アシスト
「お客様の最高」のために

LPI-JAPAN
Linux Professional Institute Japan

OSS-DB Exam Gold 技術解説無料セミナー

2015/02/22

株式会社 アシスト
データベース技術本部
喜田 紘介

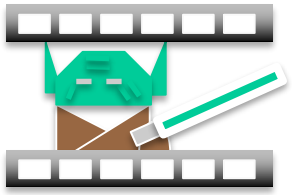
© LPI-Japan 2014. All rights reserved.

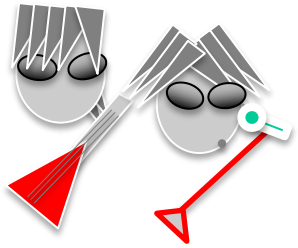
講師プロフィール

アシスト 「お客様の最高」のために　OSS-DB

■プロフィール

- 名前　喜田 紘介 (きだ こうすけ)
- 所属　株式会社 アシスト データベース技術本部
- 趣味



■PostgreSQL関連の活動

- Oracle DBの構築、教育、記事執筆、トラブル対応などのフィールド支援を経て2011年9月よりPostgreSQLの専任技術として活動。
- 現在は、新規構築するシステムのRDBMS選択支援や、商用DBからの移行難易度の評価を行う移行アセスメント支援を主に担当。

2011年　OSS-DB Silver 取得(12月)
2012年　OSS-DB Gold 取得(12月)
2013年　SQL逆引き大全 363の極意 執筆
2014年　日本PostgreSQLユーザ会 広報・企画担当就任(6月)

© LPI-Japan 2011. All rights reserved.

オープンソースデータベース（OSS-DB）に関する技術と知識を認定するIT技術者認定

OSS-DB /Silver

データベースシステムの設計・開発・導入・運用ができる技術者

OSS-DB /Gold

大規模データベースシステムの
改善・運用管理・コンサルティングができる技術者

OSS-DB技術者認定資格の必要性

商用/OSSを問わず様々なRDBMSの知識を持ち、データベースの構築、運用ができる、または顧客に最適なデータベースを提案できる技術者が求められている

© LPI-Japan 2011. All rights reserved.

OSS-DB Goldに求められること

アシスト 「お客様の最高」のために

OSS-DB

OSS-DB /Silver

データベースシステムの設計・開発・導入・運用ができる技術者

- RDBMSやPostgreSQLの構造の理解
- メンテナンスコマンドの理解（オプションレベルで、何ができるか知っている）
- SQL開発

OSS-DB /Gold

大規模データベースシステムの改善・運用管理・コンサルティングができる技術者

- PostgreSQLの詳細な構造の理解（たとえば、データの格納方式）
- メンテナンスや障害対応の必要性の判断と適切な実施
  - 出力されたログを見て、適切なメンテナンスを実施できる
  - 恒久対処のための詳細なパラメータの設定ができる
- 広い視野でチューニングができる
  - チューニングの要否を判断するための情報収集
  - 適切な対処（SQLチューニング、パラメータ設定、H/WやOSの設定）

© LPI-Japan 2011. All rights reserved.

Gold試験範囲

アシスト 「お客様の最高」のために　OSS-DB

■運用管理

- データベース構造や管理コマンド全般についての詳細設定まで理解し、健全なデータベースを構築できる

  ※ホットスタンバイ運用による大規模環境を含む

■性能監視

- アクセス統計情報やプランナ統計情報、各種ビュー、実行計画について理解し、その他の性能監視手法も活用できる

■チューニング

- 性能関連パラメータの理解や、性能監視の結果を正しく判断し、チューニングを行う

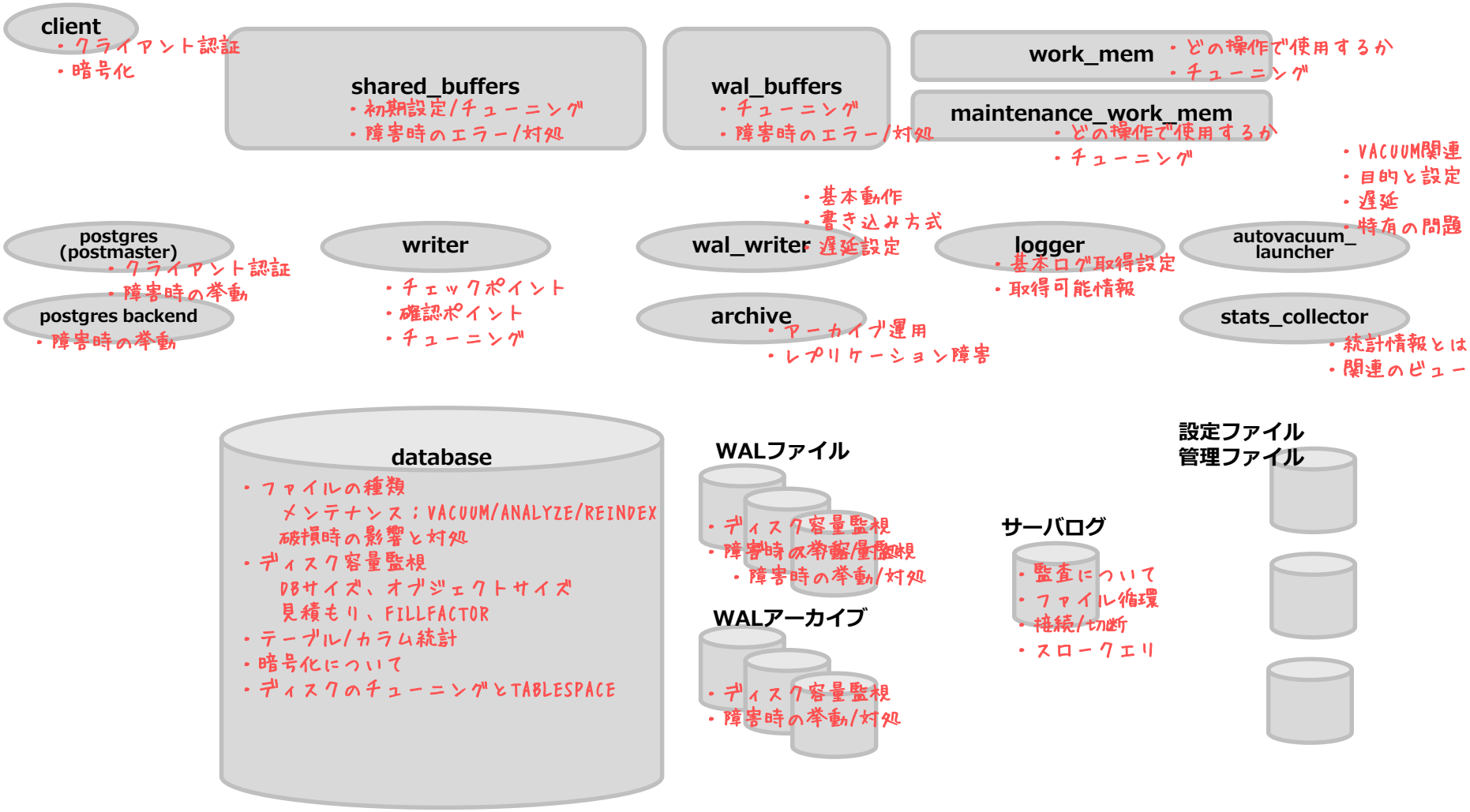
■障害対応

- 障害パターンとそれぞれに対する状態確認や復旧方法を理解している

OSS-DB出題範囲 (http://www.oss-db.jp/outline/examarea.shtml) では
コマンド例などの詳細を記載

© LPI-Japan 2011. All rights reserved.

■メモリ、プロセス、ディスク領域からなるDB構造を正しく把握



© LPI-Japan 2011. All rights reserved.

