PostgreSQLの
パラメータチューニングについて

2009.6.20
JPUG 仕組み分科会 勉強会

NTT OSSセンタ
笠原 辰仁

アジェンダ

- 本日のアジェンダ
  - DBMSのチューニングについて
  - パラメータチューニングに際して
  - チューニングを始める前に
  - 変更しておきたいパラメーター覧
  - その他
  - チューニングと対になるもの

はじめに

- 本日の趣旨
  - PostgreSQLを安心して使うために
    - どのパラメータを触ればよいのか？
    - どういう考え方で変更値を決めればよいのか？
  - 上記を(簡単に)ざっくりとお伝えします
  - 凝ったパラメータの話は別途に
  - 性能問題で困った・・・というケースのいわゆる「トラブルシューティング」でのチューニングの話も別途に
    - ただし、今回の話は「トラブルを避ける」意味もあるので、「プレシューティング」という位置づけになると思います

DBMSのチューニング

- チューニングには様々なアプローチがあります

ハードウェアレベル	CPU、メモリ、ストレージをスケールアップ RAID10 vs RAID5
ミドルウェアレベル	コネクションプーリングで接続コスト軽減 ロードバランサ/レプリケーションで負荷分散
論理/物理的な スキーマ設計レベル	テーブルスペースでIO負荷分散 巨大なテーブルはパーティショニング テーブルの垂直分割(TOAST回避)
AP/SQLレベル	関数インデックス、部分インデックスの利用 LIKE句ではなく全文検索の活用
パラメータレベル	work_mem調整でソート/ハッシュ処理の高速化 *_page_costやenable_*で実行計画の改善

本日はパラメータに関するチューニングのお話

パラメータチューニングに際して(1/2)

- 最近のPostgreSQLはチューニング対象となるパラメータが減少
  - パラメータ数的には機能追加に伴い増加中
    - 103(ver7.4)→119(ver8.0)→146(ver8.1)→154(ver8.2)→167(ver8.3)→169(ver8.4)
  - ただし、昔に比べると自動で最適化してくれるものが増加
    - 特にbgwriterやfree_space_mapなどの裏方の常駐プロセスで扱うパラメータ
  - pgtuneといった支援ツールも利用可能
    - 簡単な入力要素(メモリサイズ、使い方、コネクション数)でチューニングした設定ファイルを出力
  - 変更すべき / しておくと良いパラメータは、実はそれほど多くありません
    - 100以上ある内のせいぜい20未満くらい

(参考)pgtuneによる変更(@8.4β)

- メモリ8GB、接続数80での各用途別の値(空白は変更なし)

パラメータ	dw	mixed	oltp	web	desktop
default_statistics_target	100	50			
maintenance_work_mem	960MB	480MB	480MB	480MB	480MB
constraint_exclusion	on	on			
checkpoint_completion_target	0.9	0.9	0.9	0.7	
effective_cache_size	5632MB	5632MB	5632MB	5632MB	1920MB
work_mem	48MB	48MB	96MB	96MB	16MB
wal_buffers	32MB	8MB	8MB	4MB	1536kB
checkpoint_segments	64	16	16	8	
shared_buffers	1920MB	1920MB	1920MB	1920MB	480MB
max_connections	80	80	80	80	80

変更点は10項目
work_memについて、oltp、web多すぎ、dwは少なすぎ?・・・oltpとwebでは
スワップが心配

パラメータチューニングに際して(2/2)

- パラメータは、以下の様に変更すると良いです
  - 必ず変えておく必要のあるものを変更
    - 問題の発生を確実に防ぐため
    - 問題があった場合に有用な情報を取得しておくため
    - メンテナンスや監視に関わる箇所
  - 変更するのが面倒なもの
    - 再起動を要する・・など
  - 機械的に変更できるもの
    - HWリソースやDBサイズを考慮して決められるもの

ここまでこなしておけば、大抵のケースではOK(のはず)

