30	1.00	355.90	0.99
deadlock_timeout		100000	282.40	0.99	354.90	0.99
max_locks_per_transaction		デフォルト固定				
wal ログ格納先		DB 本体と分ける				
noatime オプション			279.30	0.98	352.60	0.99
elevator=deadline			276.20	0.97	353.80	0.99
パラメタ変更前（基準値）			285.00		357.70	

7.2.1　総合結果

総合チューニング後の BT/s とデフォルト環境のと比較として，BT/s は図 7.2-1 に，PostgreSQL8.0.6 のインタラクションごとの応答時間は図 7.2-2，図 7.2-3 に，PostgreSQL8.1.2 のインタラクションごとの応答時間は図 7.2-4，図 7.2-5 に，PostgreSQL8.0.6 の CPU 使用率を図 7.2-６，図 7.2-7 に，PostgreSQL8.1.2 の CPU 使用率を図 7.2-8，図 7.2-9 に，それぞれ示す。

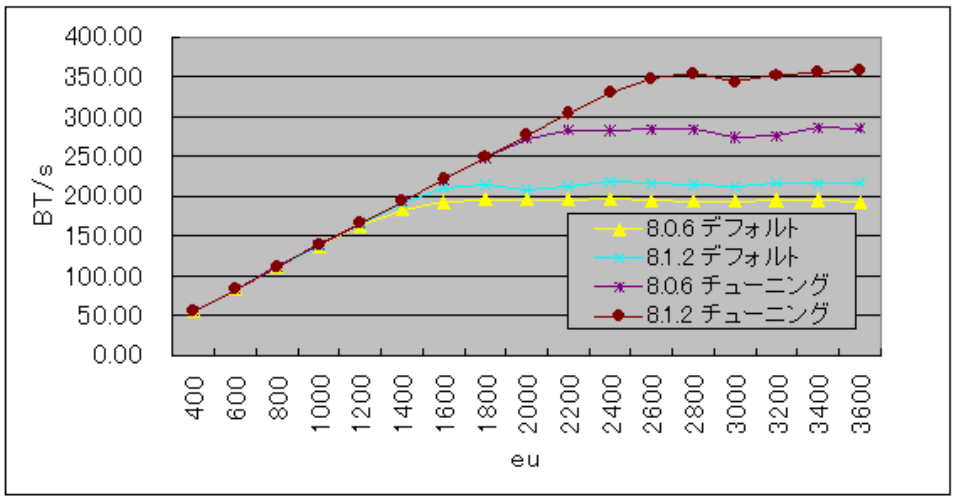

図 7.2-1　デフォルト環境と最終チューニング後環境の比較

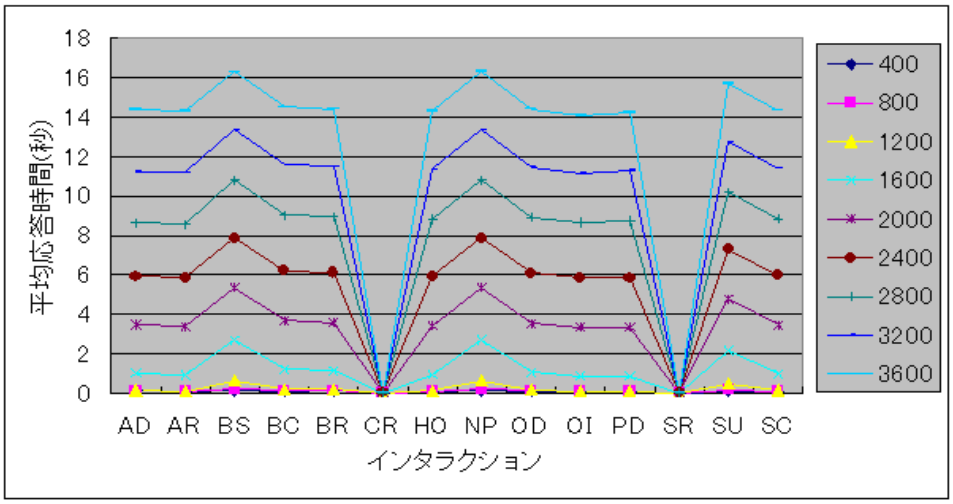

図 7.2-2　平均応答時間（PostgreSQL 8.0.6 デフォルト環境）

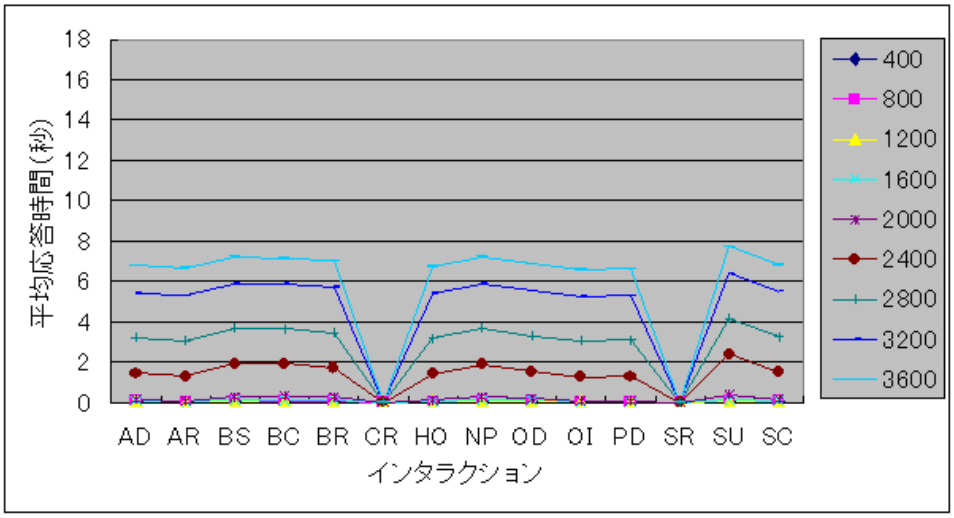

図 7.2-3　平均応答時間（PostgreSQL 8.0.6 チューニング後環境）

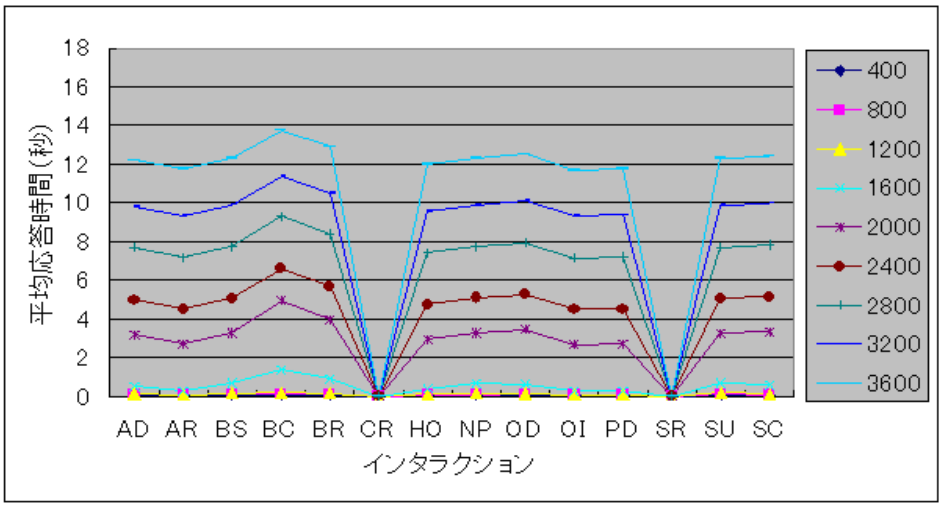

図 7.2-4　平均応答時間（PostgreSQL 8.1.2 デフォルト環境）

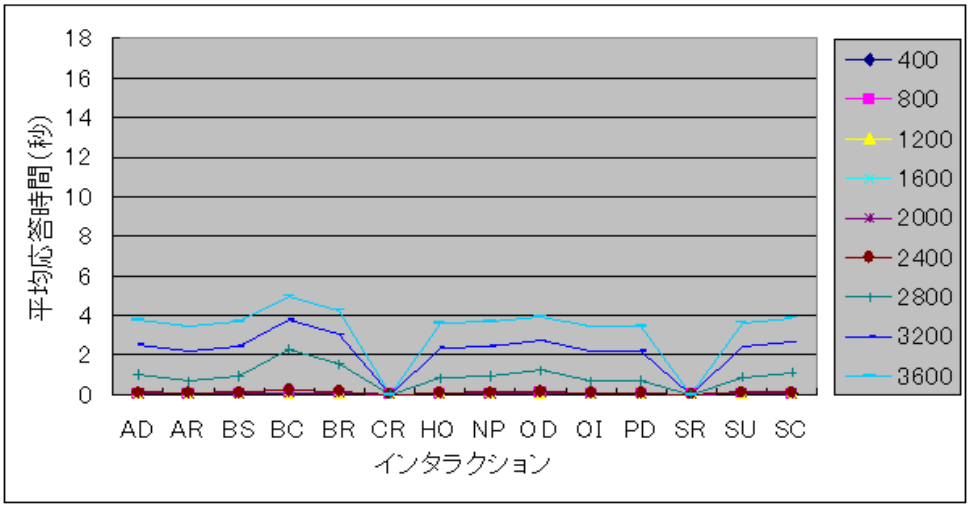

図 7.2-5　平均応答時間（PostgreSQL 8.1.2 チューニング後環境）

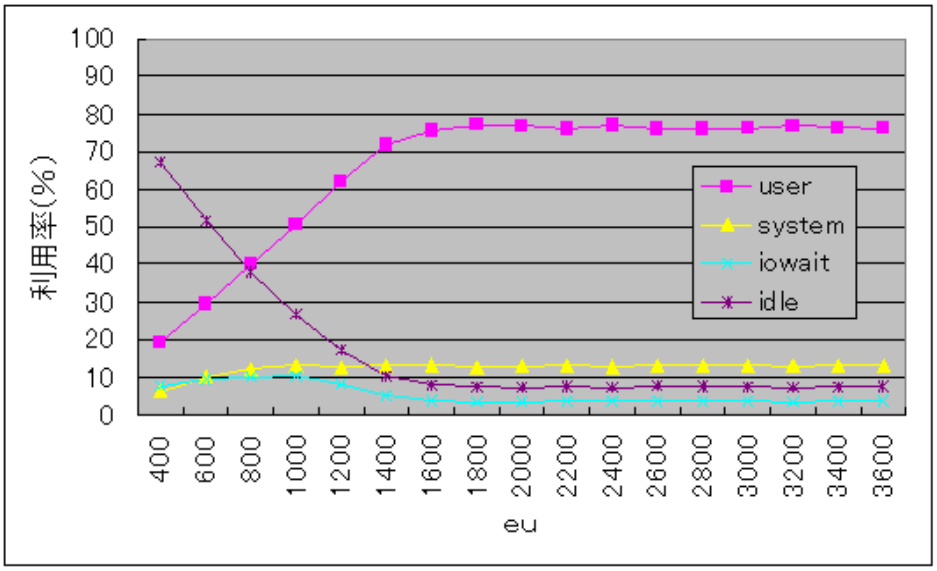

図 7.2-6　CPU 利用率（PostgreSQL 8.0.6 デフォルト環境）

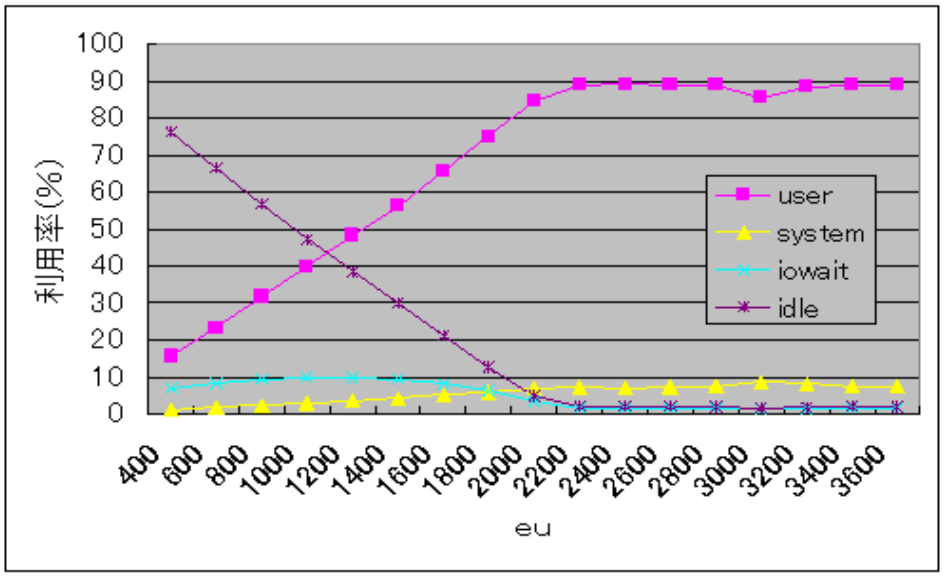

図 7.2-7　CPU 利用率（PostgreSQL 8.0.6 チューニング後環境）

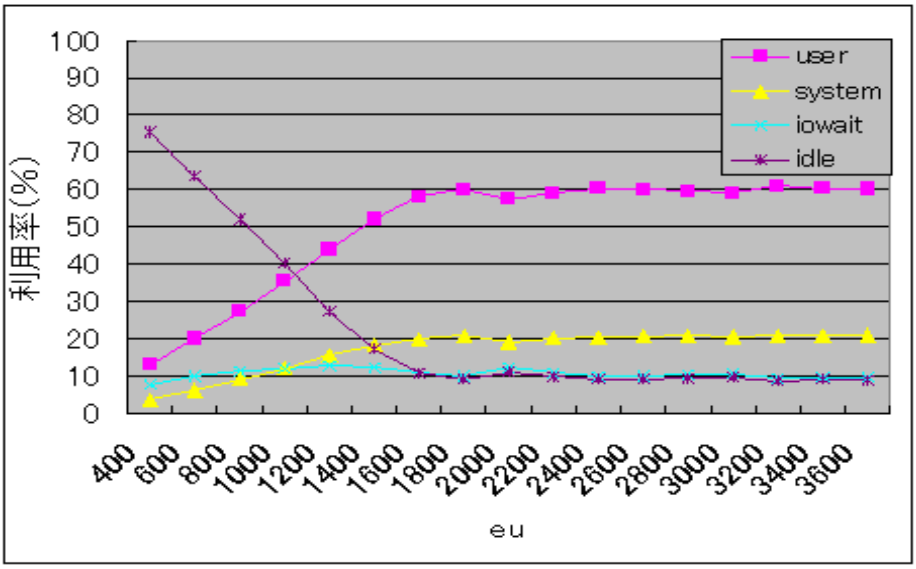

図 7.2-8　CPU 利用率（PostgreSQL 8.1.2 デフォルト環境）

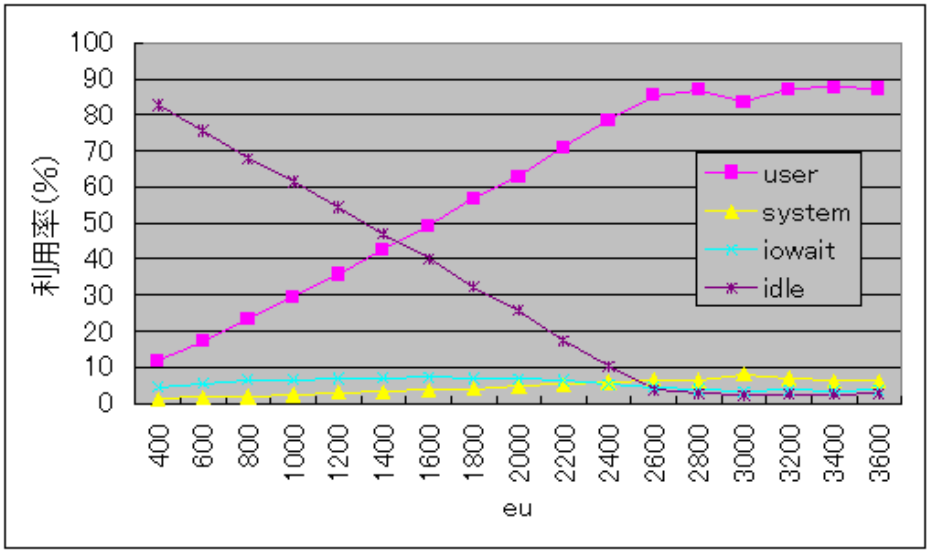

図 7.2-9　CPU 利用率（PostgreSQL 8.1.2 チューニング後環境）

7.3　考察

３つのフェーズでチューニングを実施した結果，BT/s は PostgreSQL8.0.6 で 284.1/193.8-1≒47%，PostgreSQL 8.1.2 で 354.0/216.2-1≒64%向上した。BT/s の２つのバージョン間の比率（PostgreSQL 8.1.2／PostgreSQL 8.0.6）は，デフォルト環境で 216.2/193.8≒1.12，チューニング後環境で 354.0/284.1≒1.25 となり，チューニングによる BT/s の伸び率は，PostgreSQL 8.1.2 のほうが高かった。