DB管理者の業務

広範囲な試験範囲をDB管理者の業務に当てはめて理解

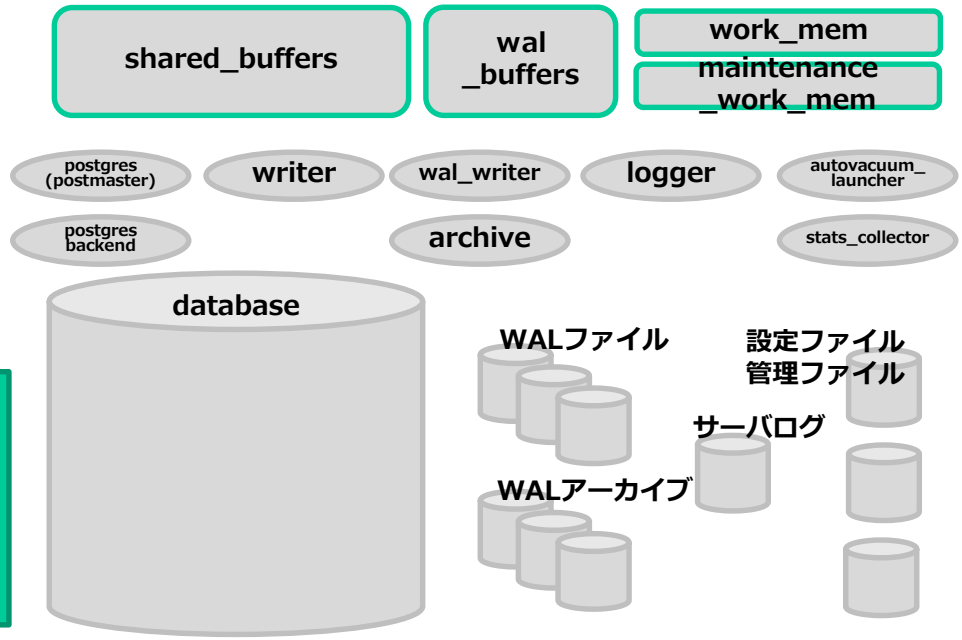
- サーバ構築
  - OSの設定～PostgreSQLのインストール
- 初期設定
  - サーバサイジング
  - パラメータ設定
  - セキュリティ
- 監視
- メンテナンス
  - オブジェクトのメンテナンス
  - ユーザのメンテナンス
  - 起動・停止
- チューニング
  - プロセスの詳細動作（DBチューニング）
  - SQLチューニング
- 障害復旧

© LPI-Japan 2011. All rights reserved.

領域ごとに用途を確認

- 共有バッファ
  - ディスクから読み取ったデータをキャッシュして、以降のユーザ要求に高速に応答
- WALバッファ
  - ログ先行書き込み(Write Ahead Logging)
  - 耐障害性とパフォーマンスを両立するための仕組み
  - WALファイルへのI/Oをシーケンシャルにするために、メモリ上に変更をキャッシュ
- work_mem
- maintenance_work_mem
  - セッション毎に確保される領域
  - ソートやハッシュの一時領域
  - メンテナンス操作

メモリ関連では、チューニング方法と、
各領域に障害が発生した時の動作を
整理しておくこと

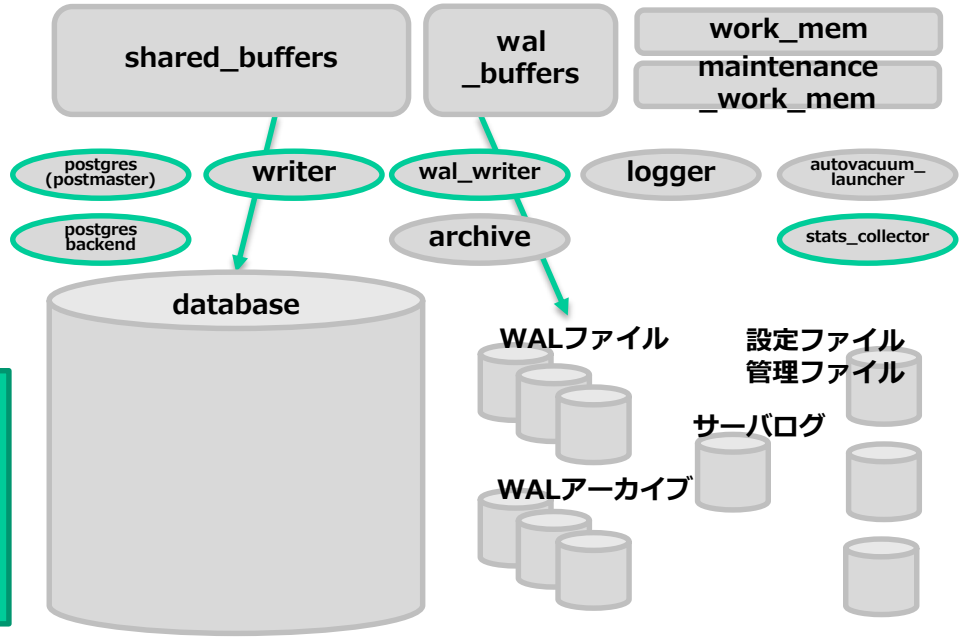


© LPI-Japan 2011. All rights reserved.

必須プロセス

- postgres (postmaster) 、postgres backend
  - クライアントからの接続を待ち受ける、すべてのプロセスの親プロセス (postgres)
  - postgresプロセスによって起動され、クライアントからの処理を担当 (backend)
- writer
  - 共有バッファのデータをディスクに書き込むプロセス
  - チェックポイントやダーティバッファの書き込み
- wal writer
  - データの変更履歴をWALファイルに書き込む
- stats collector
  - 実行時統計情報を収集する

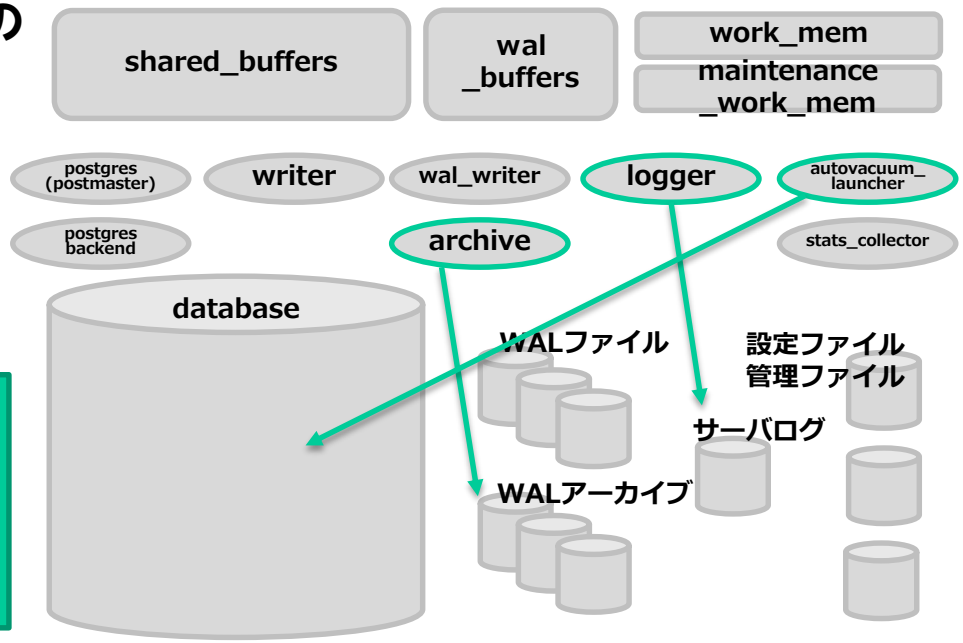
プロセス関連では、各プロセスが担う役割を詳細に理解し、動作を調整できることが期待される



© LPI-Japan 2011. All rights reserved.