- 以下は、より最適化を目指すために・・
  - 用途によっては「より効果的になるもの」を変更

チューニングを始める前に

- HWやDBMSの状況を確認しておきましょう
  - パラメータ変更の際の基準になるためです
  - HW
    - CPUコア数は？
    - メモリサイズは？(DBMSで占有できるサイズは？)
    - ストレージの能力
      - 容量、スピンドル数、キャッシュサイズ
  - DBMS
    - DBの合計サイズは？
      - その内、アクティブな(アクセス頻度の高い)領域の割合は？
    - DB内にあるテーブル/インデックスの個数やそれぞれのサイズは？
      - 特に大きなものは把握しておきましょう
    - どのようなワークロードが主ですか？
      - 更新が多い？ほとんど参照？
    - ボトルネックになりそうなのは？
      - CPUバウンドになりそう？I/Oバウンドになりそう？
    - 監視しておくべき情報は？
      - スロークエリは把握する必要はある？ユーザ別に知りたい情報が異なる？

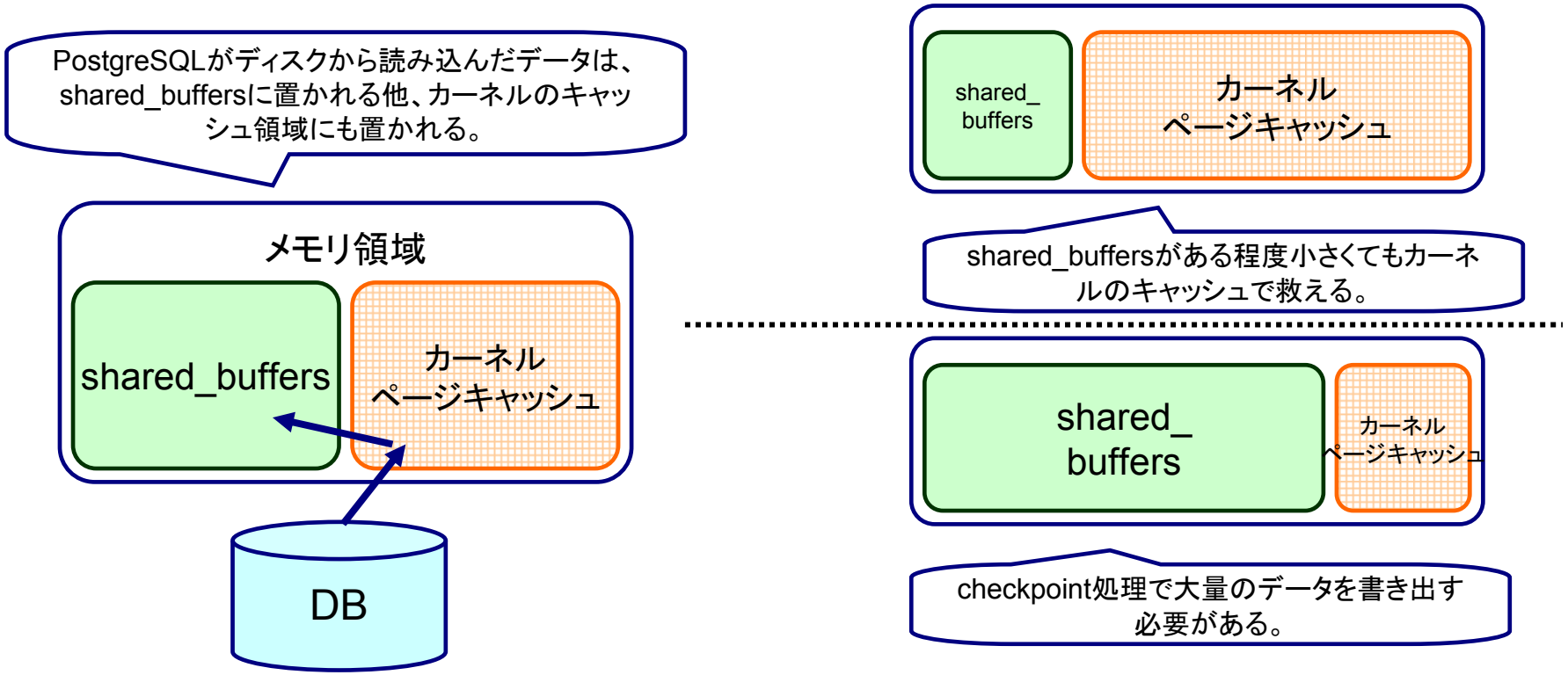
変更しておきたいパラメーター覧(対象8.1 - 8.4)