以下に，チューニング効果のあった shared_buffers，wal_sync_method，wal 格納先の変更について詳細に分析する。

7.3.1　shared_buffers

本評価ではshared_buffersが 1000 から 5000 の間でBT値が変化したが，それ以上増やしてもBT/sの大きな変化は無かった。探針測定（フェーズⅠ）時，PostgreSQL8. 1. 2 でshared_buffersを 1000，5000，100000 に指定して測定したケースのアクセス統計情報を図 7.3-1，図 7.3-2，図 7.3-3にそれぞれ示す。shared_buffersを 1000 か

ら5000に増やすことにより，ブロックリード回数が減少し，5000から100000に増やしても1000と5000の間程の差異は無いことが分かる。

このことから，shared_buffersは，アクセス頻度の高いテーブル（主にitemとauthor）がバッファに載り切ってしまうサイズを指定すれば，それ以上の値を指定してもキャッシュ効果は，あまり向上しないと言える。

relid	relname	heap_blks_read	heap_blks_hit	idx_blks_read	idx_blks_hit
16501	author	521040	11444074	48551	1374564
16527	orders	8614	27	43642	4703
16543	shopping_cart_line	108916	40634	76242	335873
16539	shopping_cart	44736	53314	46262	122689
16509	address	19816	351	71438	22620
16513	customer	119417	27924	250943	288961
16520	item	15170839	38789067	772771	7125391
16531	order_line	8617	256	30128	5577
16505	country	9430	5099	18448	9230
16535	cc_xacts	8617	25	26271	7871

図 7.3-1　PostgreSQL 8.1.2　eu=2400，shared_buffers=1000 の統計情報(table_info89.out)

relid	relname	heap_blks_read	heap_blks_hit	idx_blks_read	idx_blks_hit
16501	author	132	13871769	13	1651094
16527	orders	5912	4679	8140	48783
16543	shopping_cart_line	1997	187990	571	489271
16539	shopping_cart	3731	112066	1103	195382
16509	address	17279	6760	19048	92174
16513	customer	52059	121096	48696	577333
16520	item	1386	65425952	3957	9181751
16531	order_line	7063	3547	2389	41981
16505	country	1	17331	2	32960
16535	cc_xacts	6017	4604	4342	37649