パラメータ設定により起動するプロセス

- logger
  - PostgreSQLサーバ実行時のログを記録するプロセス
  - パラメータ設定により有効化し、何をどこに保存するか指定できる
- archive
  - チェックポイント以前の不要なWALをPITRのために別のディスクに退避
- autovacuum launcher/worker
  - 自動VACUUMの閾値を超過したもの(表・列)に対してVACUUMを実行



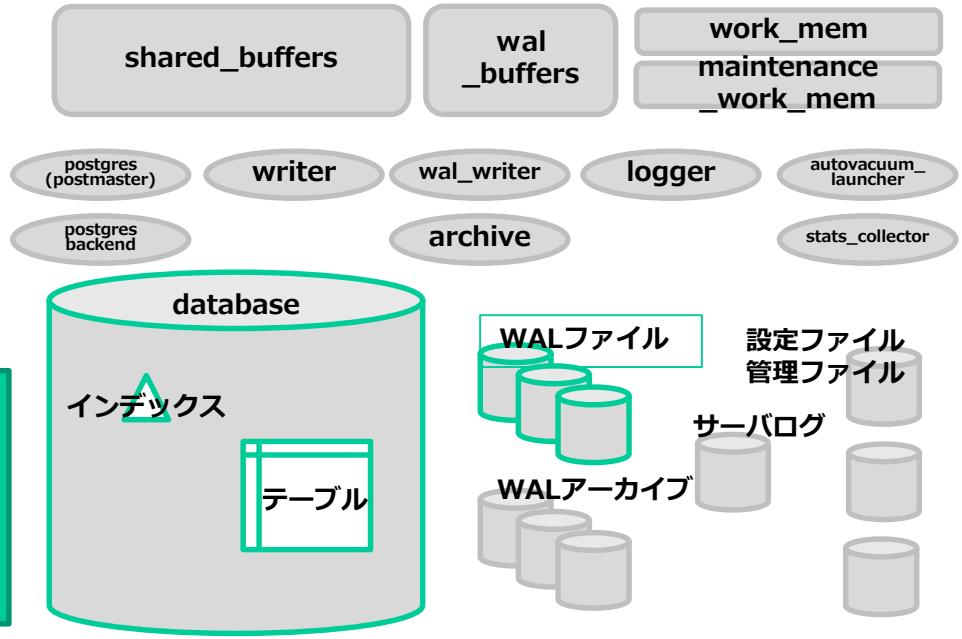
これらのプロセスはGoldの範囲では
有効が前提であり、非常によく問われる
詳細設定まで理解している必要がある

© LPI-Japan 2011. All rights reserved.

データベースクラスタの物理構造

- データベースクラスタ
  - PostgreSQLサーバのルートディレクトリ
- 配下のディレクトリ・ファイル
  - base：データベース毎にディレクトリが分かれ、データを格納
  - global：システムカタログなどの管理用オブジェクトのデータ
  - pg_xlog：WALセグメントファイル
  - その他
    - 各種設定ファイル
    - 障害対応で必要なもの

構築、運用管理、性能、障害対応全てに
関わるため、詳細に理解しておく
各ファイルに対し考慮点が多数あるため
まずはここから解説



© LPI-Japan 2011. All rights reserved.

各ファイルに対する考慮点

アシスト 「お客様の最高」のために

OSS-DB

■安定性や性能を考慮した構築ポイント

①各ファイルの配置先はどう決めるか

②テーブルデータを格納する物理ファイル

③インデックスデータを格納する物理ファイル

④WALファイルに関する構築ポイント

■障害発生時の挙動

⑤容量不足時

⑥ディスク障害発生時

© LPI-Japan 2011. All rights reserved.

①物理ファイル配置先

I/O分散や、故障リスク分散の方法を理解しておく

- DBを作る際に、ユーザが位置を指定できるもの

内容	指定方法	デフォルト位置
データディレクトリ	initdb -D	$PGDATA
WALファイル出力先	initdb -X	$PGDATA/pg_xlog
ユーザデータ格納先	TABLESPACE機能	$PGDATA/base
ログファイル出力先	パラメータ	$PGDATA/pg_log
アーカイブ退避先	パラメータ	--

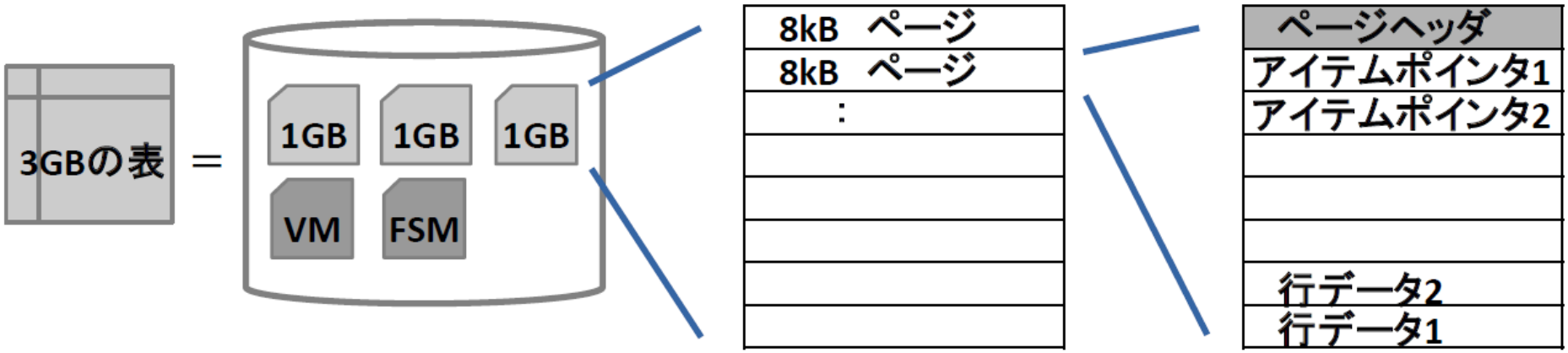
- データとWALは、同時に破損すると最新状態への復旧が困難になるため、WALファイル出力先を指定し、別のディスクとする
- テーブル、インデックスはTABLESAPCE機能を使用して配置先を分散してI/Oの効率化を図る
  - TABLESPACE機能で作成した表領域をデータベース単位で指定するか、もしくはオブジェクト作成時に個々に指定することで、データ格納先を変更
- RAIDによる保護も検討する

© LPI-Japan 2011. All rights reserved.

②テーブル構造

■物理ファイルの構造は詳細に問われる

- ファイル（ディレクトリ）構造や、ページ構造は必ず理解しておくこと
  - 1つのオブジェクト　＝ 1GB毎のセグメントファイル ＋ VM、FSM
  - セグメントファイルは8KB単位のページで構成される
- DDL文や¥dコマンド結果からテーブルサイズを見積もる



・1行のサイズは、行ヘッダ(アイテムポインタ)テーブルの場合28byteに
　データ型毎のサイズ(int：4、timestamp：8、文字型は4+平均バイト数)を加える
・ページヘッダ(24byte)とFILLFACTORを考慮し、ブロックに入る行数を算出
・テーブル行数を、1ブロックあたりの行数で除算し必要ブロック数を算出

© LPI-Japan 2011. All rights reserved.

③インデックス構造

物理ファイルの構造は詳細に問われる

- ファイル（ディレクトリ）構造や、ページ構造は必ず理解しておくこと
  - テーブルと同様の考え方で良いが、ヘッダ領域のサイズが異なる
  - インデックス対象フィールドのデータを保持するため、対象のデータ型に依存

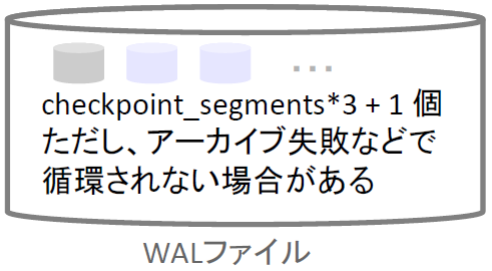
テーブル定義からインデックスサイズを見積もる

・1エントリ(ノード)あたりのサイズは、ノードヘッダ12byte+データ型毎のサイズ
・ページ内のエントリ数は、ページヘッダ40byteを考慮し、
(8192-40)/1エントリあたりのサイズ
・必要ページ数は、テーブル件数/ページ内のエントリ数
・インデックスサイズは、8192byte*必要ページ数

© LPI-Japan 2011. All rights reserved.