パラメータ名	対象ver	お勧め設定値(方針)
重要度(★★★)		
shared_buffers	(all)	搭載メモリの10～20%
max_fsm_pages / relations	(～8.3)	pages:DBサイズ(byte)*0.2/8192 relations:DBクラスタ内のテーブル/インデックス総数 + DB数 * 160
log_destination / redirect_stderr(logging_collector)	(8.1～)	(destination, redirect_stderr) = (‘syslog’, off) or (‘stderr’, on) or (‘syslog, stderr’, ‘on’) 必ずどこかにログを残す設定に！
log_min_duration_statement	(8.1～)	許容できるSQLレスポンス時間の上限
checkpoint_segments	(all)	16 - 64 (更新処理が多いほど多めに)
重要度(★★)		
superuser_reserved_connections	(all)	管理者権限で同時に実施する処理数 + 1
effective_cache_size	(all)	搭載メモリの50% - 60%
stats_block_level / row_level	(～8.2)	on
autovacuum_max_workers	(8.3～)	DBサイズの20%以上のサイズのテーブル数 + 1
autovacuum_vacuum_cost_limit	(8.3～)	200程度で様子見
重要度(★)		
work_mem	(all)	trace_sort、EXPLAIN ANALYZE 等で調整
bgwriter_all_percent / pages	(～8.2)	percent : 5, pages : 50～ (更新処理が多いほど多めに)
random_page_cost	(～8.2)	3.0～2.0
default_statistics_target	(8.1～)	大きなテーブルへのLIKEがある場合は100程度に(ロングトランザクションに注意)

