PostgreSQL における CPU スケーラビリティの評価と改善

Evaluation and Improvement of the CPU Scalability for PostgreSQL

米 田 健 治，熊 野 禎 史

要　約　サーバ分野において，オープンソース・ソフトウェア（以下 OSS）を適用したシステムが普及・拡大している．一昔前では Web サーバ，メール・サーバ，ネーム・サーバといったネットワークのフロントエンド・サーバへの適用が中心であったが，ここ数年では JBoss® などのアプリケーションサーバや PostgreSQL，MySQL® などの RDBMS を利用したエンタープライズ・システムへの適用も珍しくなくなった．

しかし，OSS を採用するにあたっては，性能的に耐えられるか，要件を満たすためにはどのようにソフトウェア・ハードウェアを構成すればよいのか，OSS の採用によりコストはどの程度減らせるのか，など確かな情報が必要となる．

独立行政法人情報処理推進機構（IPA）が事務局を務める日本 OSS 推進フォーラムの開発基盤 WG では，このような背景から 2004 年度に OSS ミドルウェアの性能評価プロジェクトを開始した．この評価プロジェクトは，次の二点を主な目的としている．

- 性能評価の結果だけでなく，手順やツール，設定データも共有し，共通化を図る．
- 性能評価結果から，OSS の適用可能範囲を明確にする．

本稿では，2005 年度に本プロジェクトで実施した，UNISYS ES7000 による PostgreSQL の CPU スケーラビリティの評価結果について報告する．この評価の結果，PostgreSQL の性能や問題点が明らかになり，PostgreSQL の適用範囲を示すことができた．

Abstract　Open Source Software (OSS)-applied systems have spread in Enterprise server area. A few years ago, mainly, OSS would be focused on Web Server, Mail Server, Name Server. In recent years, OSS is generally applied to the application servers such as JBoss, or Enterprise System using Relational Database Server including PostgreSQL and MySQL.

However, for applying OSS to the Enterprise system, SIers need to know information on performance of OSS, how to construct the hardware and/or software with OSS, and how much reduce the cost of the system.

Japan OSS Promotion Forum Server Working Group, of which secretariat is Information-Technology Promotion Agency, Japan (IPA) started Performance Evaluation project of OSS middleware software in fiscal 2004.

The purposes of the evaluation project are as follows:

* Share the evaluation procedures, tools and setting of parameters, not only evaluation results. That would be common DB evaluation methods.
* Define the applicable scope of OSS using evaluation test results.

This paper reports evaluation results of CPU scalability of PostgreSQL using UNISYS ES7000. The evaluation results have clarified the performance and issues on PostgreSQL, and enabled to present the scope of application to Enterprise OSS system.

1. はじめに

データベース・システムの性能向上手法として，CPU やメモリの増強によるスケールアップが考えられる．この場合，データベース管理システム（以下 DBMS）の基本性能が大きく影響するので，従来のオープンソース・ソフトウェア（以下 OSS）の DBMS では複数サーバによるレプリケーションなどのスケールアウトによる性能向上が推奨されていた．そのため，高負荷な更新処理が行われる大規模システムでの OSS データベースの使用は難しいと言われていた．逆に，商用の DBMS では，CPU の増加によって同時処理能力を強化することが一般的であり，DBMS の性能として CPU スケーラビリティが重要となる．

基本性能で劣ると言われていた OSS だが，近年の急速な性能向上により，大規模システムでの使用が注目されている．特に，PostgreSQL 8.1 は，バージョン 8.0 に比べてマルチ CPU における性能向上がリリースノートなどに記述されており，CPU の増強によるスケールアップが期待できる．そこで，実際に PostgreSQL でスケールアップによる効果が得られるか確認するために，UNISYS ES7000 を使用して CPU スケーラビリティを評価することとなった．

本稿では，PostgreSQL の CPU スケーラビリティの評価手法とその結果，そこで発見したボトルネックの現象，およびボトルネックの解析手順について報告する．さらに，ボトルネックを解消するために作成したパッチの内容を解説し，そのパッチ検証結果も報告する．