図 7.3-2　PostgreSQL 8.1.2　eu=2400，shared_buffers=5000 の統計情報(table_info89.out)

relid	relname	heap_blks_read	heap_blks_hit	idx_blks_read	idx_blks_hit
16501	author	132	14335879	13	1709681
16527	orders	1866	9119	1839	57849
16543	shopping_cart_line	525	193330	156	504682
16539	shopping_cart	648	118379	154	202143
16509	address	5970	18918	6163	108710
16513	customer	11021	168060	5174	641537
16520	item	1407	68644709	433	9497394
16531	order_line	3541	7121	1136	44765
16505	country	1	17898	2	34130
16535	cc_xacts	1865	9139	1704	41771

図 7.3-3　PostgreSQL 8.1.2　eu=2400，shared_buffers=100000 の統計情報(table_info89.out)

shared_buffersの最適値は，データベースの規模により変化すると考えられる。DBT1では，頻繁にアクセスされるitem, authorテーブルおよびインデクスが載り切る程度のバッファを確保することが好ましいと考える。ただし，PostgreSQL 8.0.6の探針測定の結果では，shared_buffersを50000，100000に指定したのケースが，10000に指定したのケースに比べわずかながらBT／sが劣化したこともあり，無条件で大きな値を指定するのではなく，データベース規模と実測値を考慮して最適値を求めることが必要であると考える。

7.3.2 wal_sync_method

探針測定でwal_sync_methodをデフォルト値（fdatasync）およびopen_syncに変更して測定した場合のCPU利用率(DBT1 結果ファイルのall.cpu.txt中に示されている平均値)を表 7.3-1に，iostatの結果（DBT1 結果ファイルのio.txt）を，表 7.3-12，表7.3-3 に示す。CPU利用率の結果から，PostgreSQL 8.1.2 はiowaitが解消されているが，PostgreSQL 8.0.6 ではデフォルトの状態でもiowaitがPostgreSQL 8.1.2 ほど大きな値ではなかったので，wal_sync_methodの指定値を変更してもBT/sには効果が現れなかったと考える。

表 7.3-1CPU 利用率

	チューニング観点（設定項目）	user	system	iowait	idle
PostgreSQL 8.0.6	デフォルト	80.31	11.05	3.38	5.26
	open_sync	79.49	11.93	3.25	5.32
PostgreSQL 8.1.2	デフォルト	57.41	16.09	13.72	12.77
	open_sync	71.03	21.53	4.88	2.56

表 7.3-2 PostgreSQL 8.0.6 の iostat の結果集計値（eu=2000）

	デバイス	tps	Blk_read/s	Blk_wrtn/s	Blk_read	Blk_wrtn
デフォルト(fdatasync)	Sda	0.66	0.10	19.36	1.01	193.68
	Sdb	165.41	246.06	2984.28	2461.65	29853.65
	Sdc	3.32	0.02	87.80	0.20	878.32
open_sync	Sda	0.64	0.06	19.39	0.61	194.02
	Sdb	142.29	245.89	2393.73	2459.76	23945.95
	sdc	3.43	0.03	86.75	0.27	867.76

表 7.3-3 PostgreSQL 8.1.2 の iostat の結果集計値（eu=2000）

	デバイス	tps	Blk_read/s	Blk_wrtn/s	Blk_read	Blk_wrtn
デフォルト(fdatasync)	sda	0.71	0.10	21.76	1.01	217.68
	sdb	176.14	260.46	3055.44	2605.04	30560.47
	sdc	3.43	0.03	100.08	0.27	1001.01
open_sync	sda	0.62	0.19	20.09	1.95	201.41
	sdb	150.52	281.71	2931.79	2822.45	29379.70
	sdc	3.50	0.03	106.64	0.34	1068.77

注）iostat の１回目の出力結果を除き，残りを平均した値である。スループット(BT／s)が異なるため，全デバイスについての合計値は，両ケースで正確には同値とはならない。

7.3.3 wal ログ格納ディスク変更

探針測定でデフォルト環境とwalログの格納先を変更した場合のCPU利用率(DBT1 結果ファイルのall.cpu.txt中に示されている平均値)を表 7.3-14 に，iostatの結果（DBT1 結果ファイルのio.txt）を，表 7.3-15，表 7.3-6に示す。iostatの結果で，「デフォルト」と「walログの格納先変更」を比較すると，データベースと更新ログの格納先を分けることにより，以下のようにI/O負荷（書き込みの負荷）がsdbからsdcに分散されていることが分かる。PostgreSQL 8.0.6 では，デフォルト状態でもCPU利用率が 90%以上と高く，iowaitは大きな値では無かったため，walログの格納先を変更してもBT/sには効果が現れなかったと考える。

表 7.3-4 CPU 利用率

	チューニング観点（設定項目）	user	system	iowait	idle
PostgreSQL 8.0.6	デフォルト	80.31	11.05	3.38	5.26
	wal ログ格納先変更	79.45	11.79	1.73	7.03
PostgreSQL 8.1.2	デフォルト	57.41	16.09	13.72	12.77
	wal ログ格納先変更	73.23	22.42	1.69	2.66

表 7.3-5 PostgreSQL 8.0.6 の iostat の結果集計値（eu=2000）

	デバイス	tps	Blk_read/s	Blk_wrtn/s	Blk_read	Blk_wrtn
デフォルト	sda	0.66	0.10	19.36	1.01	193.68
	sdb	165.41	246.06	2984.28	2461.65	29853.65
	sdc	3.32	0.02	87.80	0.20	878.32
wal ログ格納先変更	sda	0.74	0.39	23.97	3.90	239.73
	sdb	99.69	233.48	1719.75	2335.26	17201.28
	sdc	24.86	0.07	802.41	0.67	8025.88

表 7.3-6 PostgreSQL 8.1.2 の iostat の結果集計値（eu=2000）

	デバイス	tps	Blk_read/s	Blk_wrtn/s	Blk_read	Blk_wrtn
デフォルト	sda	0.57	0.66	20.72	6.59	207.33
	sdb	174.32	236.05	3071.44	2362.29	30738.96
	sdc	3.67	0.03	103.58	0.27	1036.64
wal ログ格納先変更	sda	0.72	0.52	25.93	5.18	259.83
	sdb	119.30	281.90	2131.68	2823.60	21356.37
	sdc	34.03	0.05	961.82	0.47	9635.43

注）iostat の１回目の出力結果を除き，残りを平均した値である。スループット(BT／s)が異なるため，全デバイスについての合計値は，両ケースで正確には同値とならない。

7.4 まとめ

３つのフェーズでチューニングした設定項目値と，その結果を表 5.5-17. 4-1 に示す。

	項番	チューニング観点（設定項目）	改善内容（設定項目値）	
			PostgreSQL 8.0.6	PostgreSQL 8.1.2
SQL 実行プラン分析によるチューニング内容	1	random_page_cost	2	4（デフォルト）
	2	enable_mergejoin	on（デフォルト）	Off
	3	enable_sort	on（デフォルト）	Off
	4	インデクス定義改善	デフォルトのまま(改善点なし)	条件判定とソートを同時に行えるインデクスを追加
探針測定によるチューニング内容	1	shared_buffers	10000	100000
	2	wal_sync_method	open_sync	open_sync
	3	wal ログ格納先	DB 本体と異なるディスク	DB 本体と異なるディスク
	4	max_connections	120	120
	5	max_fsm_pages	50000	50000
BT／s (eu＝3200)		デフォルト	193.8	216.2
		総合チューニング実施後	284.1	354.0
		向上率	47%	64%

図 7.4-1　デフォルト環境と総合チューニング後環境の比較

8 VACUUM に関する評価

8.1 目的

PostgreSQL では UPDATE／DELETE 文を実行するごとに, データベースに無効領域が蓄積されていくため, 定期的に VACUUM コマンドを実行して無効領域を解放する必要がある。VACUUM に要する時間を評価することを目的として，DBT-1 を利用して以下の測定を行う。

- DBT-1 の実行時間を 1～10 時間の間で変化させ，BT／s への影響を検証する。
- ベンチマーク終了後に VACUUM コマンドを実行し，コマンド単体性能を評価する。

測定ケースを表 8.1-1にまとめる。

表 8.1-1　測定ケース（VACUUM の評価）

測定ケース番号	run_duration	vacuum の実行
CASE1	4100 （1.14hr）	なし
CASE2	12600（3.5hr）	なし
CASE3	25200（7hr）	なし
CASE4	36000（10hr）	なし

※1　eu は差の見えやすいポイントとして 3600 をとる。

また，CASE2 の終了後に全テーブルを VACUUM するケース，item テーブルのみを VACUUM するケースを実行し，単体性能を測定する。

8.2 測定結果

測定結果を表 8.2-1 に示す。

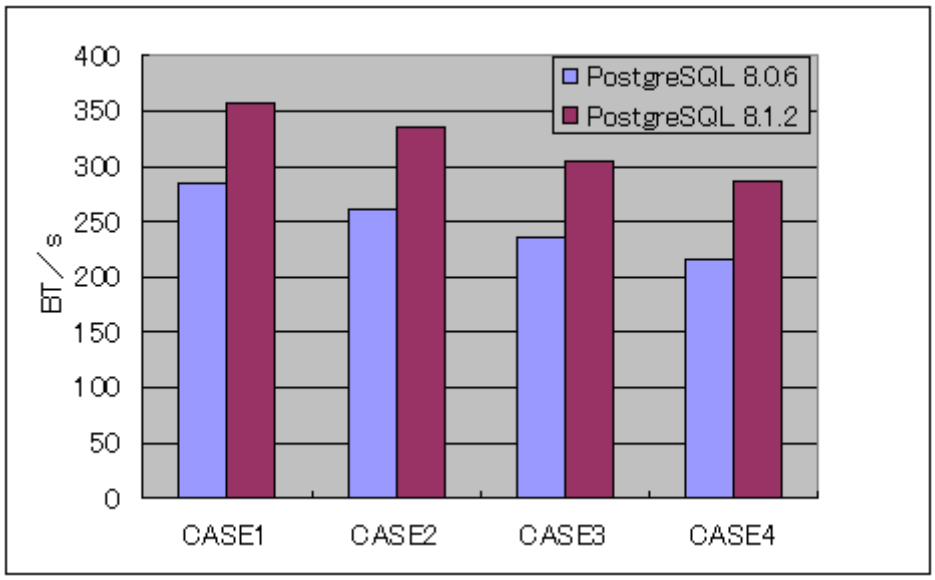

図 8.2-1　DBT-1 実行時間，VACUUM 有無と BT／s の関係

表 8.2-1　DBT-1 実行時間，VACUUM 有無と BT／s の関係（実測値）

ケース番号	CASE1	CASE2	CASE3	CASE4
PostgreSQL 8.0.6	285	260.8	236.3	215.5
PostgreSQL 8.1.2	357.7	334.6	305.3	286