「対象ver」は、それぞれのパラメータについて変更しておくと良いバージョンを示しています。
そのパラメータを扱えるバージョンという意味ではないです。

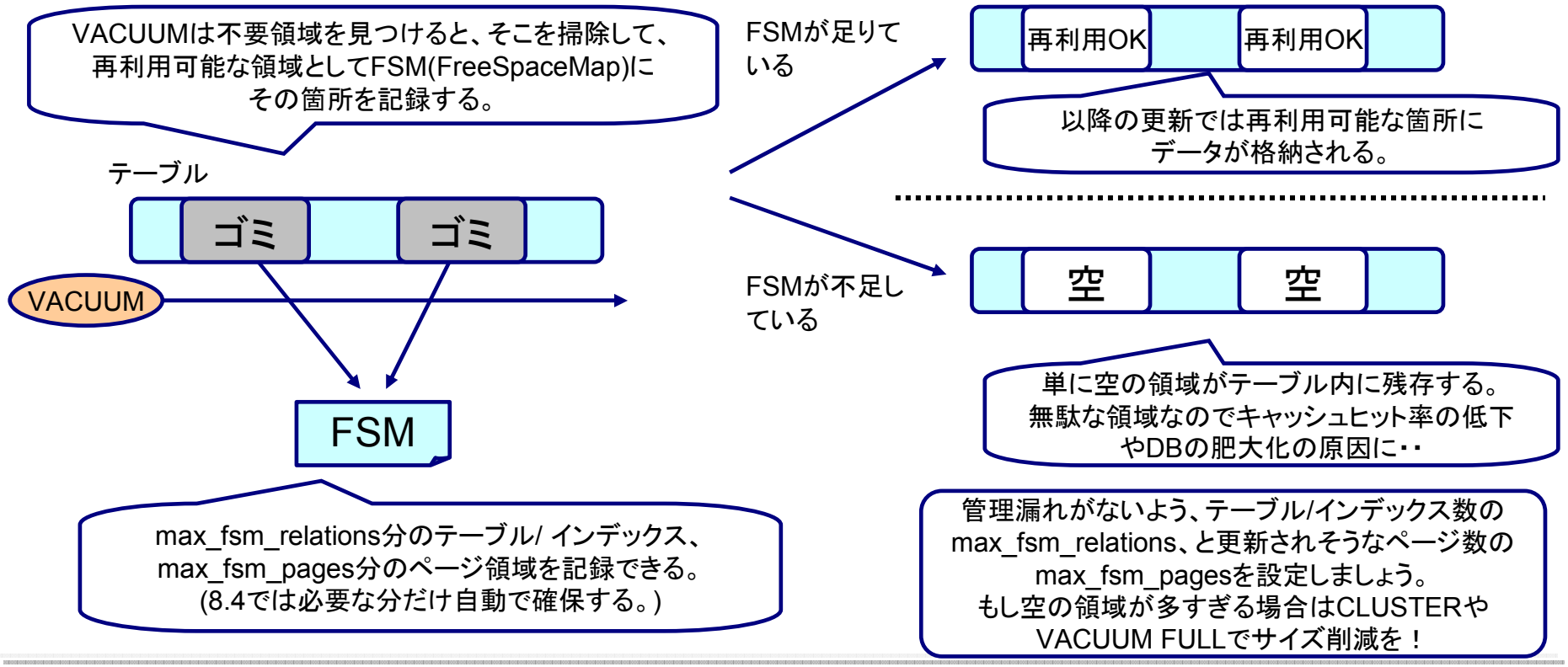
shared_buffers (ALL)

- 何をする？
  - PostgreSQLの共有メモリのサイズを調節
- 値はどうする？
  - 物理メモリの10 - 20%程度に設定
- 設定次第で問題が起こる？
  - (小さすぎると)キャッシュヒット率の低下、(大きすぎると)チェックポイントスパイク発生



max_fsm_pages / relations (～8.3)

- 何をする？
  - VACUUMで掃除をした不要領域の再利用マップのサイズ調節
- 値はどうする？
  - pages : DBサイズの20%(byte) / 8192 以上 (VACUUMされるまで20%程度更新される場合)
  - relations : DBクラスタ内のテーブル+インデックス + DB*160(←システムテーブル分)
- 設定次第で問題が起こる？
  - (小さすぎると)VACUUMをしてもDBサイズが増加しつづける



log_destination / redirect_stderr / log_min_duration_statement(ALL)

- 何をする？
  - log_destination：PostgreSQLのログの出力先調節
  - log_stderr：標準出力エラーのリダイレクト調節
  - log_min_duration_statement：規定時間以上かかったSQLの出力調節
- 値はどうする？
  - syslogに出力する場合：log_destination = ‘syslog’
  - ユーザファイルに出力する場合：log_destination = ‘stderr’、redirect_stderr = on
  - syslogとユーザファイルに出力する場合：：log_destination = ‘syslog, stderr’、redirect_stderr = on
  - log_min_duration_statement： 許容できるSQLレスポンス時間(ただし、syslogには出力しないほうが良い)
- 設定次第で問題が起こる？
  - 有用な情報が取得できない