平常時の動作と、リカバリ時の使用イメージを正確にしておく

- WALセグメントファイルサイズは16MBで、checkpoint_segmentsで指定した値によりファイル数の下限が変動する
  - WALファイルはチェックポイントが済めば不要
  - 不要なWALは自動で消去される
  - ただし、リカバリ時はベースバックアップ以降のWALが必要となる



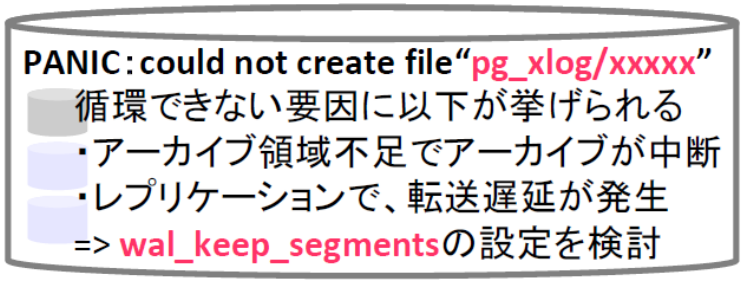
- WALファイルが蓄積されるケース
  - アーカイブモードで、アーカイブ出力先の容量不足によりコピーが完了しない場合
  - レプリケーションで、未転送のWALが消失しないように以下設定がされた場合
    - wal_keep_segments
    - レプリケーション・スロット(～9.4)を作成済で転送遅延している

© LPI-Japan 2011. All rights reserved.

⑤容量不足時

■データ、WAL領域、それぞれで発生する問題を整理

- 容量不足により各ファイルへの書き込みが失敗すると異なる挙動となる
- データファイル
  - データ破壊を引き起こす可能性がある
  - ディスク容量不足に対して、以下の対処は誤り
    - SELECT処理のみのシステムであり、そのまま運用する
    - VACUUM (FULL)を実行する
    - $PGDATA/base配下を手動で×××
- WALファイル
  - 領域が不足すると、データベースクラスタが停止する



© LPI-Japan 2011. All rights reserved.

■データ（テーブルやインデックス）、WALの物理ディスク障害

- データの復旧ではリカバリが必要
  - H/Wを正常に復旧した後に
  - 最新状態まで復旧したい場合は、PITR（アーカイブ運用が必須）
  - 論理バックアップやコールドバックアップでは、バックアップ取得時点まで復旧
- インデックスデータの場合
  - データの破損ではないため、まずはインデックス再作成を検討する
  - システムテーブルのインデックスの場合は注意が必要
  - インデックス再作成は、システムに与える影響を考慮して検討
- WALファイル
  - 起動時に読み込まれるため、WALが破損していると起動できない
  - pg_resetxlogを使用してWALのリセットを行う
  - または、正常なトランザクション位置（アーカイブ化されたもの）を指定してPITRを行う

WALのリセットは、DB停止中に、
$ pg_resetxlog -f -x [redacted] $PGDATA

pg_clog 配下のXIDを確認し、
次のXIDを指定してWALをリセット
例） pg_clog配下が0011なら、1を加え
後ろに0を5つ付けた 001200000

© LPI-Japan 2011. All rights reserved.

構築時の設定-メモリ

shared_buffersの初期設定

- ディスクから読み取ったデータをキャッシュして、ユーザ要求に高速に応答
- チューニングポイント　shared_buffersパラメータ
  - 初期設定値は、物理メモリの 25%-40% 程度とする
  - 頻繁にアクセスされるデータがキャッシュできるようにサイズを調整

- キャッシュヒット率の確認
  - キャッシュヒット率 = メモリ上でヒットしたデータ/必要とされたデータ
    - pg_stat_databaseを参照
    - ヒットしたデータ　：　blks_hit
    - 必要とされたデータ　：　blks_hit + blks_read

© LPI-Japan 2011. All rights reserved.

WALバッファの初期設定

- WALファイルのI/Oをシーケンシャルにするため、メモリ上に変更をキャッシュ
- チューニングポイント
  - WALフラッシュのタイミングを知り、回数を減らすようWALバッファのサイズを調整
  - バージョン9.x以降では自動調整されるが、自動調整の上限は16MB
  - 実際にはWALセグメントサイズの倍（32MB）程度とするのが望ましい
- WALフラッシュのタイミング
  - トランザクションがコミットされたとき
  - wal_writer_dilay時間が経過したとき　デフォルト3秒
  - WALバッファが一杯になったとき

© LPI-Japan 2011. All rights reserved.

loggerプロセス

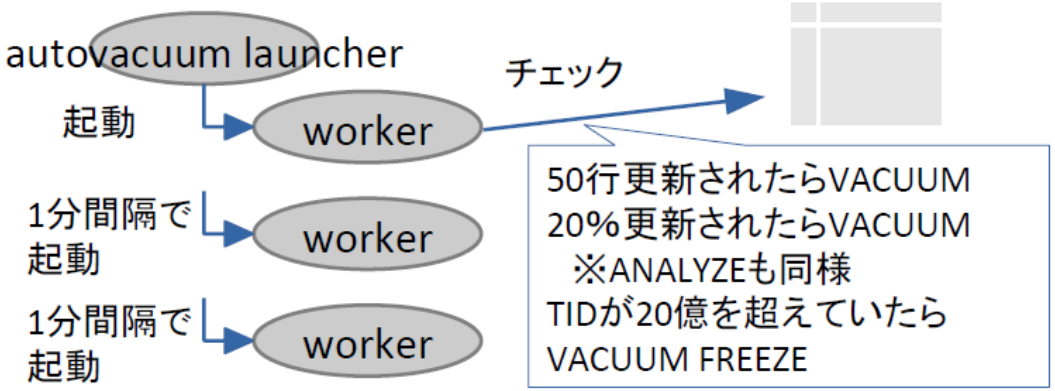
- デフォルトは無効だが、有効化しておくべき
  - logging_collector = on
  - log_destination = `ログ出力先ディレクトリ(デフォルトでOK)`
  - ログ循環設定
  - ログ出力設定
- ログに記録される情報を大別(特に障害対応に必要なもの)

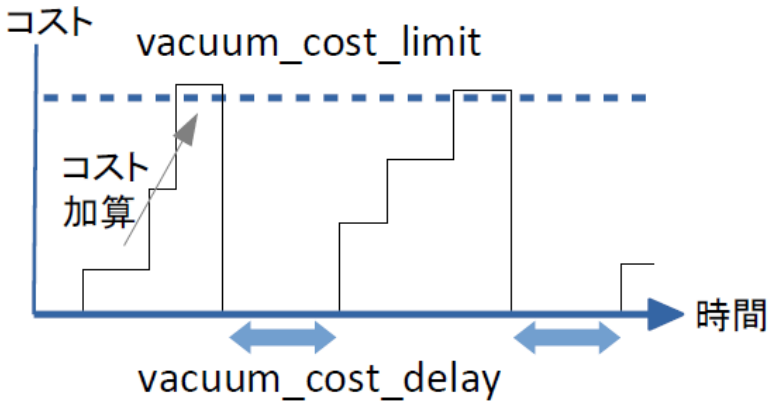
エラーレベル	内容
PANIC	サーバが停止している
FATAL	セッションが切断されている(他のセッションは正常)
ERROR	該当の処理が失敗し、セッションは残っている

© LPI-Japan 2011. All rights reserved.

autovacuum_launcherプロセス

- 自動VACUUMを活用
  - autovaccuum = on
- VACUUMの負荷はシステム全体の性能に多少の影響を与える
  - 自動VACUUMのタイミングを理解しておく
  - autovaccuum_vacuum_threshold / autovacuum_vacuum_scalefactor
  - 性能影響を抑えるために、ゆるやかにVACUUMさせる(遅延VACUUM)





© LPI-Japan 2011. All rights reserved.