表 8.2-2　VACUUM コマンドの実行時間

実行タイミング	VACUUM 対象	実行時間（秒） (8.0.6)	(8.1.2)
CASE2 の終了後に実行	item テーブルのみ	38.801	2.445
同上	全テーブル	861.79	382.127

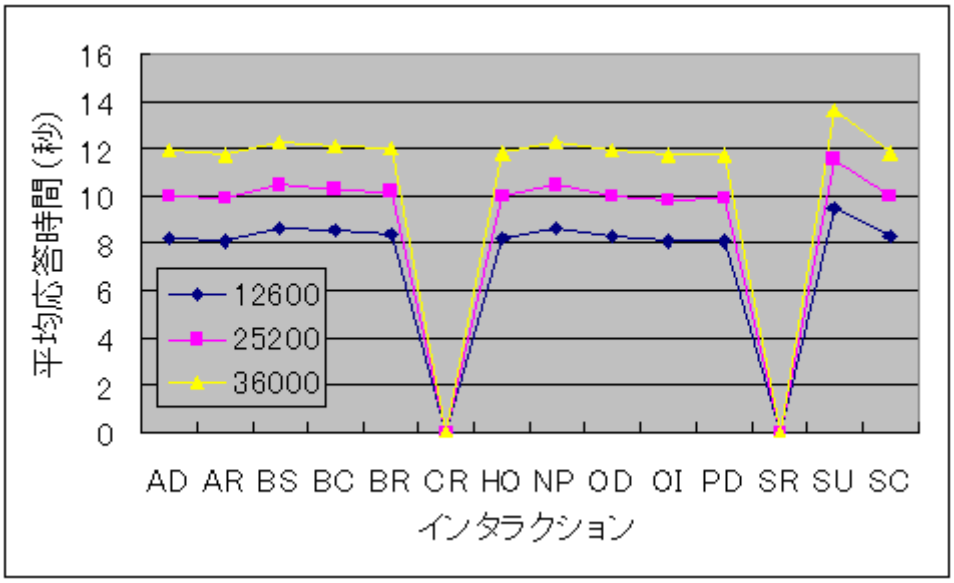

図 8.2-2　DBT-1 の実行時間と平均応答時間との関係（PostgreSQL 8.0.6）

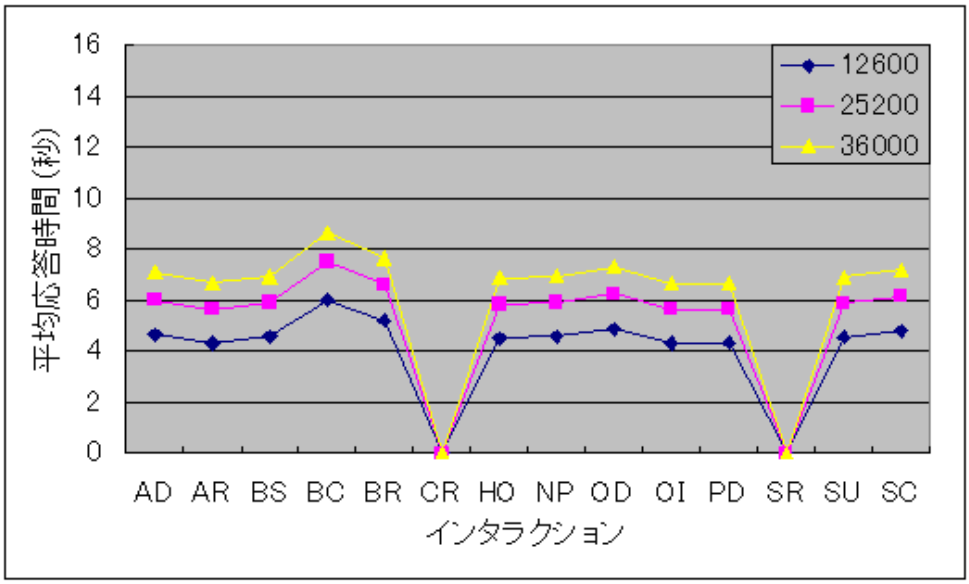

図 8.2-3　DBT-1 の実行時間と平均応答時間との関係（PostgreSQL 8.1.2）

CASE1 からCASE4 までの測定結果から，DBT1 の実行時間にほぼ反比例してBT/ｓが低下していることが分かった。また，インタラクションの平均応答時間は時間の経過に比例して長くなる傾向があった。この原因は，DBT1 を実行させることにより，データベースの無効領域が拡大し，それに伴いSQLコマンドの実行コストも増加していくことによると考える。

そこで，PostgreSQLのソースに同梱されているツールであるpgstattupleを使用し，DBT1 実行後のテーブルごとの無効領域を調査したところ，customer，item，shopping_cart，shopping_cart_lineの 4 テーブルに集中していることが分かった。これら 4 テーブルについて，無効領域の増加傾向を図 8.2-4，図 8.2-5に示す。

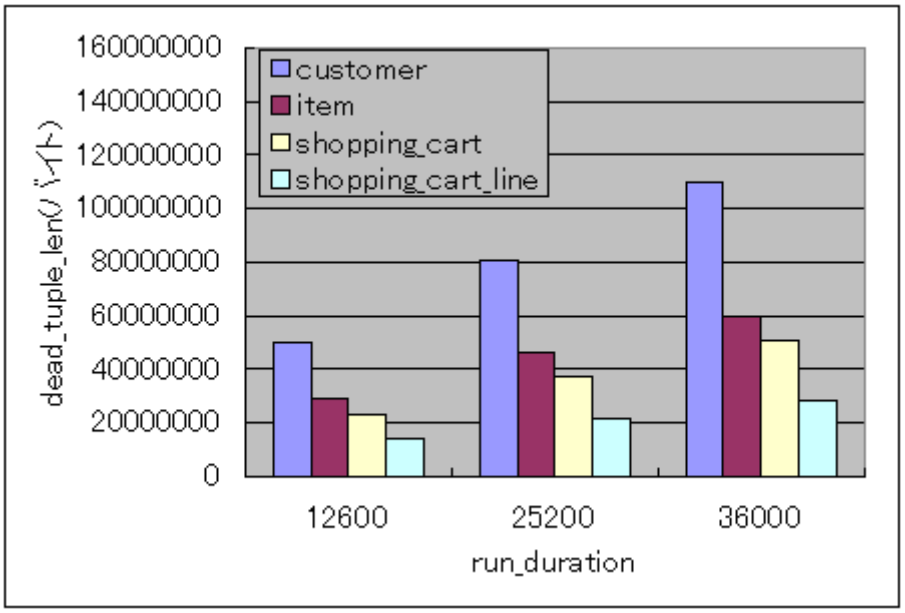

図 8.2-4　テーブル無効領域（PostgreSQL 8.0.6）

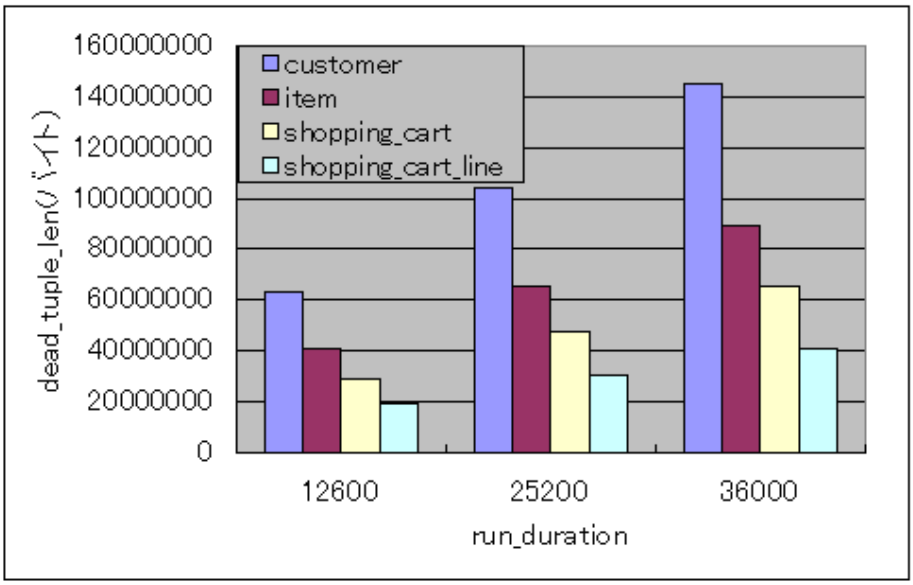

図 8.2-5　テーブル無効領域（PostgreSQL 8.1.2）

8.3 考察

測定結果から，VACUUM コマンドの実行時間は，全テーブル対象の場合で PostgreSQL 8.0.6 で 861 秒，PostgreSQL 8.1.2 で 382 秒となった。DBT1 を特定業務に見立てると，24 時間の連続運転の必要がなければ，計画停止後に VACUUM を行えばよい。