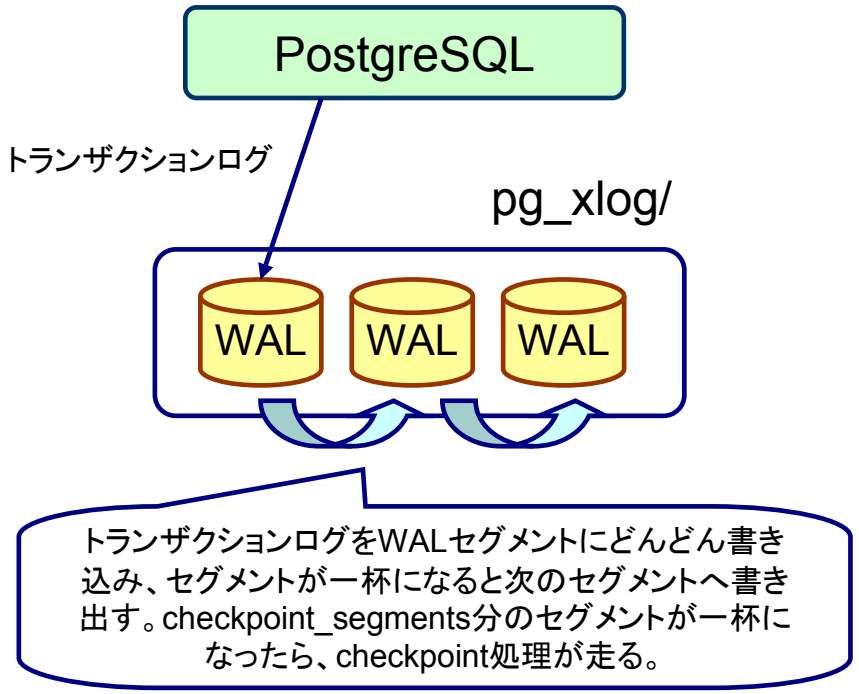
ログ情報は必ず取りましょう。syslogの他、ユーザの指定したファイルにも書き出せます。
ログに出力する情報を色々と制御できますが、有用なのは以下のものです。

お勧め！
■ log_min_duration_statement：スロークエリ(遅いSQL)を把握できます
あると便利
■log_lock_waits　　：deadlock_timeout 以上の時間ロック待ちしたSQLを把握できます
■log_error_verbosity：ERRORの発生したソースコード位置、SQLエラーコードが把握できます

なお、大量のログをsyslogに出すとそこがボトルネックになります。特に、検証時などのSQLを大量にロギングしたい場合などはユーザファイルに出すと良いでしょう。

checkpoint_segments(ALL)

- 何をする？
  - WALセグメントの数を調節
- 値はどうする？
  - 16 ～ 64以上 に設定(更新処理が多いほど多めに)
- 設定次第で問題が起こる？
  - (小さすぎると)チェックポイントスパイクが発生



checkpoint_setmentsの値が小さいと、すぐにcheckpointが頻発する。
デフォルトの3では小さすぎることが多いので、増やしておくと吉。特に更新処理が多い場合は必須。

checkpoints are occurring too frequently…
というログが出たら大抵はこれが原因です。

なお、WALセグメント(1個16MB)は最大で「checkpoint_segments * 3 + 1」のサイズになるので、その分のストレージは確保しましょう。

effective_cache_size(ALL)

- 何をする？
  - PostgreSQL内部でのキャッシュサイズの調節
- 値はどうする？
  - 物理メモリの 1/2 ～ 2/3 に設定
- 設定次第で問題が起こる？
  - (小さすぎると)インデックススキャンを実施してくれない

PostgreSQLがインデックススキャンにかかるコストを内部で算出する際のパラメータとして使用しています。このパラメータ自体は、増やしても別にリソースを消費するわけではありません。

古いHWを基準に初期値が定められているため、非常に低い値です。(最近は多めに取ってくれるようになりましたが・・それでもまだ低いです)

とりあえずメモリ量の50～60%までは引き上げてきましょう。

superuser_reserved_connections(ALL)

- 何をする？
  - 管理者(スーパーユーザ)用に確保する接続数の調節
- 値はどうする？
  - 管理者権限で同時に実施する処理数 + 1(予備)
- 設定次第で問題が起こる？
  - (足りないと)管理者権限で接続できず、重要な処理が実施できない

max_connections =
 管理者権限での接続(superuser_reserved_connections) + その他接続数
です。
十分な数のsuperuser_reserved_connectionsを確保しておかないと、いざという時に
大事な処理ができません。