■アーカイブ運用

- 障害発生直前にコミットされたデータまで復旧したい場合、必須の設定
  - archive_mode = on
  - archive_command = `WALファイルをコピーするためのコマンド(CPやSCP)`
  - wal_level = `archive`

■レプリケーション構成

- wal sender およびwal recieverプロセスが起動
  - マスター側で、max_wal_sendersで指定した数のスタンバイを持てる
- アーカイブ関連設定を変更する
  - archive_mode = on
  - wal_level = `hot_standby`
- スタンバイ側では、ホットスタンバイ有効化(リカバリ中も参照を受け付ける)
  - hot_standby = on

© LPI-Japan 2011. All rights reserved.

セキュリティ関連設定

- PostgreSQLでできることの整理
  - ①データ暗号化に関して
  - ②通信経路暗号化
  - ③監査情報の取得

- 試験ではPostgreSQLでできることのみでなく、通常対策すべき課題、OSや他のソフトウェア機能での実装を検討する内容も問われる

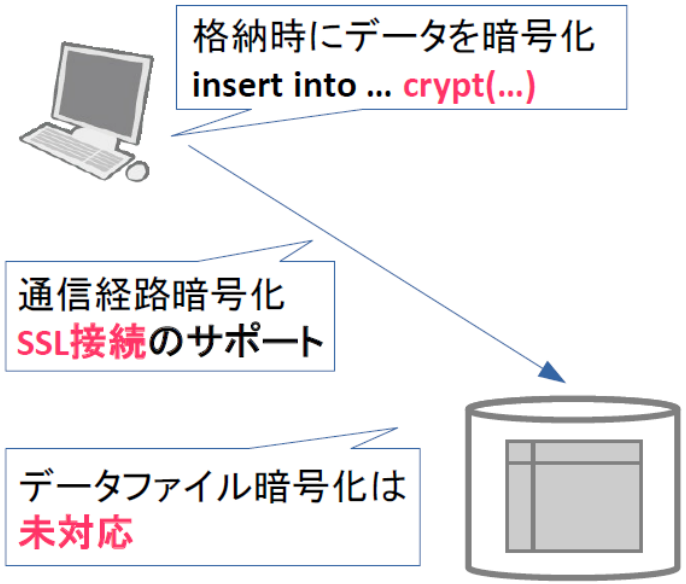
© LPI-Japan 2011. All rights reserved.

■暗号化で検討される一般的な考慮点

- 格納前の暗号化　：　個人情報を含む列のみを暗号化するなど、AP側の実装
- 通信経路暗号化　：　実行されるSQL文の盗聴を防ぐ（次項で解説）
- ファイル暗号化　　：　バックアップの持ち出しなどから保護

■PostgreSQLでの対応

- PostgreSQLでは、データ格納時にpgcrypto関数を使用して列単位の暗号化
- データファイル暗号化はできず、暗号化機能を持ったストレージ等を検討

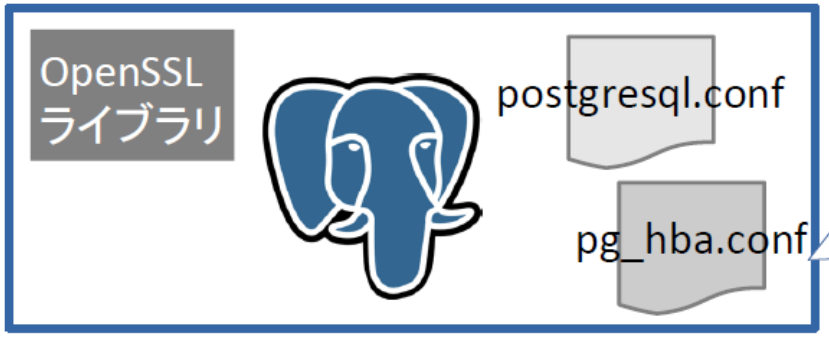


© LPI-Japan 2011. All rights reserved.

②通信経路暗号化

■クライアント-サーバ間のSSL通信設定を理解しておく

- PostgreSQLのインストール時
  - configure実行時に--with opensslオプションを追加（OpenSSLライブラリ必須）
  - RPM版によるインストールでは本設定も有効化されている
- データベースクラスタの設定
  - ssl = on
  - SSL用サーバ証明書の設定や、秘密鍵を含む設定ファイルを用意し指定
- 認証設定
  - クライアント毎にSSL通信を強制するかどうかを指定可能
  - 認証設定ファイル pg_hba.confファイルにtype = hostsslを指定



pg_hba.confの書式に従いアクセス制御

type	database	user	host	method
hostssl	db1	usr1	xx.xx.xx.xx	md5
hostnossl	db2	usr2	xx.xx.xx.xx	md5

© LPI-Japan 2011. All rights reserved.

③監査情報の取得

■監査で検討される一般的な考慮点

- 実行された処理を記録
  - 実行されたSQL文の記録
  - 失敗したSQL文の記録 ： アドホックなSQLを実行する管理者からの処理
  - 特定のテーブルに対する処理の記録 ： 個人情報を含むテーブルのみ監視
- ログイン/ログアウトを記録
  - DBAアカウントによる操作の記録 ： 申請に基づく操作が行われているか
- 専用の監査ログファイルの作成
  - DBA/セキュリティ担当者の職務分掌

■PostgreSQLでの対応

- log_statementの指定によるSQL文を記録
  - ユーザ単位で指定可能
  - テーブル単位やSELECTのみは不可
- log_(dis) connectionの指定による
  ログイン/ログアウトの記録

log_statementの設定

none	出力無し(デフォルト)
ddl	実行されたDDL文のみ記録
mod	実行されたDDL文とDML文を記録 ※COPY FROや、PREPARE、EXPLAIN ANALYZEの該当するSQL文を含む
all	実行された全SQLを記録

© LPI-Japan 2011. All rights reserved.

広範囲な試験範囲をDB管理者の業務に当てはめて理解

- サーバ構築
  - OSの設定～PostgreSQLのインストール
- 初期設定
  - サーバサイジング
  - パラメータ設定
  - セキュリティ
- 監視
- メンテナンス
  - オブジェクトのメンテナンス
  - ユーザのメンテナンス
  - 起動・停止
- チューニング
  - プロセスの詳細動作（DBチューニング）
  - SQLチューニング
- 障害復旧

© LPI-Japan 2011. All rights reserved.

運用管理

- 監視
  - 容量監視
  - プロセス監視
  - サーバーログ監視
  - パフォーマンス監視
- オブジェクトのメンテナンス
  - VACUUM (自動VACUUMを含む)
  - ANALYZE
  - HOT と FILLFACTOR
  - REINDEX
- ユーザ (セッション) の管理
  - 長時間実行しているSQL、ロック
  - ユーザ設定

© LPI-Japan 2011. All rights reserved.

容量監視

アシスト 「お客様の最高」のために

OSS-DB

ディスク容量、データベース、オブジェクト等のサイズを監視

- ディスク容量はOSコマンド(df 、du -s など)で監視
- 特に以下の領域に注意
  - データファイル
  - WALファイル
  - アーカイブ領域(ローカルディスクに配置している場合)

- データベースやオブジェクトサイズは関数で確認
  - pg_database_size('database')
  - pg_relation_size('object')
  - pg_total_relation_size('table')

© LPI-Japan 2011. All rights reserved.

死活監視

サーバの死活監視はプロセス監視またはクライアント接続で確認する

- OSコマンド(ps -ef など)で監視
  - postgresプロセスのPIDを確認
  - $PGDATA/postmaster.pidファイルに記録されたPIDと一致
  - 他のプロセスは、postgresプロセスが自動的に再起動する
- SQLによる死活監視
  - 数分間隔で SELECT 1; などの単純なSQLを実行
- 専用コマンドによる死活監視
  - pg_isreadyコマンド(9.3～)
  - 管理コマンドとしてインストールされ、死活監視に利用

正常時	接続不可の場合
$ pg_isready /tmp:5432 – accepting connections $ echo $? 0	$ pg_isready -h localhost -p 5433 localhost:5433 - rejecting connections $ echo $? 1 : 起動中などで接続を拒否 2 : 無応答 3 : pg_isreadyの実行に失敗

© LPI-Japan 2011. All rights reserved.