しかし，24 時間連続運転を必要とする場合や，停止時間が短時間に限定される場合，業務実行中に VACUUM を実行する方法もその対応策と考え，実現性を DBT1 を用いて検証した。

業務実行中に VACUUM を実行する検証ケースを表 8.3-1 に示す。全テーブルを VACUUM するケース，無効領域の発生しているテーブルの中で最もアクセス頻度の高い item テーブルのみを VACUUM するケース，無効領域が最も大きい customer テーブルのみを VACUUM するケースを追加測定し，BT/s を比較した。

表 8.3-1 業務実行中にVACUUMを実行する検証ケース（VACUUMの評価）

測定ケース番号	run_duration	vacuumの実行
CASE5	36000（10hr）	itemテーブルのみを3.5hr経過後, 7h経過後にそれぞれvacuum。
CASE6	36000（10hr）	全テーブルを3.5hr経過後，7h経過後にそれぞれvacuum。
CASE7	36000（10hr）	customer テーブルのみを 3.5hr 経過後，7h 経過後にそれぞれvacuum。

※ VACUUMの実行間隔は，BT／s低下の許容範囲をrd＝4100の測定結果の10%程度と仮定し，3.5hr間隔（3.5hr，7hrの2ポイント）とした。

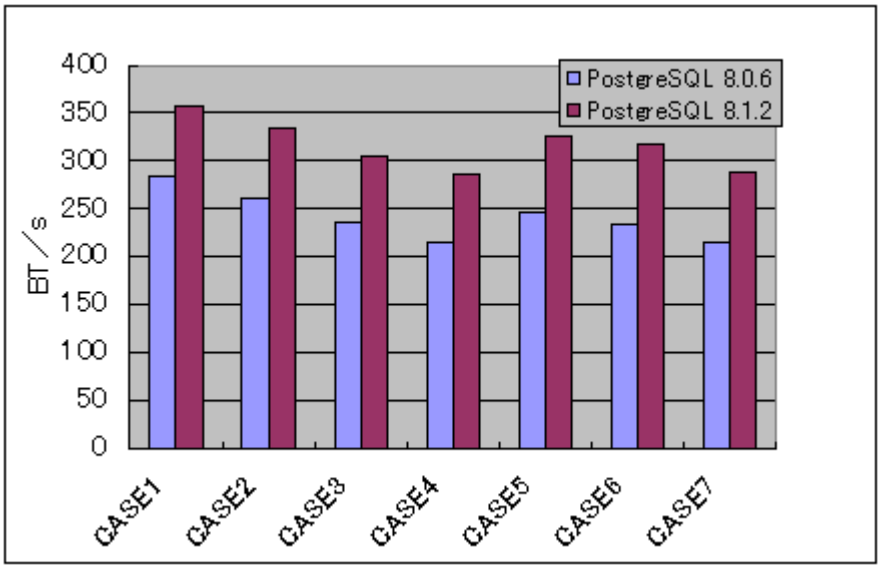

図 8.3-1 DBT-1実行時間，vacuum有無とBT／sの関係（実測値。測定ケース追加）

表 8.3-2 VACUUMコマンドの実行時間（測定ケース追加）

実行タイミング	VACUUM対象	実行時間（秒） (8.0.6)	(8.1.2)
CASE2の終了後に実行	itemテーブルのみ	38.801	2.445
同上	全テーブル	861.79	382.127
CASE5において3.5hr経過後に実行	itemテーブルのみ	58.407	6.554
CASE6において3.5hr経過後に実行	全テーブル	3816.908	1234.309
CASE7において3.5hr経過後に実行	customerテーブルのみ	1256.514	428.081

CASE5からCASE7の結果を見ると，PostgreSQL 8.0.6についてはCASE5（itemテーブルのみVACUUM）はCASE2（3.5hr実行）と同等となり，BT/sの劣化を抑えることができたが，CASE6(全テーブルのVACUUM)はCASE3（7hr実行）と同等まで劣化した。またCASE7(customerテーブルのみのVACUUM)はCASE4(10h実行)と同等となり，customerのみのVACUUMの効果がBT/sには現れないという結果になった。原因としてはcustomerテーブルはitemテーブルほどアクセス頻度が高くないこと，テーブル規模がitemテーブルに比べて大きくVACUUM自体にコストがかかることが考えられる。PostgreSQL 8.1.2では，CASE5，CASE 6とも，BT/sはCASE2（3.5hr実行）より僅かに劣化する程度で，殆ど同等であったが，CASE7はCASE4と同等となり，PostgreSQL 8.0.6と同じくVACUUMの効果は見られなかった。
両バージョンを通じて，アクセス頻度が高いテーブル(itemテーブル)のVACUUMはBT/sの劣化防止に有効であると考える。

VACUUM の実行時間を DBT1 実行中に実行したケースと，DBT 1 終了後に実行したケースを比べると，3～4 倍の差があるが，全テーブルの VACUUM は，PostgreSQL 8.0.6 で 60 分，PostgreSQL 8.1.2 で 20 分程度要しているが，item テーブル単独の VACUUM は，PostgreSQL 8.0.6 で 40 秒，PostgreSQL 8.1.2 では 10 秒以内に終了しており，システムに掛かる負荷，ユーザ応答時間に対する影響は小さいと判断する。

以上のことから、対象テーブルを絞り，業務中に VACUUM することで、BT/s の劣化防止に効果を上げることができると考える。

9 HyperThreading の有効性

9.1 目的

CPU リソースを有効に活用する手段に、HyperHtreading を有効指定し，システム運用する方法がある。これまで，探針測定（フェーズⅠ），SQL 実行プランの分析に基づくチューニング（フェーズⅡ），総合チューニング（フェーズⅢ）と評価を実施してきた。この状態で，さらに性能を伸ばす方法として，HyperHterading が有効なのか評価する。

9.2 測定結果

測定結果を表 9.2-1 に示す。PostgreSQL 8.1.2 では HyperThreading を有効にすることで, 約 20%BT/s が向上した。また, PostgreSQL8.0.6 でも、性能劣化することはなかった。

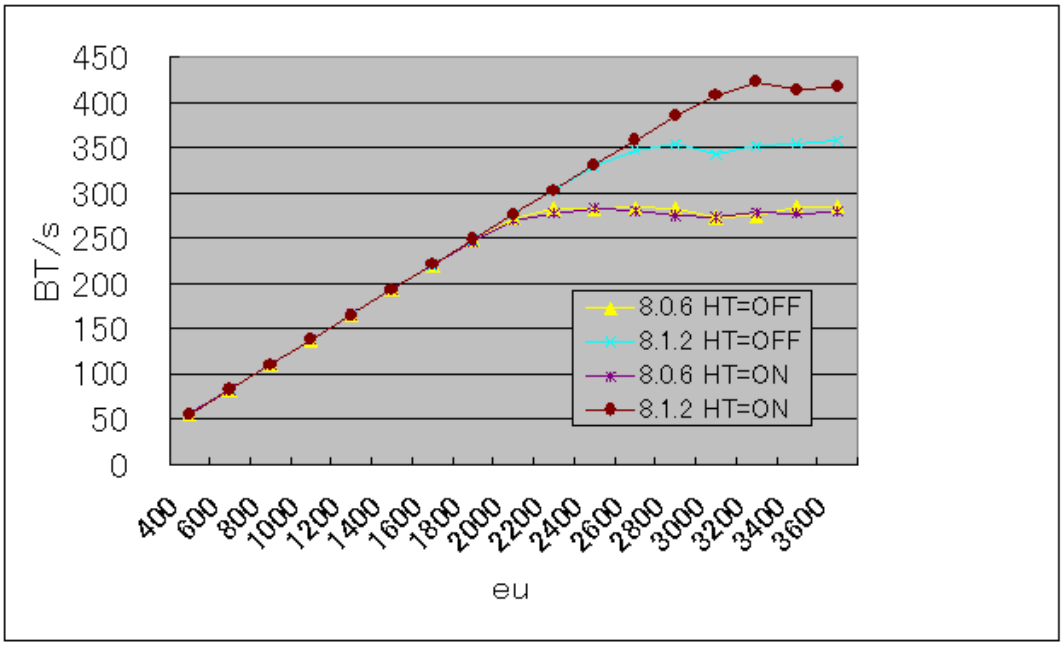

図 9-1　BT／s 推移（チューニング後環境におけるハイパースレッド(HT)有効／無効の比較）

9.3 考察

PostgreSQL8.1.2, および PostgreSQL8.0.6 を対象に DBT1 を実行した時の oprofile によって得られた当該システムで多用された関数のベスト 20 を付録.B に記載する。

PostgreSQL8.1.2 についての Oprofile の結果から，デフォルト環境では，__copy_to_user_ll のようにバッファキャッシュに対するデータのコピーが多用されていたのに対し，チューニング後環境では，それが解消されていることでより PostgreSQL の処理に CPU が割り当てられいる様子が分った。また，HeapTupleSatisfiesSnapshot 等の like 検索時に用いられる関数や，排他制御の LWLockAcquire 等は、チューニング後環境ではサンプル回数が増え，さらに HyperThreading を有効にした環境では，それらのサンプル数はさらに増えるが，１回当たりの実行時間の割合が下がっていたことで，CPU の Threading が有効に機能していることが分った。

一方, PostgreSQL8.0.6 は, デフォルト環境で HyperThreading も無効の場合, HeapTupleSatisfiesSnapshot が CPU の比率が高いことがわかる。