なお、8.2以前のautovacuumは、暗黙的に管理者権限の接続を使用してますので、注意。

stat_command_string / stat_block_level / row_level (～8.2)

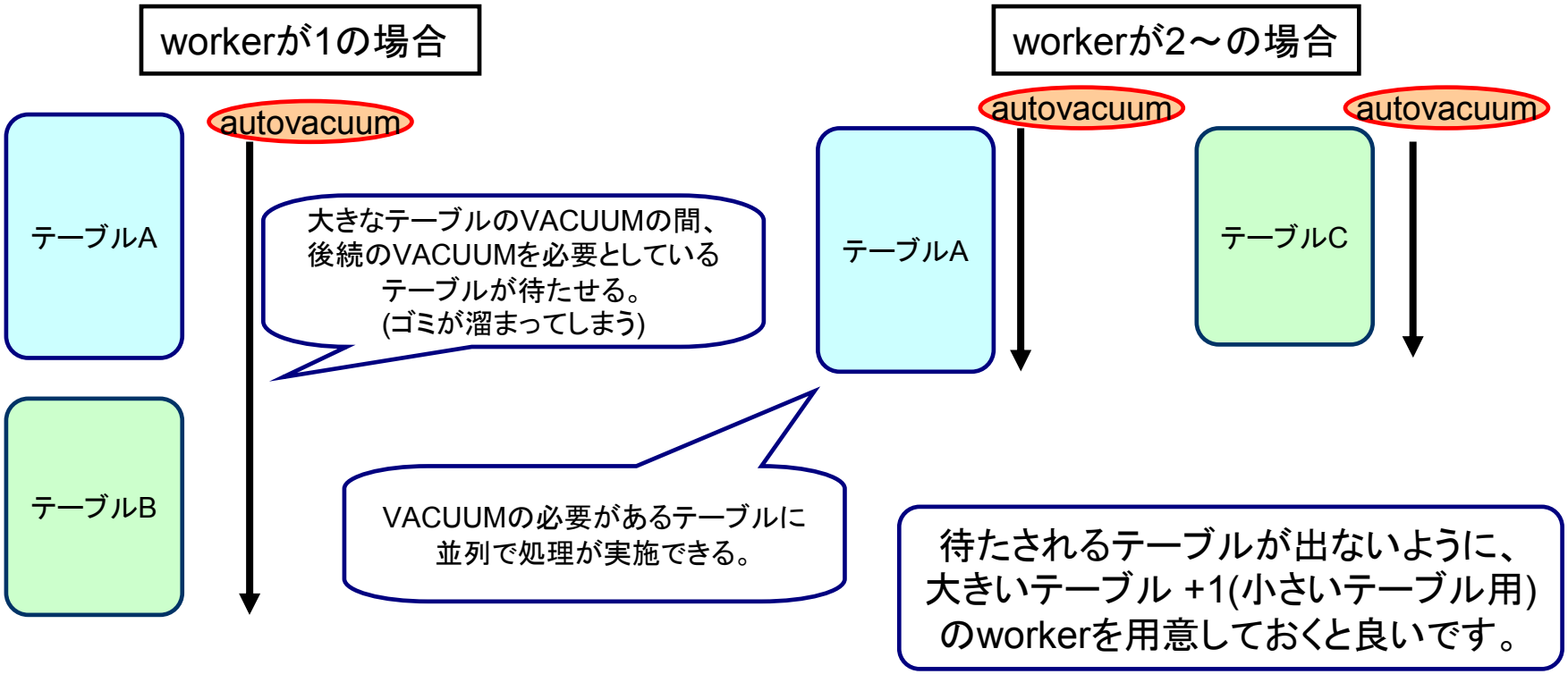
- 何をする？
  - DBのアクティビティ(実施中のSQLやスキャン数など)を捕捉するかどうか
  - 上記を稼動統計情報と呼びます
- 値はどうする？
  - on
- 設定次第で問題が起こる？
  - (offだと)有用な稼動統計情報が取得できない