パフォーマンス監視

- OSリソースやシステムカタログの監視、遅いSQLを監視
  - OSリソースの監視
    - sar、top、vmstat、mpstat、iostatなど
    - 平常時と比較し、CPU使用率やメモリ使用率が高くないか
  - システムカタログの監視
    - pg_stat_database ： キャッシュヒット率
    - pg_stat_bgwriter ： チェックポイント間隔
    - pg_stat_all_tables ： キャッシュヒット率やHOT更新、インデックススキャンの割合
    - pg_statio_all_tables ： 同上（ブロック数で表示）
    - pg_stat_activity ： ユーザセッション毎に実行されているSQLの情報を格納
    - pg_locks ： pg_stat_activityとjoinし、他セッションをブロックしているSQLを特定
  - スロークエリの検出
    - ログ監視設定で解説

© LPI-Japan 2011. All rights reserved.

サーバログ監視

サーバログの何を監視するか

- エラーラベルの監視　log_min_messagesのエラーラベル
  - INFO、NOTICE、WARNING、ERROR、LOG、FATAL、PANIC
  - 重要度の高いものは以下

エラーレベル	内容
PANIC	サーバが停止している
FATAL	セッションが切断されている(他のセッションは正常)
ERROR	該当の処理が失敗し、セッションは残っている

- 閾値超過したSQLの監視
  - log_min_error_statement ： 指定のエラーラベルに対してSQLを記録する
  - log_min_duration_statement ： 実行に長時間要したSQLの特定
  - log_lock_waits / deadlock_timeout ： ロック獲得に要した時間がtimeoutを超過
- その他指定イベントの監視
  - log_checkpoints
  - log_(dis)connections

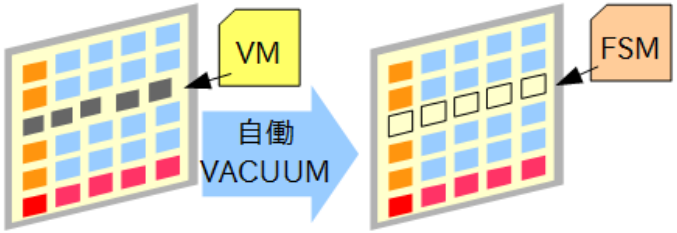
© LPI-Japan 2011. All rights reserved.

テーブルの肥大化を抑制する

- 適切なVACUUMの実行
  - VACUUMの種類と内容

種類	内容
(通常の)VACUUM	不要行をFree Space Mapに登録し、再利用可能にする
VACUUM FULL	表の再作成を行い、不要行を詰めて物理ファイルの縮小を行う ※一時的に表サイズの2倍の領域を使用するため、ディスク不足時の領域確保には使えない
VACUUM FREEZE	トランザクションID周回問題への対処

  - VACUUMの動作イメージ
    - Visibrity Mapから不要行を検索
    - 使用可能領域としてFree Space Mapに記録

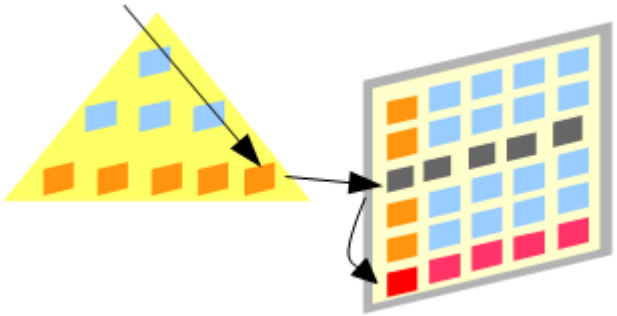


  - VACUUMが適切に実行されることで、一定サイズ以上には肥大しない

© LPI-Japan 2011. All rights reserved.

■テーブルの肥大化を抑制する

- HOTによる更新とFILFACTOR
  - HOTは行データが更新されてもインデックス更新をしない仕組み
  - 同一ページ内に更新後のデータが格納できる場合に有効となる
    - テーブル作成時にFILLFACTORを設定
  - かつ、不要となった更新前のデータは、VACUUMとは無関係に再利用可能となる



・更新により行の物理的な位置が変わるが、
HOT更新であれば索引はそのまま
・テーブル側で実データの位置にリンクを貼る
・不要行は再利用可能となる

- 不要な索引の削除
  - インデックスが張られた列が更新された場合、インデックスにも更新が必要
  - 不要な索引があるとHOTが有効にならないということ
  - pg_stat_all_indexesなどで使用されていないインデックスを特定

© LPI-Japan 2011. All rights reserved.

インデックスの肥大化を抑制する

- インデックスは、テーブルの更新に合わせて自動で書き換えられる
- 更新前のエントリは不要とわかってるが、再利用されるのはページ単位
- 以下を比較して、ページ数が非常に大きい場合に再作成を検討
  - pg_class の reltuples ： インデックスエントリの行数
  - relpages ： インデックスのページ数
- REINDEX
  - 専用コマンドが提供されているが、既存インデックスおよびテーブルを排他ロック
  - Index Scanを待機させてしまうため、影響が大きい
- DROP INDEX/CREATE INDEX
  - DROP時に一時的に元テーブルを排他ロック
  - CREATE中はテーブルへのアクセスはSeq Scanを強制
- CREATE INDEX CONCURRENTRY
  - 既存インデックスと重複する別名のINDEXを作成可能
  - 作成時間は長くなるが、上記のようなロックの影響が少ない

© LPI-Japan 2011. All rights reserved.

DB管理者の業務

アシスト 「お客様の最高」のために

OSS-DB

広範囲な試験範囲をDB管理者の業務に当てはめて理解

- サーバ構築
  - OSの設定～PostgreSQLのインストール
- 初期設定
  - サーバサイジング
  - パラメータ設定
  - セキュリティ
- 監視
- メンテナンス
  - オブジェクトのメンテナンス
  - ユーザのメンテナンス
  - 起動・停止
- チューニング
  - プロセスの詳細動作（DBチューニング）
  - SQLチューニング
- 障害復旧

© LPI-Japan 2011. All rights reserved.

チューニングについて

DBチューニングとSQLチューニング

- DBチューニング
  - 構築段階からある程度の設定が可能で、大まかな設定でも大きな効果がでる
  - パラメータチューニングが主
  - 評価指標として、システム全体でどの程度の処理が可能かを表すTPSなど
- SQLチューニング
  - 問題のあるSQLが特定し、その実行計画を調整することで劇的な効果を期待
  - 索引を使用しているかどうか、パラメータ設定による実行計画の強制
  - 評価指標として、対象SQLのレスポンスタイムなど
- 双方が与える影響
  - 高負荷な1つのSQLが改善することでシステム全体が最適化され、TPSも向上する可能性もある
  - SQLチューニングのために索引を作成したことで、他の処理に悪影響を及ぼす可能性もある

チューニングに関する詳細は http://www.slideshare.net/kkida85/postgre-sql-slideshare でも公開

© LPI-Japan 2011. All rights reserved.

pgbench

- OLTP系のベンチマークツールで、TPSの測定が可能
  - 初期化モード ： ベンチマークに必要なテーブルとデータを生成

テーブル名	スケールファクターあたりの行数
pgbench_accounts	1,000,000行
pgbench_tellers	10行
pgbench_branches	1行
pgbench_history	0行

$ pgbench –i –s 100 –f 90

-i　初期化モード
-s　スケールファクタを指定
pgbench_accountsはs*1,000,000行となる
-f　フィルファクタを指定
ページの余裕率を指定し、実環境に近くなる

  - 計測モード ： 引数で指定した回数または時間だけトランザクションを実行

処理内容
pgbench_accountsを1件更新
pgbench_accountsから1件参照
pgbench_tellersを1件更新
pgbench_branchesを1件更新
pgbench_historyに1件挿入

$ pgbench –c 10 –j 2 –T 300

-c　クライアント数　-j　スレッド数
クライアント数はスレッド数の倍数を指定

-Tまたはt　実行時間またはトランザクション数
いずれもTPSを計測するため、結果はほぼ不変

その他
-S 参照のみ -C セッションを都度生成　など

© LPI-Japan 2011. All rights reserved.