これらの報告を通じて，PostgreSQL が十分にその能力を発揮できる最大 CPU 数を明らかにするとともに OSS における問題解析手法を示す．この報告は，ハイエンドサーバシステムにおける PostgreSQL 適用促進の一翼を担うと考える．

なお，本評価作業は，独立行政法人情報処理推進機構（IPA）が事務局を務める日本 OSS 推進フォーラムでの OSS 性能評価活動として実施した．

2. PostgreSQL の性能評価

性能評価に使用するツールは，ハードウェアベンダがコストと時間をかけて最高性能をはじき出すようなベンチマークではなく，誰もが簡単に実施できる評価ツールが望ましい．また，簡単に評価を行える手順が共有されていれば，測定データの比較も容易になる．そこで，OSS 推進フォーラムの OSS 性能評価活動では，評価ツールとして OSDL[*1] DBT-1（以下 DBT-1）を使用した評価手順を確立した．本評価でもその手順に従い，DBT-1 を使用した評価を行った．

また，PostgreSQL の初期設定は低性能のコンピュータでも稼働するように設定されているため，大規模サーバではその能力に見合った設定にチューニングする必要がある．2. 3 節では，本評価で行った PostgreSQL のチューニングについて述べている．

2. 1　評価ツール DBT-1 の概要

DBT-1 は，The Linux Foundation[*2] で公開されている TPC-W[*3] に準拠した簡易版データベース負荷ツールである．OSS DBMS のために，簡単に利用できるトランザクション・ベースの負荷ツールとして The Linux Foundation で開発された．DBT-1 自体も OSS である．

DBT-1 は，オンライン書店における Web ベースのユーザの処理（商品検索，ショッピングカート処理，購入手続きなど）をシミュレートしており，実行結果として 1 秒あたりのトランザクション数が得られる．DBT-1 ではトランザクションを擬似的なものとし，BT（Bogo Transaction）と表す．1 秒あたりのトランザクションは「BT/ 秒」と表現され，BT 値と呼ばれる[1]．

図1は，DBT-1の実行イメージを示している．DBDriverが負荷クライアントの役割を担い，アプリケーションはODBCドライバを使用してデータベースにアクセスする．

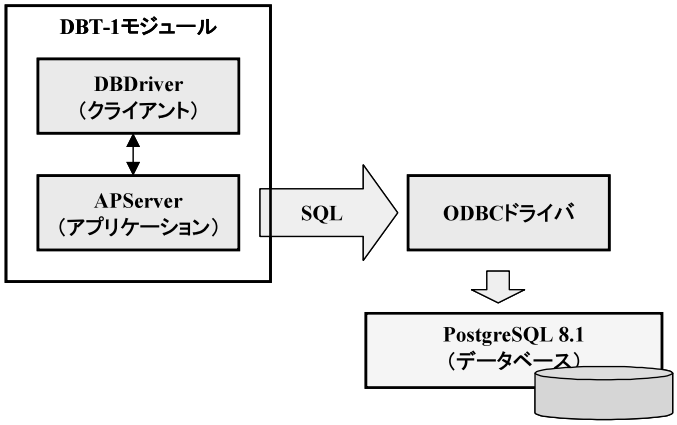

図1　DBT-1の実行イメージ

DBT-1のモデルは3階層のクライアント・サーバ型であり，クライアント（DBDriver），アプリケーション（APServer），データベース（PostgreSQL 8.1）と分割することができる．今回はCPUスケーラビリティの評価であるので，ネットワークなどの影響を受けないように同一サーバ上ですべてのモジュールを実行する．

2.2　評価結果の見方

DBT-1の実行で得られるBT値の理論値は，以下の計算式で求められる．

理論値＝仮想ユーザ数（EU）÷ユーザのオペレーション間隔（Think Time）[1]