PostgreSQLでは、stats collector というプロセスが、DB内の様々なアクティビティ情報を収集し、記録しています。これらの情報は pg_stat_* というシステムビューで提供され、非常に有用です。8.2以前では各テーブルやインデックスへのアクセス回数や読み取られたブロック数を記録するかどうかのオプションがoffになっています。ぜひonにしておきましょう。
性能的にも数%のオーバーヘッドがかかる程度です。
なお、8.2でautovacuumを使う場合は、stats_low_levelがonになっている必要があります。

8.3からはtrack_countsというパラメータになっており、デフォルトでonです。

autovacuum_max_workers(8.3～)

- 何をする？
  - 同時に動くautovacuumプロセス数の調節
- 値はどうする？
  - 大きな(DBの2割近いサイズ)テーブルの数 + 1
- 設定次第で問題が起こる？
  - (小さすぎると)autovacuumを必要とするテーブルが待たされる



autovacuum_vacuum_cost_limit(8.3～)

- 何をする？
  - autovacuumの遅延の有無と遅延時間の調節
- 値はどうする？
  - 200程度に設定(ディスク性能が高ければ多めに)
- 設定次第で問題が起こる？
  - (-1のままだと)autovacuumがリソースを喰ってしまう、(大きすぎると)autovacuumの遅延がされない

PostgreSQL内部では、もしautovacuum(またはVACUUM)の遅延が有効になっていた場合。バッファを読んだり書き換えたりする度に、それらの処理に予め相当するコストをカウントアップしていく。具体的には・・

■バッファ上のページ読み込みの際キャッシュヒットした = vacuum_cost_page_hit (def=1)
■バッファ上のページ読み込みの際キャッシュヒットしなかった = vacuum_cost_page_miss (def=10)
■バッファ上のページを書き換えた = vacuum_cost_page_dirty (def=20)

これが cost_limitを超えると、autovacuum（またはVACUUM)はcost_delay時間分休止する。

autovacuumが実施される契機は不定。サービスの繁忙期にautovacuumが実施されると、マシンリソースの一部が取られてしまう。そのため、処理の忙しい時はautovacuumを「ゆっくり」行う設定をして、システムの本業にリソースを与えられるようにすると良い。

変えておくと良いパラメーター覧(対象8.1 - 8.4)

パラメータ名	対象ver	役割 / お勧め設定値(方針)
maintenance_work_mem	(all)	VACUUMやREINDEX処理用の作業メモリ領域の調節 - 128MB程度～(大きなテーブルがあるほど多めに)
work_mem	(all)	ソートやハッシュ処理用の作業メモリ領域の - trace_sort や EXPLAIN ANALYZE(8.3～)を見ながら調節
random_page_cost	(～8.2)	PostgreSQL内部でのランダムアクセスのコスト調節 - 0.2程度まで下げる(インデックススキャンを使ってくれない場合)
bgwriter_all_percent / pages	(～8.2)	バックグラウンドライターによるダーティバッファの書き出し量調節 - percent:5～、pages:50～ 程度で様子見(更新処理が多いほど多めに)
default_statistics_target	(8.1～)	ANALYZE時のサンプリング数の調節 - 行数の多いテーブルへのLIKEがある場合は100程度に
constraint_exclusion	(8.1～)	CHECK制約を考慮して、検索対象とするテーブルを絞りこむ - パーティション機能を使用している場合は on (8.4では partition)に
log_autovacuum_min_duration	(8.3～)	規定時間以上かかったautovacuumのロギング調節 - 10min 程度に設定し、時間のかかったautovacuumの処理内容を把握

