

# PostgreSQL ハードウェアパフォーマンスチューニング

2001年12月17日

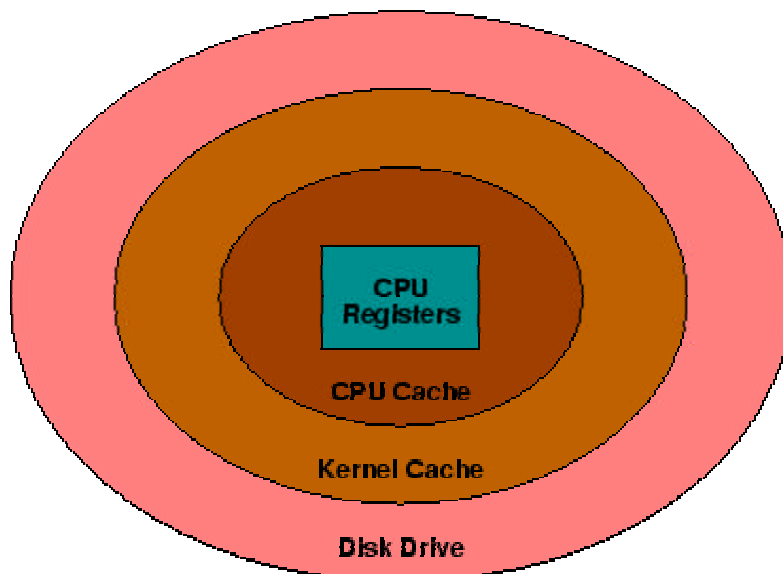
POSTGRESQL は世界各地に点在する開発者集団により、インターネット上で開発されたオブジェクトリレーショナルデータベースです。Oracle や Infomix のような商用データベースの替わりとなるオープンソースデータベースです。

POSTGRESQL はカリフォルニア大学バークレイ校で独自に開発されました。1996年には、あるグループがインターネット上で開発をはじめました。彼らはアイデアを電子メールで交換し、ファイルサーバを使用してソースコードを共有しました。POSTGRESQL は今日、機能、性能、信頼性において商用データベースに匹敵します。また、トランザクション、ビュー、ストアドプロシージャ、参照性制約をサポートしています。ODBC、Java(JDBC)、TCL/TK、PHP、Perl、Python などといった多数のプログラムインタフェースもサポートしています。POSTGRESQL はインターネットの有能な開発者たちによって非常に早いペースで改善され続けています。

## 1 パフォーマンスコンセプト

データベースのパフォーマンスチューニングには 2 つの側面があります。1 つ目はコンピュータの CPU、メモリ、ディスクをデータベースがどう使うかの改善です。2 つ目はデータベースに送られた問い合わせを最適化することです。この文章はハードウェアの側面からパフォーマンスチューニングについて記載します。問い合わせの最適化は CREATE INDEX、VACUUM、VACUUM ANALYZE、CLUSTER、EXPLAIN といった SQL コマンドを用いることで効果を上げることができま。これらは本著、*PostgreSQL : Introduction and Concept* - <http://www.postgresql.org/docs/awbook.html> (日本 PostgreSQL ユーザー会訳「はじめての PostgreSQL」ピアソン・エデュケーションズ ISBN4-89471-461-2) の中で述べられています。

ハードウェアパフォーマンスに関することを理解するためには、コンピュータの中でどのようなことが起こっているかを理解することが重要です。単純にコンピュータはストレージに囲まれた中央演算装置(CPU)として考えることができます。CPU 内のチップ上では、中間処理結果、変数ポインタ、カウンタが格納されたいくつかの CPU レジスタが存在します。これを囲んでいるのが最新のアクセス情報を保持した CPU キャッシュです。CPU キャッシュをさらに取り囲むように、実行中のプログラムやデータを保持する大容量のランダムアクセス主記憶領域 (RAM) があります。RAM を取り囲むようにより大きな情報を保持するディスクドライブがあります。ディスクドライブは唯一の永久的情報格納領域ですから、コンピュータの電源を落としても保持する必要のあるものはここに記録しなければいけません。まとめると、これが CPU を取り囲む記憶領域となります。



格納領域	計測単位
CPU レジスタ	バイト
CPU キャッシュ	キロバイト
RAM	メガバイト
ディスクドライブ	ギガバイト

CPU から遠くなるにしたがって格納領域が増大しているのがわかると思います。理想的には、非常に大きな永続的記憶領域を CPU のすぐ隣に配置するのがよいのですが、動作が非常に遅く、高価になってしまいます。実際は、最も頻繁に使われる情報を CPU のそばに格納し、あまり使わない情報を遠くに格納して必要に応じて CPU に取り込みます。