10 まとめ

今回の評価で分かったことを以下にまとめる。

- DBT1 に代表するように、Webの参照系を中心とするサービスにPostgreSQLを用いるような場合、データベースパラメタのshared_buffres等，表 5.5-1に示す値を調節することで、概略的なチューニングが可能である。
- DBT1 の性能値（BT/s）を，パラメタチューニング（shared_buffres，wal_sync_method，プランナ関連パラメタ），wal ログ格納先の変更，インデクス改善によりデフォルト状態から，PostgreSQL 8.0.6 で約 47%，PostgreSQL 8.1.2 で約 64%向上した。
- PostgreSQL 8.1.2 は，プランナの動作に関して，PostgreSQL 8.0.6 よりも次の点で向上していると判断する。
  ① デフォルト状態でインデクスが使用されるようになり，チューニングが容易になった。
  ② マルチカラムインデクスが効率的に使用されるようになった。
- 計画停止が困難なシステムでは，オンライン中の VACUUM であっても，対象テーブルをアクセス頻度，更新頻度などを考慮することで，システムに大きな負荷をかけずに，業務中に VACUUM しスループット性能を維持させる方法も考えられる。
- オンライン中の VACUUM であっても，対象テーブルをアクセス頻度，更新頻度，データベースの規模に応じて限定すればシステムに多大な負荷をかけなくてもスループットの低下を抑えることができた。
- ハイパースレッドを有効にすることにより PostgreSQL 8.1.2 については 20%程度の BT/s の向上させることが可能であり、前バージョンから、CPU スレッドスケーラビリティが向上していると判断する。

11　付録.A

OSDL/DBT-1 の SQL の全 SQL

（アミカケは、単純 SQL のため，分析から除外した SQL）

Q#	INCLUDE ファイル名	DEFINE 名称	SQL テキスト	SQL 実行プラン分析			
				Seq scan	Sort	ジョイン方式	改善点
1	addToSC.h	STMT_addToSC_selsumSCL	SELECT sum(scl_qty) FROM shopping_cart_line WHERE scl_sc_id=%lld HAVING sum(scl_qty) IS NOT NULL;	無	無	該当せず	
2		STMT_addToSC_selSCL	SELECT scl_qty FROM shopping_cart_line WHERE scl_sc_id=%lld AND scl_i_id =%lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
3		STMT_addToSC_updSCL	UPDATE shopping_cart_line SET scl_qty=%d WHERE scl_sc_id=%lld AND scl_i_id =%lld	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
4		STMT_addToSC_sellITM1	SELECT i_cost, i_srp, i_title, i_backing FROM item WHERE i_id=%lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
		STMT_addToSC_insSCL	INSERT INTO shopping_cart_line VALUES(%lld, %lld, 1, %lf, %lf, '%s', '%s');				
5		STMT_addToSC_selSCLC	SELECT count(scl_sc_id) FROM shopping_cart_line WHERE scl_sc_id=%lld;	無	無	該当せず	
6		STMT_addToSC_sellITM2	SELECT t1.i_cost, t1.i_srp, t1.i_title, t1.i_backing, t2.i_related1 FROM item t1, item t2 WHERE t1.i_id=t2.i_related1 AND t2.i_id=%lld;	無	無	Nested Loop	特に無し。Nested Loop の内外表とも，プライマリキーのインデクスで高々１件に絞り込まれる。
7	createSC.h	STMT_createSC_selCUST	SELECT c_fname, c_lname, c_discount FROM customer WHERE c_id=%lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
		STMT_createSC_insSC1	INSERT INTO shopping_cart(sc_id, sc_c_id, sc_date, sc_c_fname, sc_c_lname, sc_c_discount) VALUES (%lld, %lld, NOW(), '%s', '%s', %lf);				分析対象外
		STMT_createSC_insSC2	INSERT INTO shopping_cart(sc_id, sc_date) VALUES (%lld, NOW());				分析対象外
		STMT_createSC_sellnsVal	select nextval('scid');				分析対象外
	DigSyl.h						

8		STMT_getCustInfo	SELECT c_id,c_passwd,c_fname,c_lname, c_phone,c_email,c_data,c_birthdate ,c_discount,addr_street1,addr_stre et2,addr_city,addr_state,addr_zip, co_name FROM customer, address, country WHERE c_uname = '%s' and addr_id = c_addr_id and addr_co_id = co_id;	無	無	Nested Loop	特に無し。3 表の Nested Loop となっている。内側の 2 表（address, country）はプライマリキーのインデクスで，各々一件に絞り込まれる。最外側表(customer)の検索も，ユニーク値を持つ列（c_uname）のインデクスで絞り込まれている。
9	getCustInfo.h	STMT_updateCustInfo	UPDATE customer SET c_login=now(),c_expiration=(now()+'02:00:00') WHERE c_id = %lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
10	getOrderItems.h	STMT_getOrderItems	SELECT ol_i_id,i_title,i_publisher,i_cost ,ol_qty,ol_discount,ol_comments FROM item, order_line WHERE ol_o_id = %lld AND ol_i_id = i_id;	無	無	Nested Loop	特に無し。Nested Loop の外側表（order_line）は検索のキーとなっている ol_o_id の重複度から見て平均 3 件程度，内側表はプライマリキーインデクスにより高々一件に絞り込まれる。
11	getPromoImages.h	STMT_getPromoImages	SELECT i_id,i_thumbnail FROM item WHERE i_id = (SELECT i_related1 FROM item WHERE i_id = %lld) OR i_id = (SELECT i_related2 FROM item WHERE i_id = %lld) OR i_id = (SELECT i_related3 FROM item WHERE i_id = %lld) OR i_id = (SELECT i_related4 FROM item WHERE i_id = %lld) OR i_id = (SELECT i_related5 FROM item WHERE i_id = %lld);	無	無	該当せず	特に無し。PG8.1 では，Bitmap Index scan／BitmapOr が適用されている。
12	getSCDetail.h	STMT_getSCDetail	SELECT scl_i_id,scl_title,scl_cost,scl_srp,scl_backing,scl_qty FROM shopping_cart_line WHERE scl_sc_id=%lld;	無	無	該当せず	
13	getSCSubtotal.h	STMT_getSCSubtotal	SELECT SUM(scl_cost*scl_qty) FROM shopping_cart_line WHERE scl_sc_id=%lld;	無	無	該当せず	
	initOrderItems.h						
	initSCItems.h						
14	InsertCust.h	STMT_InsertCust_getCountry	SELECT co_id FROM country WHERE co_name='%s';	有 (1)	無	該当せず	特に無し。country テーブルの統計情報(pg_class の relpages 列の値）を見るとテーブル容量が１ページとなっており，インデクスを使用する必要はない。

15		STMT_InsertCust_getAddr	SELECT addr_id FROM address WHERE addr_co_id = %d AND addr_zip = '%s' AND addr_state = '%s' AND addr_city = '%s' AND addr_street1 = '%s' AND addr_street2 = '%s';	無	無	該当せず	特に無し。インデクスでユニークに絞り込まれる。
		STMT_InsertCust_insAddr	INSERT INTO address (addr_id,addr_street1,addr_street2,addr_city,addr_state,addr_zip,addr_co_id) VALUES(%d,'%s','%s','%s','%s','%s',%d);				分析対象外
		STMT_InsertCust_insAddr_selInsVal	select nextval('addrid');				分析対象外
16		STMT_InsertCust_LastCID	select c_id from customer order by c_id desc limit 1;	無	無	該当せず	特に無し。customer テーブルがインデクス経由で１行アクセスされる。
		STMT_InsertCust_insCust	INSERT INTO customer(c_id,c_uname,c_passwd,c_fname,c_lname,c_addr_id,c_phone,c_email,c_since,c_last_visit,c_login,c_expiration,c_discount,c_balance,c_ytd_pmt,c_birthdate,c_data) VALUES(%lld,'%s','%s','%s','%s',%d,'%s','%s',NOW(),NOW(),TIMESTAMP'NOW',(NOW()+'02:00:00'),%f,0.00,0.00,'%s','%s');				分析対象外
		STMT_InsertCust_selInsVal	select nextval('custid');				分析対象外
	interaction.h						
17		STMT_ADMIN_CONFIRM_UPD	UPDATE item set i_image=%lld,i_thumbnail=%lld,i_cost=%lf,i_pub_date=Date(Now()) WHERE i_id = %lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
18	interaction_admin_confirm.h	STMT_ADMIN_CONFIRM_RET	SELECT i_image,i_thumbnail,i_cost,i_title,a_fname,a_lname,i_subject,i_desc,i_srp,i_backing,i_page,i_publisher,i_pub_date,i_dimensions,i_isbn From item, author WHERE i_id = %lld and i_a_id = a_id;	無	無	Nested Loop	特に無し。Nested Loop の内外表（author, item）ともプライマリキーのインデクスで一件に絞り込まれる。
19	interaction_admin_request.h	STMT_ADMIN_REQUEST	SELECT i_srp, i_cost, i_title, i_image, i_thumbnail, a_fname, a_lname FROM item, author WHERE i_id = %lld AND i_a_id = a_id;	無	無	Nested Loop	特に無し。Nested Loop の内外表（author, item）ともプライマリキーのインデクスで一件に絞り込まれる。