Think Timeは一定の値とするので，仮想ユーザ数が増えれば発生するトランザクション数も増え，BT値も上がることになる．したがって，仮想ユーザ数を増やしながらDBT-1を実行し，理論値に達しない仮想ユーザ数でのBT値が，処理の限界点であると言える．図2に，仮想ユーザ数によるBT値の変化をグラフ化したものを示す．

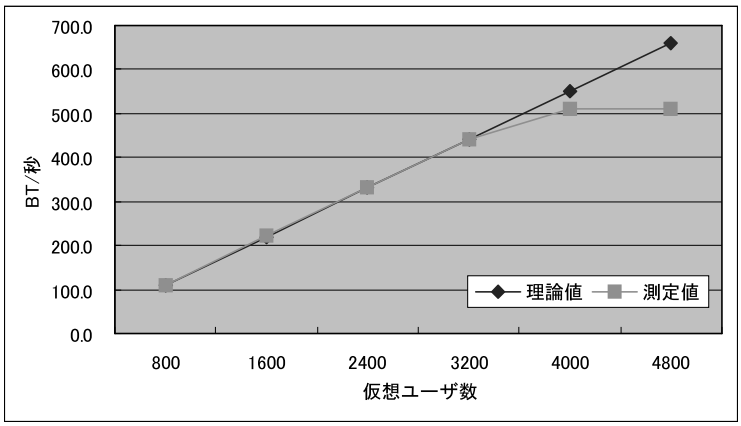

図2　DBT-1の実行結果の例

この例では，Think Time を 7.2 秒としているので，BT 値の理論値は，仮想ユーザ数 800 の場合は約 111 となり，以降は仮想ユーザ数に比例する．測定値は，仮想ユーザ数 3200 までは理論値に近い値になっているが，仮想ユーザ数 4000 以降は頭打ちとなり，BT 値は 500 強が限界であると見ることができる．

2.3　PostgreSQL のチューニング

一般に，ハードウェア環境やアプリケーションの特徴によって，チューニングの効果は変化する．ある環境で効果がみられた設定でも，他の環境では効果がマイナスになる場合もある．そのため，今回の評価環境で事前にチューニングの効果を検証し，DBT-1 のモデルに適した設定値を決定した．

本測定で変更した PostgreSQL の設定パラメータとその効果は，以下のとおりである．

- shared_buffers

  設定値を増やすことで PostgreSQL が使用する共有メモリを拡大し，より多くのデータをメモリにキャッシュさせる．メモリ・サイズに余裕がある場合は，まずこの値を増やすと良い．データ量によって効果も異なるが，今回の調査では 10000 以上の値に設定しても効果が出なかったため，初期値 1000 から 10000 に変更した．

- wal_buffers

  設定値を増やすことで，ログ先行書き込み（wal）用のバッファを拡大する．これにより，1 回のログの書き込みサイズが大きくなり，ログの書き込み回数を減らすことができる．この値も，メモリ・サイズに余裕がある場合は増やすと良い．今回の測定では，初期値 8 から 32 に変更した．

- checkpoint_segments

  設定値を増やすことで，チェックポイントが起動するセグメント数を拡大する．これにより，チェックポイントの発生回数を減らすことができる．ただし，チェックポイントの発生回数を減らすと障害発生時のリカバリに時間がかかるので，運用面からの考慮が必要になる．今回の測定では，初期値 3 から 16 に変更した．

- random_page_count

  DBT-1 のテーブルにはインデックスが設定されているものが多い．この設定値を減らすことで，インデックス・サーチが選択されるケースが多くなり，検索効率が上がる．今回の測定では，初期値 4 から 3 に変更した．

3.　CPU スケーラビリティ評価

今回の CPU スケーラビリティ評価では，IA32 アーキテクチャである ES7000/540 を使用し，表 1 に示す評価環境にて，CPU の数を 2，4，8，12，16 と変化させて DBT-1 を実行した．図 3 のグラフが評価結果である．これを見ると，CPU 数が 8 までは処理性能の伸びが見られ，スケールアップしていることがわかる．しかし，CPU 数が 12，および 16 の場合は，処理性能の伸びが見られず，CPU 数が 4 の場合と同程度になり，CPU 数 8 に比べると性能が劣化していることがグラフより読み取れる．