## 2 CPU の近くに情報保持

種々の格納領域間における情報の移動は自動的に行われます。コンパイラはどの情報がレジスタに格納されるべきかを決定します。CPU チップの論理構造では、最近使用された情報を CPU キャッシュに格納します。基本システム(OS)はどの情報が RAM に格納されるか、そしてどの情報がディスクドライブから読み込まれ、またディスクドライブへ戻されるかを制御します。

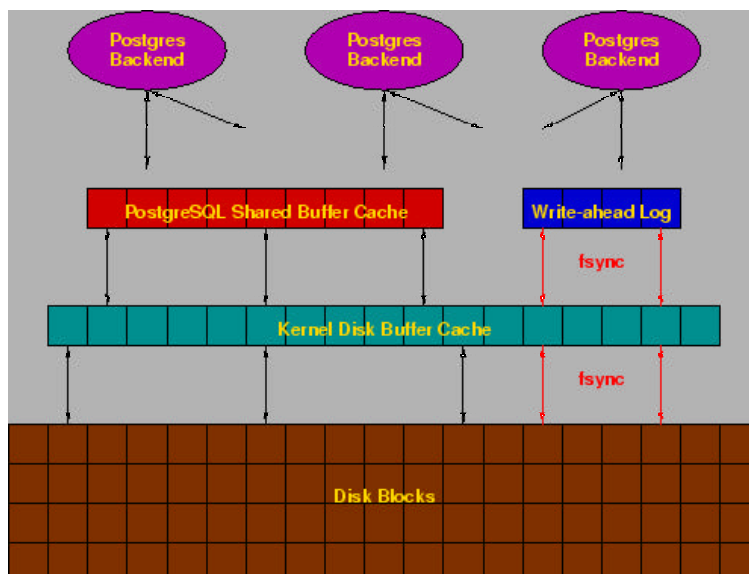
CPU レジスタと CPU キャッシュはデータベース管理者では効果的にチューニングすることができません。効果的なデータベースのチューニングは、よく使う情報を大量に RAM に格納することです。そうすることで可能な限りディスクアクセスが妨げられます。

これは簡単にできると考えるかもしれませんが、そうはいきません。コンピュータの RAM にはたくさんのものが格納されているからです。

- 実行中のプログラム
- プログラムのデータとスタック
- PostgreSQL の共有バッファキャッシュ
- カーネルディスクバッファキャッシュ
- カーネル