20	interaction_best_sellers.h	STMT_BEST_SELLERS	SELECT i_id, i_title, a_fname, a_lname FROM item, author WHERE i_subject = '%s' AND i_a_id = a_id ORDER BY i_pub_date DESC, i_title ASC;	有(1) (8.0) 無 (8.1)	有(2)	Merge Join	<8.0> item テーブルの検索にインデクスが使用されない。 <8.0, 8.1> ソートが発生している。
21	interaction_buy_confirm.h	STMT_BUYCONF_selSC	SELECT sc_sub_total, sc_tax, sc_ship_cost, sc_total FROM shopping_cart WHERE sc_id = %lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
22		STMT_BUYCONF_selCS	SELECT c_fname, c_lname, c_discount, c_addr_id FROM customer WHERE c_id = %lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
23		STMT_BUYCONF_selADRCNT	SELECT addr_id, co_id FROM address, country WHERE co_name = '%s' AND addr_co_id = co_id AND addr_zip = '%s' AND addr_state = '%s' AND addr_city = '%s' AND addr_street1 = '%s' AND addr_street2 = '%s';	有(1)	無	Nested Loop	特に無し。Nested Loop の内側表（ｃｏｕｎｔｒｙ）がSeq scanになっているが，容量が1ページと小さく，外側表 (address) もインデクスでユニークに絞り込まれることから問題無しと考える。
		STMT_BUYCONF_getCountry	SELECT co_id FROM country WHERE co_name= '%s';	問題無し	問題無し	該当せず	STMT_InsertCust_getCountry と同一SQL。
		STMT_BUYCONF_insADR	INSERT INTO address(addr_id , addr_street1,addr_street2,addr_city,addr_state,addr_zip,addr_co_id) VALUES(%lld , '%s', '%s', '%s','%s', '%s', %d);				分析対象外
		STMT_BUYCONF_selADRID	select nextval('addrid');				分析対象外
24		STMT_BUYCONF_selADR	SELECT addr_co_id FROM address WHERE addr_id = %lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
		STMT_BUYCONF_insODR	INSERT INTO orders VALUES(%lld, %lld, TIMESTAMP'NOW',%lf,%lf,%lf,'SHIP', TIMESTAMP'NOW'+¥'%d days¥',%lld,%lld,'PENDING');				分析対象外
25		STMT_BUYCONF_selSCL	SELECT scl_i_id, scl_cost, scl_qty FROM shopping_cart_line WHERE scl_sc_id = %lld;	無	無	該当せず	
		STMT_BUYCONF_insODRL	INSERT INTO order_line VALUES(%d, %lld, %lld, %d, %lf, '%s');				分析対象外
26		STMT_BUYCONF_selITM	SELECT i_stock FROM item WHERE i_id = %lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。

27		STMT_BUYCONF_updITM	UPDATE item SET i_stock = %d WHERE i_id = %lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
		STMT_BUYCONF_insXACT	INSERT INTO cc_xacts VALUES (%lld, '%s', '%s', '%s', '%s', '%s',%lf,NOW(), %d);				分析対象外
		STMT_BUYCONF_selInsVal	select nextval('scid');				分析対象外
28	interaction_buy_request.h	STMT_BUYREQ_updtsc	UPDATE shopping_cart SET sc_c_fname = '%s',sc_c_lname = '%s',sc_c_discount = %f,sc_c_id = %lld WHERE sc_id = %lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
29	interaction_home.h	STMT_HOME	SELECT c_fname, c_lname FROM customer WHERE c_id=%lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
	interaction_new_products.h	STMT_NEW_PRODUCTS	SELECT i_id, i_title, a_fname, a_lname FROM item, author where i_subject='%s' and i_a_id=a_id ORDER BY i_pub_date DESC, i_title ASC;				STMT_BEST_SELLERS と同一 SQL。
30		STMT_ORDER_DISPLAY_selODRInfo	SELECT c_fname, c_lname, c_phone, c_email,o_id, o_date, o_sub_total, o_tax,o_total, o_ship_type, o_ship_date,o_status, o_bill_addr_id, o_ship_addr_id,cx_type, cx_auth_id FROM customer, orders, cc_xacts WHERE c_uname = '%s' AND c_passwd = '%s' AND o_c_id = c_id AND cx_o_id = o_id ORDER BY o_date DESC;	無	有 (1)	Nested Loop	3 表の Nested Loop を行ない，結果をソートする実行プランとなっている。各表はインデクスで十分絞り込まれるため（customer 平均１行，ｏｒｄｅｒｓ平均２行，ｃｃ_xacts1 行），ジョイン結果は平均２行程度と見積もることができ，ソートコストも問題視するほどでは無いと思われる。
31	interaction_order_display.h	STMT_ORDER_DISPLAY_selADRCNT	SELECT addr_street1, addr_street2, addr_city,addr_state, addr_zip, co_name FROM address, country WHERE addr_id = %lld AND addr_co_id = co_id;	無	無	Nested Loop	特に無し。各表（country，address）は，プライマリキーのインデクスで１件に絞り込まれている。
32	interaction_order_inquiry.h	STMT_ORDER_INQUIRY	SELECT c_uname FROM customer WHERE c_id= %lld;	無	無	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
33	interaction_product_detail.h	STMT_PRODUCT_DETAIL	SELECT i_title, a_fname, a_lname, Cast(i_pub_date as char), i_publisher,i_subject, i_desc, i_image, i_cost, i_srp, Cast(i_avail as char),i_isbn, i_page, i_backing, i_dimensions FROM item, author WHERE i_id = %lld AND i_a_id = a_id;	無	無	Nested Loop	特に無し。各表 (author，item) は，プライマリキーのインデクスで一件に絞り込まれる。

	interaction_search_request.h						
34	interaction_search_results.h	STMT_SEARCH_RESULTS_AUTHOR	SELECT i_id, i_title, a_fname, a_lname FROM item, author WHERE i_a_id=a_id AND a_lname LIKE '%%%s%%' ORDER BY i_title ASC;	有(1)	有(1)	Nested Loop	<8.0, 8.1> LIKE 述語の部分一致検索を使用しており，author 表の検索にインデクスが使用されない。 <8.0, 8.1> ソートが発生している。
35		STMT_SEARCH_RESULTS_SUBJECT	SELECT i_id, i_title, a_fname, a_lname FROM item, author WHERE i_subject = '%s' AND i_a_id = a_id ORDER BY i_title ASC;	有(1)（8.0) 無(8.1)	有(2)	Merge Join	<8.0> item テーブルの検索にインデクスが使用されない。 <8.0, 8.1> ソートが発生している。
36		STMT_SEARCH_RESULTS_TITLE	SELECT i_id, i_title, a_fname, a_lname FROM item, author WHERE i_title LIKE '%%%s%%' AND i_a_id = a_id ORDER BY i_title ASC;	有(1)	有(1)	Nested Loop	<8.0, 8.1>LIKE 述語の部分一致検索を使用しているため，item テーブルの検索にインデクスが使用されない。<8.0, 8.1>ソートが発生している。
37	interaction_shopping_cart.h	STMT_SC_upd	UPDATE shopping_cart SET sc_sub_total=%lf, sc_date=NOW() WHERE sc_id=%lld;	無	○	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
38	refreshSC.h	STMT_refreshSC_del	DELETE FROM shopping_cart_line WHERE scl_sc_id=%lld AND scl_i_id=%lld	無	○	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
		STMT_refreshSC_upd	UPDATE shopping_cart_line SET scl_qty=%d WHERE scl_sc_id=%lld AND scl_i_id =%lld;				STMT_addToSC_updSCL と同一SQL。
39	updateSC.h	STMT_updateSC_sel	SELECT scl_i_id, scl_qty, i_cost FROM shopping_cart_line,item WHERE i_id=scl_i_id AND scl_sc_id=%lld;	無	無	Nested Loop	Nested Loop の内外表とも，インデクスで 1～2 件に絞り込まれる。
40		STMT_updateSC_updscl	UPDATE shopping_cart_line SET scl_cost=%lf WHERE scl_i_id=%lld AND scl_sc_id=%lld;	無	○	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。
41		STMT_updateSC_updsc	UPDATE shopping_cart SET sc_date=NOW(), sc_sub_total=%f,sc_tax=%f, sc_ship_cost=%f,sc_total=%f WHERE sc_id=%lld;	無	○	該当せず	特に無し。プライマリキーのインデクスで一件に絞り込まれる。

12　付録.B

PostgreSQL8.1.2 の oprofile（sammary）

DEF-Htoff　(BT=195.8)				
samples	%	image name	app name	symbol name
17639139	10.548	vmlinux	vmlinux	_copy_to_user_ll
8047771	4.8125	postgres	postgres	pg_mblen
7183311	4.2955	postgres	postgres	wchareq
6483408	3.877	postgres	postgres	MBMatchText
6005517	3.5912	libc-2.3.4.so	libc-2.3.4.so	memcpy
5481221	3.2777	postgres	postgres	pg_euc_mblen
4911893	2.9372	postgres	postgres	HeapTupleSatisfiesSnapshot
4061356	2.4286	postgres	postgres	LWLockAcquire
2947260	1.7624	postgres	postgres	AllocSetAlloc
2850943	1.7048	postgres	postgres	LWLockRelease
2830082	1.6924	postgres	postgres	SearchCatCache
2648619	1.5838	postgres	postgres	pg_eucjp_mblen
2386096	1.4269	postgres	postgres	hash_search
2011121	1.2026	postgres	postgres	nocachegetattr
1686419	1.0085	postgres	postgres	slot_deform_tuple
1367909	0.818	vmlinux	vmlinux	kunmap_atomic
1338868	0.8006	psqlodbc.so	psqlodbc.so	SOCK_get_next_byte
1333917	0.7977	oprofiled	oprofiled	(no symbols)
1284226	0.7679	postgres	postgres	heapgettup
1218799	0.7288	postgres	postgres	yyparse
1143631	0.6839	postgres	postgres	OpernameGetCandidates

PostgreSQL8.1.2 の oprofile（sammary）

DEF-Hton　(BT=209.1)				
samples	%	image name	app name	symbol name
22854292	7.0436	vmlinux	vmlinux	_copy_to_user_ll
14621861	4.5064	postgres	postgres	wchareq
13494018	4.1588	postgres	postgres	MBMatchText
13485145	4.1561	postgres	postgres	pg_mblen
11677956	3.5991	postgres	postgres	pg_euc_mblen
10665174	3.287	libc-2.3.4.so	libc-2.3.4.so	memcpy
7561530	2.3304	postgres	postgres	HeapTupleSatisfiesSnapshot
6640258	2.0465	postgres	postgres	LWLockAcquire
5901483	1.8188	postgres	postgres	AllocSetAlloc
5706359	1.7587	postgres	postgres	SearchCatCache
5558912	1.7132	postgres	postgres	pg_eucjp_mblen
4746425	1.4628	oprofiled	oprofiled	(no symbols)
4683453	1.4434	postgres	postgres	LWLockRelease
4529267	1.3959	postgres	postgres	hash_search
4382318	1.3506	postgres	postgres	nocachegetattr