表 1　評価環境概要

ハードウェア	CPU	Intel® Xeon™　2.8GHz　(Hyper-Threading off)
	メモリ	16Gバイト
	ストレージ	SANARENA®1570 460.2Gバイト (RAID-0)
ソフトウェア	OS	MIRACLE LINUX® 4.0
	RDBMS	PostgreSQL 8.1.2

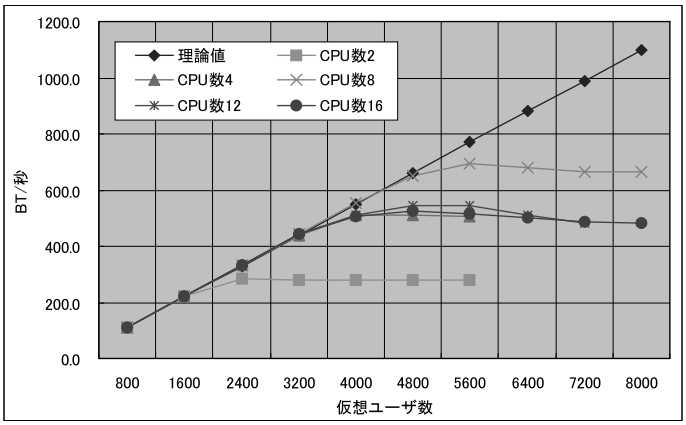

図 3　CPU の増設とスループットの推移

4.　ボトルネックの解析手法および解析結果

3 章の結果から，PostgreSQL は 8CPU を超えるとボトルネックが顕在化することがわかったため，そのボトルネックの原因を解析した．

4. 1　解析手順

OSS の場合，商用ソフトウェアと違ってボトルネックの解析手法が確立されていない．そのため，使用者がインターネットなどから独自に情報収集し，試してみる必要がある．今回，PostgreSQL のボトルネックを解析する過程で，解析手順のひとつを確立させることができた．提示する方法は，すべて OSS の標準的なツールを使用しており，PostgreSQL だけでなく一般的なシステム効率測定結果を分析する手法としても役立てることができる．

確立した解析手順は以下である．

① vmstat（4. 1. 1 項参照）による各種 CPU リソースの使用率の把握
② oprofile（4. 1. 2 項参照）による使用関数のプロファイル状況の分析
③ ①，②で特定した関数を基にしたソースコードレベルでの解析

4. 1. 1　vmstat について

Linux 標準ツールである vmstat は，リソースの使用状況を時系列に分析できる．vmstat を使用してリソース使用状況を調査することで，CPU リソースが主に何に消費されているかを概観することができる．主な CPU リソースの種類には，システム（system），ユーザ（user），I/O wait（iowait）および idle（idle）がある．今回は，ボトルネックを解析するために，CPU 待

機を示す項目である I/O wait と idle に着目した．図 4 をみるとスループットが落ちた 12CPU や 16CPU では I/O wait はほとんど無いが，idle に関しては 12CPU で 10% 以上に，16CPU では 30% 近くにまで増えてしまっていることがわかる．同じ仮想ユーザ数の 8CPU のリソース使用状況と比べると，idle が増加し，CPU リソースが有効に使えていないために BT 値が減少したと考えられる．idle が増えた原因を調べるために，oprofile で状況を詳細に分析する．