OS が使用するメモリ以外の領域に影響を及ぼさない間は可能な限り多くのデータベース情報を RAM に格納しつづけることが適切なチューニングとなります。

## 3 PostgreSQL の共有バッファキャッシュ

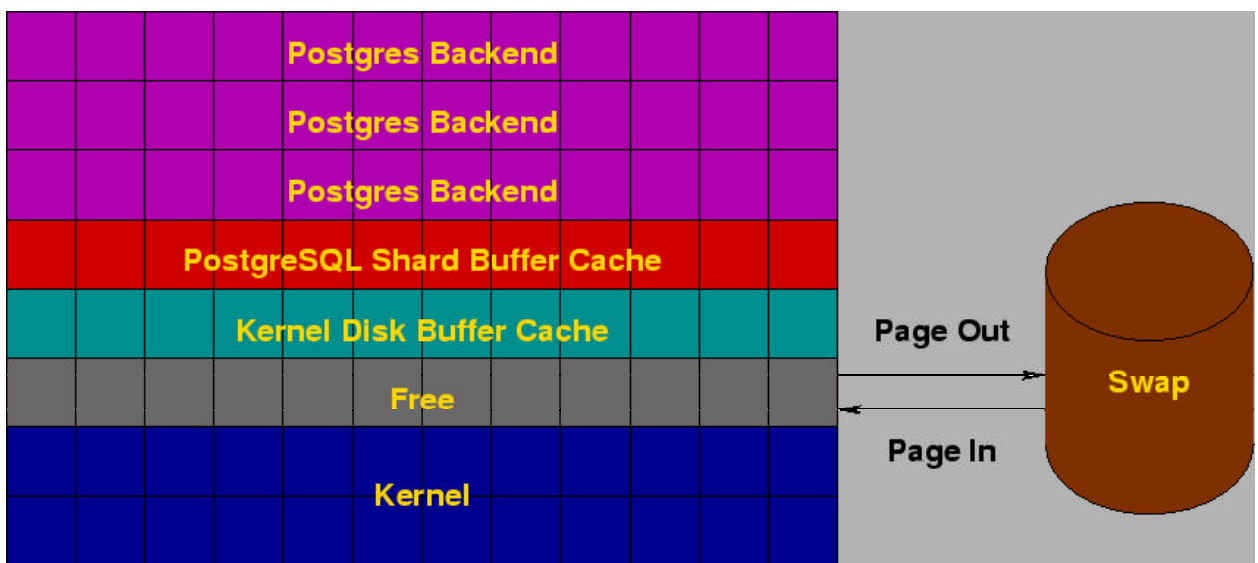


POSTGRESQL はディスク上の情報を直接書き換えません。その代わりに、ディスクに格納された情報を POSTGRESQL 共有バッファキャッシュに読み出すように要求します。そのとき、POSTGRESQL のバックエンドはこれらのブロックを読み書きし、最終的にディスクに書き戻します。

テーブルにアクセスする必要があるバックエンドは、最初にこのキャッシュ内の必要なブロックを検索する必要があります。既にそこにデータがあれば、すぐに処理を続けます。もしなければ、OS の要求によって該当ブロックをロードします。そのブロックは、カーネルバッファキャッシュもしくはディスクからロードされます。これらは非常にコストがかかる操作です。

デフォルトの POSTGRESQL の設定では、64 個の共有バッファが割り当てられます。それぞれのバッファサイズは 8 キロバイトです。バッファの数を増加させると、バックエンドが必要な情報をキャッシュからより見つけやすくなり、コストの高い OS 要求を避けることができます。これは `postmaster` コマンドの `-B` コマンドラインオプションか、`postgresql.conf` の `shared_buffers` の値を変えることで変更できます。

#### 4 どれくらいのメモリを割り当てるべきか？



「RAM を全て POSTGRESQL 共有バッファに割り当てたい」と考えるかも知れません。しかし、そうしてしまうとカーネルや他のプログラムが動作するための領域がなくなります。POSTGRESQL 共有バッファの適切なサイズは、他の動作に影響を与えない最も大きなサイズです。

ちゃんとした動作を理解するために、Unix OS がどのようにメモリ管理を行うのか 理解する必要があります。もしプログラムとデータを保持するのに十分なメモリがあれば、ほとんどメモリ管理は必要とされません。逆に RAM が少なければ、カーネルはメモリページをスワップと呼ばれるディスク領域に書き出しはじめます。その動作は、最近使われていないメモリページから移動します。この操作は、**スワップページアウト**と呼ばれています。ページアウトは活動していない期間の間におこるため問題ありません。何が悪いのかと言うと、これらのページがスワップから戻される必要がある時で、スワップに移動された古いページがもう一度 RAM に戻される必要がある時です。これは**スワップページイン**と呼ばれています。スワップからページが移動される間、ページインが完全に終了するまでプログラムはサスペンドするため良くありません。

ページインの動作は `vmstat` や `sar` などのシステム解析ツールによって見ることができ、機能を果たすための十分な使用可能メモリが無いことを確認できます。通常システム操作の一部であるファイルシステムからページを読み出す動作を含む、通常のページインとスワップページインを混同しないでください。スワップのページインが見つからない場合、ページアウトが多くなっていれば、それが現在スワップのページインをも実行中であることの良いバロメータとなります。

(訳注: linux の `vmstat` の場合、`si`: スワップページイン、`so`: スワップページアウト、`bi`: ページアウト、`bo`: ページインがそれぞれ該当する部分となります。 `bi`、`bo` がその名前に対して文章中では逆の意味を指しますので注意してください。)

## 5 キャッシュサイズの影響

何故キャッシュサイズがそれほど重要なのかと不思議に思うかもしれません。まず、POSTGRESQL 共有バッファが完全にテーブルを保持するのに十分大きいものを想像してください。テーブルの連続的な検索の繰り返しはデータが既にキャッシュ内にあるため、ディスクアクセスを要求しません。今度はテーブルの1ブロックより小さいものを想像してください。テーブルの連続的な検索は最後のテーブルブロックに達するまで全てのテーブルブロックをキャッシュにロードし続けます。そのテーブルのブロックが必要となった場合、一番古いブロックはすぐに削除され、キャッシュにはテーブルの最初のブロックが格納されます。次にキャッシュに格納された最初のブロックは必要ないので、新たなブロックをロードするため削除されます。この場合、キャッシュにはテーブルの2番目のブロックが格納されます。この次に必要なブロックの押し出しはテーブルの最後のブロックに到達するまで続きます。これは極端な例ですが、キャッシュを1ブロック小さくすることでキャッシュの効果が100%から0%に変化することがわかつてと思います。正しいキャッシュサイズを探すことは、劇的にパフォーマンスに影響を与えることを示しています。