(参考)trace_sort

- trace_sortは、隠し？設定パラメータ
  - postgresql.confに”trace_sort = on”と記述
    - sortの処理内容をログに出力できます

```
LOG:  begin tuple sort: nkeys = 1, workMem = 1024, randomAccess = t
LOG:  switching to external sort: CPU 0.00s/0.00u sec elapsed 0.00 sec
LOG:  performsort starting: CPU 0.01s/0.01u sec elapsed 0.05 sec
LOG:  finished writing run 1: CPU 0.01s/0.01u sec elapsed 0.06 sec
LOG:  finished writing final run 2: CPU 0.02s/0.02u sec elapsed 0.09 sec
LOG:  finished merge step: CPU 0.02s/0.02u sec elapsed 0.10 sec
LOG:  performsort done: CPU 0.02s/0.02u sec elapsed 0.10 sec
LOG:  external sort ended, 103 disk blocks used: CPU 0.05s/0.03u sec elapsed 0.16 sec
```

ディスクの103ブロックを使用：ソート処理のために
work_memを増やす必要がある

```
8.4はEXPLAIN ANALYZEでソート処理内容を確認可能
=# EXPLAIN ANALYZE SELECT * FROM test2 ORDER BY 1;
                                  QUERY PLAN

-------------------------------------------------------------------------------------------------
 Sort  (cost=1.27..1.29 rows=10 width=4) (actual time=0.112..0.115 rows=10 loops=1)
   Sort Key: id
   Sort Method:  quicksort  Memory: 25kB (メモリ不足の時はexternal sort  Disk: nnnnnnkB)
   ->  Seq Scan on test2  (cost=0.00..1.10 rows=10 width=4) (actual time=0.010..0.013 rows
=10 loops=1)
 Total runtime: 0.155 ms
(5 rows)
```

その他

- 以下の点もチェックしましょう
  - そもそも動かない or 問題発生の可能性が・・
    - カーネルパラメータ
      - shmmax、shmall
      - shared_buffersより大きくしないとFATALエラーで起動できません
      - Linuxであれば”/etc/sysctl.conf/”に設定を記述します
    - max_connections * work_mem + shared_buffers
      - 上記が物理メモリ量を大きく超えるとスワップの危険
      - カーネルの消費するメモリ(1GB程度)くらいは余るように！
  - pg_hba.conf
    - アクセス制御用のファイル
    - リモートからのアクセスを許可していますか？

その他

- TIPS
  - パラメータの一部はユーザやDB毎に設定値を与えることが可能です
    - 例1:あるユーザのみ、実施したSQLを全て記録したい
      - ALTER USER some_user SET log_statement TO 'all‘;
    - 例2:あるDBのみ、1分以上かかったSQLを強制的に終了させたい
      - ALTER DATABASE some_db SET statement_timeout TO '60s';

```
変更の確認例
=# SELECT usename, useconfig FROM pg_user;
 usename   |      useconfig
-----------+---------------------
 some_user | {log_statement=all}

=# SELECT datname, datconfig FROM pg_database ;
  datname  |              datconfig
-----------+-------------------------------------
 some_db   | {statement_timeout=60s}
```

  - パラメータの一部はSET文で動的に変更することもできます
    - 例1: ある処理の時だけ、work_memを大きくしたい
      - psql some_db
      - =# SET work_mem TO ‘128MB’;

チューニングと対になるもの

- チューニングの効果を観測できる手段を持っておきましょう
  - OSで取得できるリソース情報
    - vmstat、iostat、sar、ps、top….
      - 手軽に取得できます
  - PostgreSQLで取得できるリソース情報
    - pg_stat_*….
      - PostgreSQLの活動状況が動的に把握できます
    - EXPLAIN / EXPLAIN ANALYZE
      - SQLの実行計画が把握できます
    - ログ情報
      - 必須です
  - その他
    - ベンチマーク
      - pgbench ....
    - プロファイラ
      - Oprofile、Dtrace、systemtap….

おわり

- 本日に紹介しきれなかったパラメータについては、
  lets postgres などで解説 or 次回以降の勉強会で！

- PGCon2009@オタワで使用された資料が良い参考になります
  - http://www.pgcon.org/2009/schedule/events/188.en.html

- ご清聴ありがとうございました