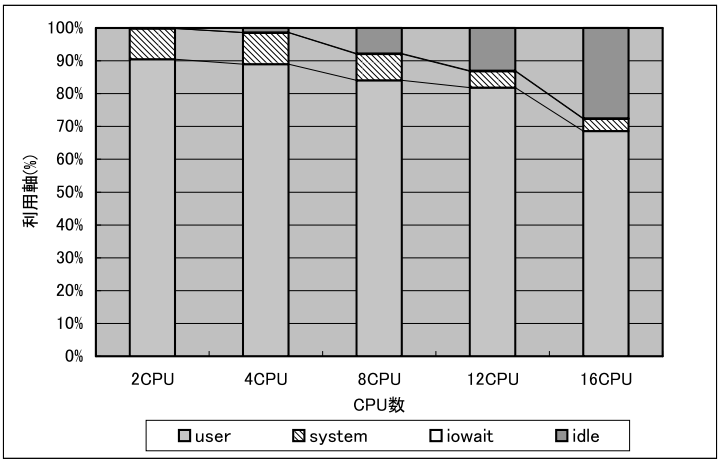

図 4　CPU の増設と CPU 利用率の推移

4. 1. 2　oprofile

oprofile はプロファイラーと呼ばれ，ある間隔でシステムを使用した関数をプロファイリングするツールである．各種イベントを登録することで，システムで発生する様々なイベントをプロファイリングすることができる．デフォルトであれば，CPU リソースを多く使用した順に関数を表示することができる．ただし，Linux のデバッグ・オプションつきのカーネルと gcc コンパイラ[*4] に -g オプションをつけてコンパイルしたソースを使用する必要がある．　プロファイラーは，プロファイリングする間隔にあわせて出現する回数が多い関数を上位に表示するため，少ない関数はその履歴リストの中に埋没してしまう可能性もある．そのため，プロファイルしたリスト状況に関しては，通常の状態と比較した上で深い洞察力が必要となる．

4. 2　ボトルネック詳細

まず，8CPU と 12CPU のプロファイリングの結果を比較することによって，ボトルネック発生時には s_lock 関数での CPU 消費が多くなっていることがわかった．

さらに，s_lock 関数を基にした oprofile のコールグラフを調査した．コールグラフとは，ある関数を基にして，その関数を呼び出した関数とその関数から呼ばれる関数に対して，それぞれの割合をプロファイリングする機能である．s_lock 関数を基準にした oprofile のコールグラフの出力結果と，その情報を基にしたソースコードの解析の結果，問題が発生する関数の呼び出しフローを読み取ることができた．そのフローにおいて，データベース・ブロックをメモリに読みこむ処理である ReadBuffer 関数からそのメモリにロックをかけるための LWLockAcquire 関数を呼んだときに s_lock 関数を多く呼ぶことがわかった．

s_lock 関数とは，PostgreSQL のなかでスピンロックを実装した関数である．スピンロック

とは，あるプロセスがあるリソースを使用しているときに，他のプロセスがそのリソースにアクセスする場合，そのリソースの解放を CPU のリソースを消費しながら待つ方法である．リソースを利用している排他時間が短い場合には，プロセス間通信による排他制御を行うよりかなり軽い実装となる．近年のロックの実装としては一般的なロック処理のひとつである．

その s_lock 関数の中では，ある一定回数ロックの獲得に失敗するとセマフォ待機となる処理フローがあった．

プロファイリングによって浮き上がった関数と，実際のソースコードの処理の流れから，12CPU において同時実行並行処理数が増えると，ロックが獲得できずに s_lock 関数での滞留時間が長くなり，結果としてそこからセマフォ[*5]待機になっているとの仮説を立てた．その仮説にたつと，現象としてはリソースのロック競合により PostgreSQL がスピンロックし，さらにセマフォ待機になり，結果として秒あたりのトランザクション処理件数が減ったのではないかと仮定できる．このことは，vmstat による CPU リソース使用状況率の idle が増加したことにも結びつく．

4. 2. 1　s_lock について

PostgreSQL は管理する内部的なリソースのロックが競合した場合，PostgreSQL 内でスピンロックを実行する s_lock 関数が実行される．

しかし，スピンロックしてもロックが獲得できない場合，カーネルのセマフォの待ち行列に入れられて，カーネルから通知されるまでそのプロセスは idle 状態になる．

メインフレームの場合，PostgreSQL で見られるスピンロックの実装では，短いスリープをすることが多い．メインフレーム上で動くソフトウェアは，CPU などのハードウェアが高価な時代に作られたため，短期間といえども CPU を消費して待機することは避けられている．その代わりに短いスリープを繰り返し，ロックの獲得を得ようとしている．スリープする時間もロックが獲得できないと徐々に長くなるように設計されている．