## 6 共有バッファキャッシュの適当なサイジング

理想的な POSTGRESQL 共有バッファキャッシュサイズは以下の通りです：

- 最も良くアクセスされる（複数の）テーブルを格納するのに十分の大きさ
- スワップのページイン動作を回避するのに十分の小ささ

*postmaster* は起動時に（指定した）全ての共有メモリを確保することを覚えておいてください。この領域は誰もデータベースにアクセスしなくても同じ領域でありつづけます。OS が行ういくつかのページアウトは、他の OS では共有メモリを RAM ロックするのに対し、共有メモリを参照しませんでした。

POSTGRESQL 管理者ガイド（<http://developer.postgresql.org/docs/postgres/kernel-resources.html>）に種々の OS のカーネル設定情報が書かれていますのでそちらも参照してください。

（訳注：日本語マニュアルは <http://osb.sra.co.jp/PostgreSQL/Manual/PostgreSQL-7.1-ja/kaernel-resource.html> にあります。）

## 7 ソートメモリのサイズ

もう一つのチューニングパラメータはソート処理に使用するメモリの総量です。大きなテーブルまたは結果をソートする時、POSTGRESQL はそれらを部分的にソートし、一時ファイルに中間結果を格納します。これらのファイルは全ての行がソートされるまでマージと再ソートが行われます。ソートメモリサイズを増加させると多くの一時ファイルが作成され、より速いソート処理が行われます。しかし、ソートメモリが大きすぎるとソート中にスワップへのページアウトが起こるため、ページインを引き起こします。この場合、ソートメモリを少なくし、多くの一時ファイルを作成するようにすることで多くのメモリが確保された時にスワップページインを限定し、より速く処理を行うことができます。このパラメータは ORDER BY、CREATE INDEX もしくはマージ結合(*merge join*)といったソート処理をおこなうすべてのバックエンドにおいてそれぞれ使用されることを覚えておいてください。いくつか同時にソートが行われるとメモリの使用量の総計はそれらの整数倍となります。

この値は *postmaster* コマンドの *-o '-S'* コマンドラインオプションを変更するか、*postgresql.conf* の *sort\_mem* の値を変更することで変更できます。

## 8 キャッシュサイズとソートサイズ

キャッシュサイズとソートサイズは共にメモリ使用量に影響を及ぼしますので、片方に影響を及ぼすこと無しに一方を最大値にすることができません。キャッシュサイズは *postmaster* 起動時に確保されソートサイズが実行されるソートの数により変化することを覚えておいてください。一般にキャッシュサイズはソートサイズより重要だとされています。しかし、ORDER BY、CREATE INDEX もしくはマージ結合を用いた特定の問い合わせでは多くのソートメモリを使用したほうが速

度の向上を望めます。

また、多くの OS では共有メモリの割り当て量に制限があります。この制限を拡張するには OS 固有のカーネルの再コンパイルや再設定の知識を必要とします。POSTGRESQL 管理者ガイド

( <http://developer.postgresql.org/docs/postgres/kernel-resources.html> )に多くの情報があります。

( 訳注：日本語マニュアルは <http://osb.sra.co.jp/PostgreSQL/Manual/PostgreSQL-7.1-ja/kaernel-resource.html> です。)

## 9 ディスクの位置

ディスクドライブの物理的特性にもとづく性能特性によって、この文章で記載したほかの記憶媒体領域とは異なったものになっています。他の記憶媒体（メインメモリ等）はどの部分も同じ速度でアクセスすることができます。回転するプラッタと移動するヘッドで構成されるディスクドライブはヘッドの現在位置から遠いところのデータより、近いところのデータにずっと速くアクセスします。