稼動統計情報の参照

- 稼動統計情報
  - stats collectorが収集するDBやオブジェクトに対するアクセスの実績を保持
  - 500ms毎に取得、更新されている
  - pg_stats_＊　や、pg_statsio_＊　は全て稼動統計情報
  - 実行時統計情報やアクセス統計情報と記される場合もある
- 統計情報を参照してチューニングの要否を判断
  - shared_buffersのチューニング
    →pg_stats_databaseからキャッシュヒット率を計算
  - チェックポイント間隔のチューニング
    →pg_stats_bgwriterから checkpoints_req >> checkpoint_timed の場合

サーバログに記録されたイベントを確認

- チェックポイント間隔
  - log_checkpoint や　checkpoint_warning

© LPI-Japan 2011. All rights reserved.

効果が大きいが信頼性を犠牲にするもの

- fsync (デフォルト on)
  - onでは、WALの同期書き込みを保障するため、データの一貫性を保障できる
  - offにすると、同期書き込み命令は送るが、本当に書かれたかどうかは未保障
- full_page_writes (デフォルト on)
  - onでは、チェックポイント直後に更新されたページ全体をWALに書き込む
  - offにすると、チェックポイント後も変更箇所のみをWALに書き込む

障害時に一部のトランザクションの結果が失われるもの

- synchronous_commit (デフォルト on)
  - onでは、コミットが完了する前にWAL書き込みが完了したことを保障
  - offでは、最大でwal_writer_delay*3秒後にトランザクション結果が保障される
  - fsyncとの違いは、一定時間経過後はディスクに正しく書かれるため、バックアップからの復旧は可能である点

© LPI-Japan 2011. All rights reserved.

その他のWAL関連設定

- commit_delay
  - 同時に多数のトランザクションが実行される場合に効果が期待できる
  - コミット準備が完了したら、commit_delay秒間だけ他のトランザクションが準備完了するまで待機し、複数トランザクションをまとめてコミットする
  - 待機するのはcommit_siblings値以上のトランザクションが実行されている場合
  - 他のトランザクションが完了しない場合、無駄に待機することになる
- wal_sync_method
  - WALの同期書き込みで使用するシステムコールを選択
  - 最適な書き込み方式を選択できるものの、効果は限定的
  - pg_test_fsyncを使用して、最適な書き込み方式を調査できる

© LPI-Japan 2011. All rights reserved.

DB管理者の業務

広範囲な試験範囲をDB管理者の業務に当てはめて理解

- サーバ構築
  - OSの設定～PostgreSQLのインストール
- 初期設定
  - サーバサイジング
  - パラメータ設定
  - セキュリティ
- 監視
- メンテナンス
  - オブジェクトのメンテナンス
  - ユーザのメンテナンス
  - 起動・停止
- チューニング
  - プロセスの詳細動作（DBチューニング）
  - SQLチューニング
- 障害復旧

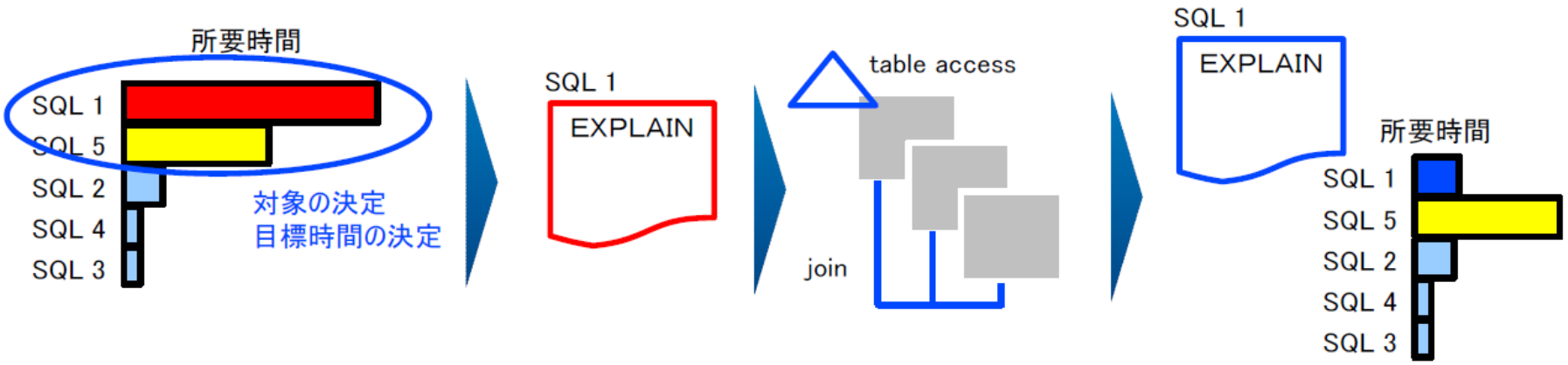
© LPI-Japan 2011. All rights reserved.

SQLチューニングの考え方

- 実行時間が長いSQLを対象に、レスポンス要件を満たすように改善

SQLチューニングのステップ

- どのSQLをどこまで早くするか
- 現状を確認する
- どうやって早くするか
- 効果を測定する



© LPI-Japan 2011. All rights reserved.

対象SQLの決定

- 統計情報から実行時間の長いSQLを特定
  - pg_stat_activity
  - pg_stat_statements (contribツール)
- エラーログへの出力から実行時間の長いSQLを確認
  - log_min_duration_statementsパラメータ
  - auto_explain (contribツール)
- キャッシュヒット率やアクセスブロック数を基準にする
  - キャッシュヒット率の監視で平時より低い値が記録された
  - 1行を対象にした検索なのに大量ブロックにアクセスしている
    - pg_stat_all_tables/pg_statio_all_tables

© LPI-Japan 2011. All rights reserved.

現状を正しく理解する

■実行計画を読み解く

- 実行計画：該当SQLを実行すると、どのような経路でデータを取り出すか
- クエリの先頭にEXPLAIN　または　EXPLAIN ANALYZEを付ける
  - EXPLAIN　：　統計情報から予想される実行計画を表示
  - EXPLAIN ANALYZE　：　クエリを実行し、実行された実行計画を表示
    ※更新処理の場合、begin;　でトランザクションを明示的に開始しておくこと

■実行計画の書式

```
postgres=# EXPLAIN SELECT b.logno,e.empname,b.log_text FROM log_master m,log_body b,emp e
           WHERE b.logno=m.logno AND e.empno=m.empno AND m.status = 'WI';

QUERY PLAN
------------------------------------------------------------------------------------------
Hash Join (cost=79.00..1754.21 rows=10000 width=66)
 Hash Cond: (m.empno = e.empno)
  -> Merge Join (cost=0.00..1525.21 rows=10000 width=24)
      Merge Cond: (m.logno = b.logno)
        -> Index Scan using log_master_pkey on log_master m (cost=0.00..6907.33 rows=31373 width=8)
             Filter: (status = 'WI'::bpchar)
        -> Index Scan using log_body_pkey on log_body b (cost=0.00..707.27 rows=10000 width=20)
  -> Hash (cost=54.00..54.00 rows=2000 width=50)
        -> Seq Scan on emp e (cost=0.00..54.00 rows=2000 width=50)
```

計画ツリー

計画タイプ(プラン)　コストの推定値　推定行数や列幅

© LPI-Japan 2011. All rights reserved.

実行計画－計画タイプ

■計画タイプから、意図した方式が選択されているかを確認

- 計画タイプ：計画ノードの先頭にどの処理をどの方式で行うか記載
- 代表的なスキャンと結合の計画タイプ（他にソートや集合のタイプがある）

計画タイプ	内容
Seq Scan	全表スキャン：表内の大量の行にアクセスする場合に有効
Index Scan	索引スキャン：表内のごく一部の行にのみアクセスする場合に有効
Index Only Scan	索引のリーフブロックのみにアクセスするスキャン(表へのアクセスをスキップ)
Bitmap Index Scan	ビットマップ使用した索引スキャン
Bitmap Heap Scan	ビットマップスキャンで表にアクセス
Function Scan	ファンクションの実行結果に対するスキャン
Nested Loop	ネステッドループ結合：片方の表のうち、ごく少数の結果を条件として他方の表からデータを取得
Merge Join	マージ結合：両方の表の行数が多い場合、ソートし、上から順に値を比較して該当行を抽出
Hash Join	ハッシュ結合：表の行数に差があり、かつ小さい表の重複が少ない場合に有効

© LPI-Japan 2011. All rights reserved.