一方，CPU が安価になった近年ではロックの期間が短いことが想定される場合にはスピンロックを使用するほうが簡単で確実な実装といえる．

4. 2. 2　LWLockAcquire 関数について

データベース管理システムの場合，データの整合性を保つためにロック制御を実装することが不可欠である．PostgreSQL の場合，内部的なリソースのロックは“Light Weight Lock”で処理される．このロックは，共有ロックか排他ロックかの 2 種類しかないため，実装は文字通り“軽い”処理のロックである．LWLockAcquire 関数は，その軽いロックをするための汎用関数である．ロックを解く処理が LWLockRelease 関数となる．一方，表やレコードといったデータベースオブジェクトを管理する場合には，重量ロックと呼ばれる 6 種類のロックの複雑な別関数で処理する．

4. 2. 3　ReadBuffer 関数について

次に PostgreSQL が ReadBuffer 関数で処理するメモリ領域について解説する．ReadBuffer 関数は，PostgreSQL がページを探索するときに呼ばれる関数で，検索・更新時に頻繁に使われる重要な関数である．

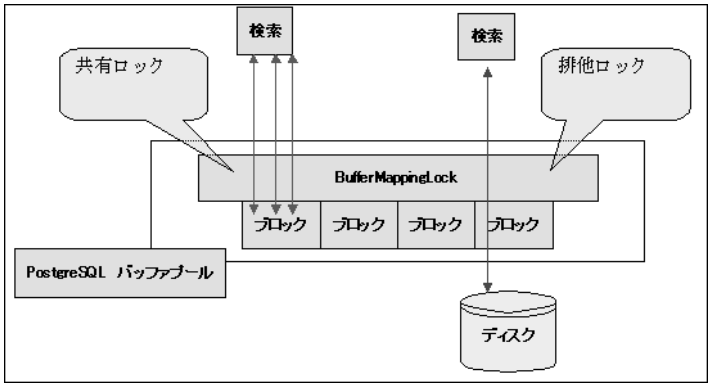

図5　メモリ管理とロック

PostgreSQL は，共有メモリ（Shared_buffer）上にブロックと呼ばれる単位でメモリを確保する（図5参照）．ブロックはディスクに対する I/O の単位であり，複数のレコードから構成される．PostgreSQL の場合，ブロックの大きさは 8K バイトである．検索処理で読むべきブロックがすでにメモリ上に読み込まれていれば，ディスクから I/O する必要はなく，共有メモリ上の内容を使用すればよい．そのため，まずは共有メモリ上に目的のブロックがキャッシュされているかを探索する必要がある．ブロックがキャッシュされていれば使用カウントに 1 を加え，他のプロセスに対して該当ブロックが使用中であることを知らせる．これは，他のプロセスが異なるブロックを読み込んで共有メモリを書き換えてしまうことを防ぐためである．一方，使用するブロックが共有メモリ上にキャッシュされていない場合，共有メモリ上のフリー・ブロック・リストから書き換えても良いブロックを探し，ディスク上のブロックに対して I/O 処理を行い，内容を書き換える．

PostgreSQL は，BufferMappingLock と呼ばれる一つの内部的なリソースに対して，ブロックがキャッシュされているか探索するために共有ロックをかける．キャッシュされていない場合，ブロックをディスクから読み込むために，BufferMappingLock に排他ロックをかける．共有ロックが BufferMappingLock にかかっている間は，排他ロックをかけることができないため，排他ロックは，同時にブロックを探している処理が終わらないと，ロックをかけられない．その際，排他ロックしようとするプロセスは，スピンロックを実行して待つ．また，排他ロックがかかっている間は，共有メモリ上に目的のブロックがキャッシュされていても探し出して使用できないため，同様にスピンロックして待つことになる．