プラッタ上の別のシリンダにディスクヘッドを移動させるには、更に多くの時間を要します。Unix カーネル開発者はこのことを知っていて、ディスクに大きなファイルが格納される時は、分割されたファイルの分片がそれぞれ近くに配置されるように 試みます。例えば、ディスク上の 10 ブロックが要求されるファイルを仮定します。OS がブロック番号 1-5 をひとつのシリンダに配置し、ブロック番号 6-10 を別のシリンダに配置したとします。ファイルが最後まで読み込まれるのに、たった 2 回のヘッドの移動( 1 回目はブロック番号 1-5 が保持されたシリンダを得るために、2 回目はブロック番号 6-10 が保持されたシリンダを得るために )ですみます。しかし、もしファイルが連続的に読まれない場合（例えば ブロック番号 1,6,2,7,3,8,4,9,5,10 の順で読み込まれる時）10 回のヘッドの移動が要求されます。このように、ディスクによる連続的なアクセスはランダムアクセスより非常に速くアクセスできることがわかります。これは PostgreSQL が、あるテーブルの大量の行データを読み込む必要がある場合に、インデックス検索より連続的な検索を好む理由です。また、これはキャッシュの価値を最大限に高めます。

## 10 複数枚のディスクのスピンドル

データベースの動作中はディスクのヘッドが小刻みに動き回ります。非常に多くの読み書き要求が発生すると、ドライブは飽和しはじめパフォーマンスの低下を引き起こします。( *vmstat* と *sar* でそれぞれのディスク上での動作を確認することができます。)

ディスクの飽和を軽減させる解決方法の一つに、PostgreSQL データファイルを別のディスクに移動するというものがあります。同一ディスク上の別ファイルシステムにデータファイルを移動しても意味がないことに注意してください。一つのドライブ上にあるすべてのファイルシステムには同一のディスクヘッドが使用されるからです。

データベースアクセスをディスクドライブに散在させるにはいくつかの方法により可能です：

### データベースの移動

*initlocation* で別のドライブにデータベースを作成することができます。

### テーブルの移動

テーブルとインデックスのファイルをシンボリックリンクとして他のドライブに移動することができます。移動の際は PostgreSQL を必ず停止して行ってください。また、PostgreSQL はそのファイルがシンボリックリンクかどうかは関知しませんのでテーブルを削除し再作成する場合、そのデータベースのデフォルトの場所に作成されなければなりません。バージョン 7.1 では *pg\_database.oid* と *pg\_class.relfilenode* がデータベース名、テーブル名、インデックス名とそれらの実体である数字のファイル名とのマッピングを行っています。

### インデックスの移動

インデックスのファイルをそれらのヒープテーブルからシンボリックリンクとして他のドライブに移動することができます。これにより、片方のディスク上でインデックススキャンを実行し、2 つ目のディスクでヒープ検索を行うことができます。

### 結合(Joins)の移動

結合されたテーブルをシンボリックリンクとして別々のディスクに移動することができます。もし、テーブル A

と B が結合されていれば、片方のドライブで A を処理し、2 つ目のドライブで B を処理することができます。

#### ログの移動

pg\_xlog ディレクトリをシンボリックリンクとして別のディスクドライブに移動することができます (pg\_xlog は PostgreSQL release 7.1 以降に存在します)。他の書き込みと違い、PostgreSQL のログ書き込みはトランザクションが完了する前にディスクにフラッシュされている必要があります。これらの書き込みを遅延させるためキャッシュを使用することはできません。ログ書き込みを別のディスクで行うと、ディスクヘッドを現在のログシリンダの場所に停滞させておくことができるのでヘッド移動の遅延無しに書き込み処理を行うことができます。( *postgres -F* パラメータを使用するとログ書き込みをディスクにフラッシュするのを防ぐことができますが、OS がクラッシュした場合バックアップファイルからリストアしなければいけません。)

(訳注: ログを書き込まずにクラッシュした場合、ログが無いので自動復旧できません。)

他のオプションは 1 つのファイルシステムをいくつかのドライブに点在させるような RAID の機能を含んでいます。

## 11 まとめ

幸運にも PostgreSQL は多くのチューニングを要求しません。多くのパラメータは最適なパフォーマンスが出るように自動で調整されます。キャッシュサイズとソートメモリのサイズは、管理者がより多くの利用可能なユーザメモリを割り当てる制御を行うことができます。ディスクアクセスもまたドライブを分散配置できます。他のパラメータは *share/postgresql.conf.sample* で設定することができます。このファイルを *data/postgresql.conf* にコピーして使用します。PostgreSQL の珍しいパラメータのいくつかも試すこともできます。

***Bruce Momjian***

訳: 日本 PostgreSQL ユーザー会

2001-12-29