実行計画-推定行数

行数の推定が間違っていると、正しい計画タイプとならない

- プランナ統計情報
  - ANALYZE統計情報やテーブル/カラム統計情報とも呼ばれる
  - ANALYZEによってサンプリングされた実データから得られた統計
- 行推定の方法
  - カラム統計情報はpg_statisticに格納されており、読み取りやすいように加工されたpg_statsを参照
  - テーブルの行数から、nullの行数を除外し、個別値・頻出値の組　または ヒストグラム境界値から検索条件に合致する行数を推定する

pg_statsの列	内容
null_frac	nullの割合を1以下の小数で記録、1ならnullが100%
n_distinct	個別値の具体的な数　男・女なら2 名前など重複が少ない場合、(-個別値の数/行数)で記録
most_common_vals most_common_freqs	個別値のうち、登場回数が多いものを配列として記録 valsとfreqsは対応して、freqsはその登場割合を記録
histogram_bounds	全行を均等分割した境界値を記録 n_distinctがマイナス値(個別値の数が多)の場合に記録される

© LPI-Japan 2011. All rights reserved.

行推定の例

- Most Common Valsが使える場合
  - 算出した割合とpg_classの行数で推定行数が計算できる
  - SeqScanとなる可能性が高い

会員表の”性別=男”を検索

個別値=男、女の「2」通り
nullの割合=0.15
MCV=(男,女)　割合=(0.6,0.4)

=>非null値のうち、MCVから割合を計算
(1-0.15)*0.6=0.51→ 約51%の行が”男”

- ヒストグラムの分布から計算
  - 個別値はマイナス値であり、上記のような割合からの計算ができない
  - ヒストグラムの境界値で分布を予測（PostgreSQLが知っているのは緑の情報だけ。面積が等しい。）

会員表の”入会日=xxxx/xx/xx”を検索

個別値= -0.015　(1500通り/10万行)
ヒストグラム
＝分布の面積を
等分するような
境界値で記録

=>データの数%のみ該当でIndexScan

© LPI-Japan 2011. All rights reserved.

■コストが適正かどうか

- プランナコスト定数と統計から読み取った相対値
  - 左が1行目を取得するまでの初動コスト、右が全行取得するコスト
  - 初動コストがかかるかどうかは、計画タイプによる
    （例：BitmapScanではテーブルへのアクセス前にビットマップを作成）
- 問題があるSQLでは、期待する結果は1行にもかかわらず、非常に大きいコストがかかっている等、不可解な点がある
  - 統計情報が古い、列中で特別な値であり、頻出値から漏れるなど

© LPI-Japan 2011. All rights reserved.

問題箇所に対して有効と思われる対策を試す

- 計画タイプの問題
  - Index Scanを期待したがSeq Scanになっている
    →索引は作成されているか
    →統計情報は問題ないか、最新になっているか
  - Hash Joinを期待したがNested Loopになっている
    →セッション単位で、enable_hashjoinパラメータをoffにする
  - Index Only Scanを期待したが、Seq Scanになっている
    →インデックス列のcountなどでは、Index Onlyによる大幅な高速化が期待される
    →ただし、Visibrity Map を使用するため、VACUUM直後であることが必須

© LPI-Japan 2011. All rights reserved.

行推定の問題

- 推定された行数が現実と乖離している場合
  - ANALYZEによりカラム統計情報を更新した後、再実行
    →カラム統計情報が古くなっている可能性があるため
    →バッチで大量更新した直後などが該当
  - カラム統計情報が最新である場合、列値が特殊であるケースも考える
    →サンプリングから漏れている可能性
    →default_statistics_targetパラメータ　ヒストグラムの境界値の個数を指定
    　ALTER TABLE ・・・ SET STATISTICSで、列ごとに調整も可能

© LPI-Japan 2011. All rights reserved.

その他のチューニング

- ソート
  - trace_sortパラメータを有効化すると、ソート処理の詳細がログに記録される
  - ディスクソートが発生している場合、該当処理前にwok_memを増加
  - EXPLAIN ANALYZEでもディスクソートの発生を確認可能
- チェックポイント
  - pg_stat_bgwriterの監視や、log_checkpointによるチェックポイント頻度の監視
  - checkpoint_segmentsパラメータを増加することで対処
  - 遅延チェックポイント設定も可能
- バッチ処理
  - 大量データの変更を行い、かつ、障害時は処理の再実行が可能な前提
    →インデックスや制約の消去
    →アーカイブ運用の停止、wal_levelのminimal化、checkpoint_segmentsの増加
- メンテナンス処理
  - VACUUM、DDL文、pg_restore実行時は、maintenance_wok_memを増加

© LPI-Japan 2011. All rights reserved.

DB管理者の業務

広範囲な試験範囲をDB管理者の業務に当てはめて理解

- サーバ構築
  - OSの設定～PostgreSQLのインストール
- 初期設定
  - サーバサイジング
  - パラメータ設定
  - セキュリティ
- 監視
- メンテナンス
  - オブジェクトのメンテナンス
  - ユーザのメンテナンス
  - 起動・停止
- チューニング
  - プロセスの詳細動作（DBチューニング）
  - SQLチューニング
- 障害復旧

© LPI-Japan 2011. All rights reserved.

障害の種類

■障害の種類を整理しておく

- 電源障害、プロセス障害
  - メモリ上のデータが消失するが、起動時にリカバリされるため対処は必要ない
- ファイル破損
  - 正常なバックアップがあれば、復旧は可能
  - 限定的な障害の場合、少ない影響で復旧させることを検討

→エラーログから障害箇所を特定　PANIC:could not create file pg_xlog・・・

WALの破損(pg_xlog)であればpg_resetxlogの検討

データの破損(baseやテーブルスペース)では、zero_damaged_pages

システムデータの破損(global)では、シングルユーザモードで接続、REINDEX

インデックス(baseやテーブルスペース)の破損であれば、インデックス再作成

© LPI-Japan 2011. All rights reserved.

障害パターンによる対処

障害箇所が特定できたら

- WALの破損(pg_xlog)であればpg_resetxlogの検討
- データの破損(baseやテーブルスペース)では
  - oid2name(contrib)を使用して、対象のオブジェクトを特定
  - 論理バックアップからのリストアやzero_damaged_pagesモードでの救済を検討
- システムデータの破損(global)では
  - データーベースに接続できないなどの影響がある
  - シングルユーザモードで接続し、REINDEX SYSTEM を試す

  →システムデータへの索引が破損している可能性を考慮し、影響の少ないほうから
- インデックスの破損(baseやテーブルスペース)では
  - インデックス再作成
  - REINDEX、DROP/CREATE、CREATE INDEX CONCURRENTRYを検討

© LPI-Japan 2011. All rights reserved.

終わりに

OSS-DBの普及

- 現代の契約社会を支えるデータベース技術では、これまで商用製品が圧倒的なシェアを有していたが、近年の製品品質の向上や、国内での情報整備、サービス提供企業の存在から、急速にOSS化が進んでいる
- 商用/OSSを問わず様々なRDBMSの知識を持ち、データベースの構築、運用ができる、または顧客に最適なデータベースを提案できる技術者が求められている

OSS-DB資格の重要性

- 製品選択の観点から、体系的な知識を持った技術者の存在は重要であり、ベンダ資格がないPostgreSQLにとっては普及の起爆剤となる
- 前述の通り、製品自体が普及してきていることから、資格取得による個人のキャリアアップと、さらなる製品の普及促進の両面から非常に重要

© LPI-Japan 2011. All rights reserved.

ご清聴ありがとうございました。

■お問い合わせ■

© LPI-Japan 2011. All rights reserved.