共有メモリには，データベースの参照されるべきブロック（今回の場合，10000 ブロック）がメモリにキャッシュされ，多くのプロセスが同時に読み出し可能である．しかし，一つのプロセスが，あるブロックをディスクから入れ替える処理のために，10000 ブロック全体に対して排他処理を行おうとする．結果として，同時ユーザ数が多くなるにつれて，排他ロックをかけにくくなるのである．

5.　スケーラビリティ改善パッチの概要

今回の評価で発覚したボトルネックを解消するためにユニアデックス(株)が作成したパッチの概要について示す．

ReadBuffer 関数処理のロック基本構造は，共有メモリ上のブロックを探索するプロセスが増えるほど共有ロックと排他ロックが競合し，スピンロックする可能性が高くなり，さらにセマフォ待機による idle 状態となると考えられる．

サーバの CPU 能力が向上し，同時並行処理が増加するにつれて，そのブロックを探しているプロセスも増えることになり，排他ロックすることが難しくなる．結果として，CPU の数が増加したために共有ロックをかけているプロセスが増え，排他ロックしようとするプロセスがスピンロックしたうえでセマフォ待機することとなる．

スピンロックからのセマフォ待機によるボトルネックであることが考えられるため，スピンロックを解消すれば，idle のリソース使用率を削減できると仮定した．スピンロックを解消するためには，基本的に 2 種類の方法が考えられる．ロックする期間を最適化して短くするか，またはロックの競合を避けるかである．今回のケースでは，CPU 数増加によってロックが競合する可能性が高い構造であることから，このブロックに対するロック競合を減らすことに注力した．

今回ロックの競合が起こっている場所は，基本的なメモリ管理部分であるといえる．PostgreSQL は，BufferMappingLock に対してロックが確立した後，使用者数をカウントするためにブロックに対して“Pin”と呼ばれる処理を行う．“Pin”は，ブロックに紐付けられる使用状況の一覧表を更新する．共有メモリにキャッシュされているブロックのリンクは，表名とブロック番号でハッシュされている．今回は，ブロックを探すために共有ロックをかけているが，ブロック自体はすでにハッシュされているため，共有メモリ全体にロックしたとしても，結局はハッシュされたキー値によってブロックが分散されることになる．結果として，ハッシュ単位にロックをかければ，他のブロックとロックが競合しないはずである．当然，同じハッシュ・チェイン上のブロック同士の処理は競合してしまうが，共有メモリ全体をロックするよりはるかに競合が減る（図 6 参照）．

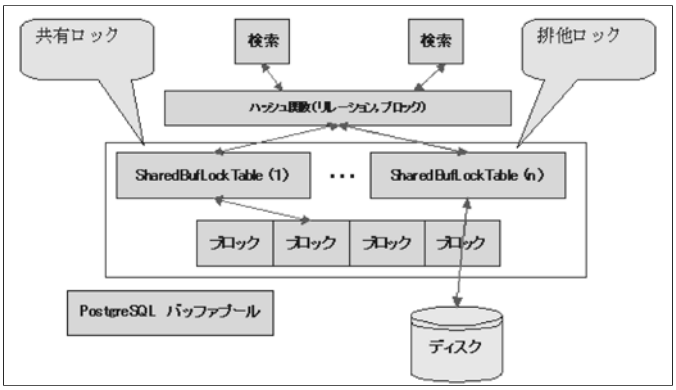

図 6　パッチ適用後のメモリ管理とロック

5. 1　パッチ検証結果

解析内容に基づいて，ロックする単位をブロックによってハッシュされたキーの単位で分散させるような PostgreSQL のパッチを作成した．そのパッチが適用された PostgreSQL8.1.3 で DBT-1 を用いて効率測定した結果が図 7 である．