3980446	1.2268	postgres	postgres	slot_deform_tuple
3346251	1.0313	postgres	postgres	heapgettup
2941579	0.9066	psqlodbc.so	psqlodbc.so	SOCK_get_next_byte
2613010	0.8053	postgres	postgres	comparetup_heap
2459178	0.7579	vmlinux	vmlinux	schedule
2174547	0.6702	postgres	postgres	AllocSetFree

PostgreSQL8.1.2 の oprofile （sammary）

Tune-Htoff	(BT=325.9)			
samples	%	image name	app name	symbol name
12585382	6.4768	postgres	postgres	pg_mblen
11500540	5.9185	postgres	postgres	wchareq
10060989	5.1777	postgres	postgres	MBMatchText
8542622	4.3963	postgres	postgres	pg_euc_mblen
8049830	4.1427	postgres	postgres	HeapTupleSatisfiesSnapshot
7742895	3.9847	libc-2.3.4.so	libc-2.3.4.so	memcpy
5688123	2.9273	postgres	postgres	LWLockAcquire
4109655	2.1149	postgres	postgres	pg_eucjp_mblen
4018066	2.0678	postgres	postgres	SearchCatCache
3807287	1.9593	postgres	postgres	AllocSetAlloc
3709679	1.9091	postgres	postgres	hash_search
3647132	1.8769	postgres	postgres	LWLockRelease
2705207	1.3922	postgres	postgres	PinBuffer
2327550	1.1978	postgres	postgres	heapgettup
2021624	1.0404	psqlodbc.so	psqlodbc.so	SOCK_get_next_byte
1772262	0.9121	postgres	postgres	slot_deform_tuple
1661905	0.8553	postgres	postgres	yyparse
1546379	0.7958	postgres	postgres	_bt_compare
1500176	0.772	postgres	postgres	OpernameGetCandidates
1462892	0.7528	libc-2.3.4.so	libc-2.3.4.so	free
1458103	0.7504	libc-2.3.4.so	libc-2.3.4.so	malloc

PostgreSQL8.1.2 の oprofile （sammary）

Tune-Hton	(BT=359.6)			
samples	%	image name	app name	symbol name
22818668	6.268	postgres	postgres	wchareq
21151665	5.8101	postgres	postgres	pg_mblen
20553040	5.6457	postgres	postgres	MBMatchText
18095339	4.9706	postgres	postgres	pg_euc_mblen
13169390	3.6175	libc-2.3.4.so	libc-2.3.4.so	memcpy
11274390	3.0969	postgres	postgres	HeapTupleSatisfiesSnapshot
8810209	2.4201	postgres	postgres	LWLockAcquire
8485235	2.3308	postgres	postgres	pg_eucjp_mblen
7540740	2.0714	postgres	postgres	SearchCatCache
7422886	2.039	postgres	postgres	AllocSetAlloc
6850921	1.8819	postgres	postgres	hash_search

5863984	1.6108	postgres	postgres	LWLockRelease
5464621	1.5011	postgres	postgres	heapgettup
4366330	1.1994	psqlodbc.so	psqlodbc.so	SOCK_get_next_byte
4186139	1.1499	postgres	postgres	PinBuffer
4166903	1.1446	oprofiled	oprofiled	(no symbols)
4073231	1.1189	postgres	postgres	slot_deform_tuple
2974644	0.8171	postgres	postgres	_bt_compare
2883004	0.7919	postgres	postgres	AllocSetFree
2707121	0.7436	postgres	postgres	FunctionCall2
2608949	0.7166	postgres	postgres	yyparse

PostgreSQL8.0.6 の oprofile (sammary)

DEF-Htoff (BT=166.4)				
samples	%	image name	app name	symbol name
11156618	5.6307	postgres	postgres	pg_mblen
9782458	4.9372	postgres	postgres	HeapTupleSatisfiesSnapshot
8324546	4.2014	postgres	postgres	pg_euc_mblen
8279327	4.1786	vmlinux	vmlinux	__copy_to_user_ll
8117024	4.0966	postgres	postgres	wchareq
7085343	3.576	postgres	postgres	nocachegetattr
6487257	3.2741	postgres	postgres	LWLockAcquire
6346325	3.203	libc-2.3.4.so	libc-2.3.4.so	memcpy
5376014	2.7133	postgres	postgres	MBMatchText
4792301	2.4187	postgres	postgres	heapgettup
4747332	2.396	postgres	postgres	LWLockRelease
4641255	2.3424	postgres	postgres	ExecMakeFunctionResultNoSets
4017697	2.0277	postgres	postgres	pg_eucjp_mblen
2916766	1.4721	postgres	postgres	AllocSetAlloc
2768308	1.3972	postgres	postgres	hash_search
2610073	1.3173	postgres	postgres	SearchCatCache
2239959	1.1305	postgres	postgres	AtEOXact_CatCache
2183979	1.1022	postgres	postgres	ExecEvalVar
1873946	0.9458	postgres	postgres	pg_detoast_datum
1673356	0.8445	vmlinux	vmlinux	schedule
1547341	0.7809	postgres	postgres	PinBuffer

PostgreSQL8.0.6 の oprofile (sammary)

DEF-Hton (BT=141.9)				
samples	%	image name	app name	symbol name
15698227	4.8982	postgres	postgres	pg_mblen
13798893	4.3056	postgres	postgres	pg_euc_mblen
13432131	4.1912	postgres	postgres	wchareq
12034815	3.7552	postgres	postgres	nocachegetattr
11350821	3.5417	postgres	postgres	HeapTupleSatisfiesSnapshot
9322348	2.9088	postgres	postgres	MBMatchText

8991707	2.8056	libc-2.3.4.so	libc-2.3.4.so	memcpy
8812956	2.7499	postgres	postgres	LWLockAcquire
8437013	2.6326	postgres	postgres	heapgettup
8273494	2.5815	postgres	postgres	ExecMakeFunctionResultNoSets
7579380	2.365	vmlinux	vmlinux	_copy_to_user_ll
6166551	1.9241	postgres	postgres	pg_eucjp_mblen
5739618	1.7909	postgres	postgres	LWLockRelease
4791626	1.4951	postgres	postgres	AllocSetAlloc
4428292	1.3817	postgres	postgres	ExecEvalVar
4312392	1.3456	postgres	postgres	SearchCatCache
4181143	1.3046	vmlinux	vmlinux	schedule
4067468	1.2692	postgres	postgres	hash_search
3756048	1.172	oprofiled	oprofiled	(no symbols)
2838465	0.8857	postgres	postgres	AllocSetReset
2836699	0.8851	postgres	postgres	ExecClearTuple

PostgreSQL8.0.6 の oprofile（sammary）

Tune-Htoff (BT=254.4)				
samples	%	image name	app name	symbol name
15828899	7.673	postgres	postgres	pg_mblen
12083343	5.8573	postgres	postgres	pg_euc_mblen
11478749	5.5643	postgres	postgres	wchareq
8840425	4.2853	libc-2.3.4.so	libc-2.3.4.so	memcpy
7621954	3.6947	postgres	postgres	HeapTupleSatisfiesSnapshot
7588156	3.6783	postgres	postgres	MBMatchText
5694485	2.7604	postgres	postgres	pg_eucjp_mblen
4692573	2.2747	postgres	postgres	nocachegetattr
4579631	2.22	postgres	postgres	LWLockAcquire
3962065	1.9206	postgres	postgres	AllocSetAlloc
3570774	1.7309	postgres	postgres	SearchCatCache
3531366	1.7118	postgres	postgres	LWLockRelease
3051982	1.4794	postgres	postgres	ExecMakeFunctionResultNoSets
2955322	1.4326	postgres	postgres	AtEOXact_CatCache
2558386	1.2402	postgres	postgres	hash_search
1983811	0.9616	postgres	postgres	ExecEvalVar
1830948	0.8875	postgres	postgres	heapgettup
1687131	0.8178	psqlodbc.so	psqlodbc.so	SOCK_get_next_byte
1582671	0.7672	postgres	postgres	AllocSetFree
1390935	0.6742	postgres	postgres	yyparse
1368470	0.6634	postgres	postgres	StrategyBufferLookup

PostgreSQL8.0.6 の oprofile（sammary）

Tune-Hton (BT=227.6)				
samples	%	image name	app name	symbol name
27286246	7.7228	postgres	postgres	pg_mblen
21490506	6.0824	postgres	postgres	wchareq

20861877	5.9045	postgres	postgres	pg_euc_mblen
14487291	4.1003	postgres	postgres	MBMatchText
13173646	3.7285	libc-2.3.4.so	libc-2.3.4.so	memcpy
9436343	2.6707	postgres	postgres	pg_eucjp_mblen
8672018	2.4544	postgres	postgres	nocachegetattr
8290919	2.3466	postgres	postgres	HeapTupleSatisfiesSnapshot
6796777	1.9237	postgres	postgres	AllocSetAlloc
6444936	1.8241	postgres	postgres	LWLockAcquire
6172469	1.747	postgres	postgres	SearchCatCache
5876532	1.6632	postgres	postgres	ExecMakeFunctionResultNoSets
4346207	1.2301	postgres	postgres	LWLockRelease
4202393	1.1894	postgres	postgres	ExecEvalVar
4008073	1.1344	oprofiled	oprofiled	(no symbols)
3708534	1.0496	postgres	postgres	hash_search
3535374	1.0006	postgres	postgres	heapgettup
3176282	0.899	postgres	postgres	AtEOXact_CatCache
3014084	0.8531	psqlodbc.so	psqlodbc.so	SOCK_get_next_byte
2881220	0.8155	Vmlinux	vmlinux	schedule
2602011	0.7364	postgres	postgres	AllocSetFree

―以上―