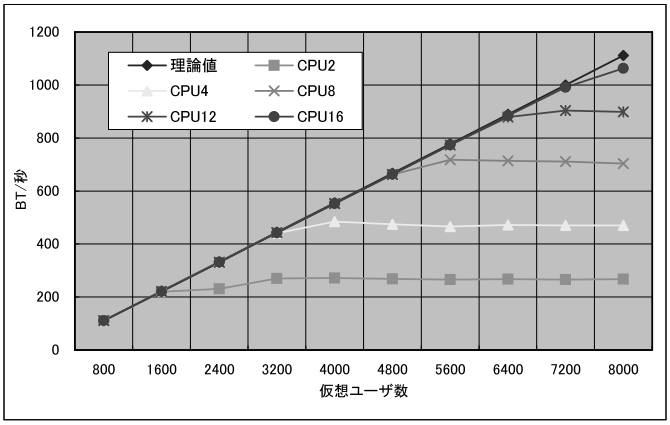

図 7　パッチ適用後の CPU 数とスループットの推移

結果として 12CPU 以上においても期待通りにスケールアップするようになり，ボトルネックに対して立てた仮説が正しかったこと，およびパッチの有効性が証明できた．

6. お わ り に

今回の評価によって，PostgreSQL 8.1 はパッチを適用しない場合 CPU 数 8 までスケールアップすることが確認できたが，同時にボトルネックが内在することによる性能の限界も見られる結果となった．OSS の場合，常にどこに限界があるのかを見極めることが重要となる．

今回の CPU スケーラビリティの評価手順や結果は，IPA が開設している OSS 情報提供サイト「OSS iPedia」（http://ossipedia.ipa.go.jp/）を通じて，広く全世界に公開されている．さらに，PostgreSQL 8.2 では，今回ユニアデックス(株)が作成したパッチと類似した修正が行われており，今回のボトルネックが解消されていることを，その後の評価で確認している．これらの評価から，少なくとも PostgreSQL の DBT-1 モデルにおいては 16CPU 以上の CPU スケーラビリティがあることを実証でき，PostgreSQL はエンタープライズ用途でも十分に実用性があることを示している．

今回の評価では，16CPU までしか計測していないが，パッチ適用後の PostgreSQL の計測グラフを見る限り，まだ仮想ユーザ数を増やすことができそうである．機会があれば，CPU 数をさらに 20，24 と増やした計測を行い，PostgreSQL の基本性能，特に CPU スケーラビリティの向上に貢献していきたいと考える．

＊ 1　Open Source Development lab の略称．Linux のビジネス利用を推進するための非営利組織．
＊ 2　Linux のビジネス利用を推進するための非営利組織．2007 年に OSDL と Free Standards Group の合併により設立された．（http://www.linux-foundation.org/）
＊ 3　トランザクション処理性能評議会で策定された，電子商取引（EC）のベンチマーク．
＊ 4　GNU プロジェクトによるフリーな C コンパイラ．
＊ 5　OS レベルでプロセス間の同期を取る仕組みのこと．

参考文献　［1］　日本 OSS 推進フォーラム，「DB 層の評価」報告書，2005 年 3 月，
http://www.ipa.go.jp/software/open/forum/development/index.html

執筆者紹介　米　田　健　治（Kenji Yoneda）

1987年日本ユニバック(株)入社．IXシリーズ・メインフレームのデータベース管理システムの主管作業に従事．2004年からはOSS関連業務に着手し，日本OSS推進フォーラムのメンバとしてOSSの普及活動に参加．現在，ユニアデックス(株)プロダクト事業グループ ソフトウェアプロダクト統括部 OSS推進部に所属．

熊　野　禎　史（Yoshifumi Kumano）

1991年日本ユニシス(株)入社．IXシリーズ・メインフレームの言語プロセッサ，ランタイム・システムの主管作業に従事．2004年からはOSS関連業務に着手し，日本OSS推進フォーラムのメンバとしてOSSの普及活動に参加．現在，ユニアデックス(株)プロダクト事業グループ ソフトウェアプロダクト統括部 OSS推進部に所属．