PostgreSQL による共有ディスク・クラスタ PG-CALS の設計と実装

Design and Implementation of Shared Disk Cluster PG-CALS using PostgreSQL

中 山　陽 太 郎

要　約　PG-CALS は，米国 Unisys 社の協力の下でユニアデックス（株）が調査研究を行っている PostgreSQL による共有ディスク・クラスタを実現するデータベース管理システムである．PG-CALS では，共有ディスク・クラスタのアーキテクチャを採用し，アクティブ/アクティブ型クラスタとして，可用性と負荷分散，及びシステムの拡張性を同時に実現する．共有ディスク型のクラスタは，共有リソースへのアクセス競合により，一般にシステム性能を保証する実装は容易ではない．PG-CALS では，シェアードページキャッシュ機能を提供し，効率的な共有データリソースのアクセスを実現する．

本稿では，共有ディスク・クラスタ PG-CALS の実装技術について解説を行い，PG-CALS による効率的なトランザクション処理を検証する．

Abstract　PG-CALS is a database management system that realizes shared disk cluster for PostgreSQL, which is under investigation by Uniadex with co-operation of Unisys. It adopts the shared disk cluster architecture and realizes the availability, load balancing and system scalability at the same time by the Active-Active cluster feature. It is said that the shared disk cluster is difficult to implement to guarantee the system performance because of contentions of the shared resources. PG-CALS realizes effective shared data resource access by its shared page caching feature.

In this report, we describe the implementation technique of shared disk cluster PG-CALS and prove the efficient transaction processing by PG-CALS.

1. は　じ　め　に

企業システムにおけるデータベースの運用において，可用性と拡張性は重要な課題である．高可用性のクラスタリング技術として，アクティブ/スタンバイ構成によるフェイルオーバー型クラスタが一般的であるが，可用性と負荷分散を同時に実現することができない．アクティブ/アクティブ型の共有ディスク・クラスタでは，負荷分散と障害の局所化を可能とし，システムの拡張性と高可用性の点で優れている．しかし，共有データリソースに対するアクセス競合の問題があり，一般にシステム性能を保証する実装は容易ではない．

PG-CALS（Cache And Lock Service）は，米国 Unisys 社の協力の下でユニアデックス（株）が調査研究中の PostgreSQL[*1] [6]による共有ディスク・クラスタを実現するデータベース管理システム（DBMS）である．PG-CALS では，可用性と負荷分散，及びシステムの拡張性を同時に実現し，また，シェアードページキャッシュ機能によって，効率的な共有データリソースのアクセスを可能とする．

本稿では，共有ディスク・クラスタ PG-CALS の実装技術について解説を行う．また，PG-CALS で提供されるシェアードページキャッシュ機能によって，効率的な共有データリソ

ース・アクセスが可能であることを検証する．

2章で，データベースのクラスタリング技術の特徴を概観し，共有ディスク・クラスタとして必要なデータベース技術の拡張を説明する．3章で，PG-CALSの実装技術の概要について解説し，4章で性能検証について述べる．

2. 共有ディスク・クラスタのためのデータベース技術の拡張

2.1 共有ディスク・クラスタの特長

アクティブ/アクティブ型の共有ディスク・クラスタでは，可用性と負荷分散，及び拡張性を同時に実現することが可能である．可用性として，あるノードで障害が発生した場合，その障害は局所化され，クラスタ全体の停止は回避される．このシステム無停止による高可用性は，共有ディスク・クラスタの特長である．また，全てのノードは同じデータリソースを共有するため，ノードの追加（スケールアウト）によるシステム拡張に柔軟性をもつ．アクティブ/アクティブ型のクラスタを実現するものとしては，他に非共有型やレプリケーション方式によるものがある．表1に，クラスタ方式とそれぞれの特徴を示す．

共有ディスク・クラスタのデータベース製品としてはOracle RAC[5]，汎用機ではUnisysのXTC-UDS[7]などがある．

表1　クラスタ方式の特徴

クラスタ方式		特徴
アクティブ/スタンバイ型	フェイルオーバー型	アクティブ/スタンバイ方式．負荷分散やスケールアウトによる拡張性に対応できない．
アクティブ/アクティブ型	レプリケーション型	同期レプリケーションと非同期レプリケーションがある．検索処理の負荷分散に向くが，同期レプリケーションでは，更新性能が低下する．（例）pgpool，PGCluster（いずれもPostgreSQL対応）
	非共有型（シェアード・ナッシング）	並列処理性能やスケールアウトによる拡張性に優れる．データの区分によっては，複数ノードにまたがるデータアクセスの効率が低下する．（例）IBM DB2，PostgresForest®
	共有ディスク型（シェアード・エブリシング）	可用性や柔軟な拡張性の面で非共有型より優れているが，リソース共有によるホットスポットが生じ易い．（例）Oracle RAC

2.2 データベース技術の拡張

共有データベースを実現するDBMSは，それぞれのデータベース・インスタンスのキャッシュ上に共有するデータを読み込み，データの参照と更新を行う．そのため，ノード間におけるロックの排他制御と，各ノードでキャッシュされるデータの一貫性（Cache Coherence）を適切に管理する必要がある．ノード間でロックの管理を行うものは，グローバル・ロックマネージャ（Global Lock Manager）と呼ばれ，キャッシュ一貫性についても，データブロックのノード間での排他制御と共にそのステータスを管理する．

各ノード上のDBMSインスタンスは，ノード間において，ACID特性―原子性（atomicity），一貫性（consistency），独立性（isolation），持続性（durability）―を満たすトランザクション処理を実行する．そのため，トランザクション管理，ロック管理，バッファ管理，障害回復等のデータベース技術の拡張が必要である．

共有ディスク・クラスタの研究としては，障害回復手法として一般的であるWALプロトコルにおける効率的なアルゴリズムを提案したARIES[2]（Algorithm for Recovery and Isolation

Exploiting Semantics）に対する，共有ディスク・クラスタ拡張のための研究[3][4]がある．共有ディスク型 ARIES では，データ共有環境におけるデータベース技術の拡張と効率的なグローバルロック管理，キャッシュ一貫性管理について論じている．

3. PG-CALS の概要

PG-CALS は，複数ノードから共有するデータをシェアードページキャッシュ上に保持することによって，効率的な共有データリソースへのアクセスを可能とすることに特長がある．本章では，PG-CALS の実装技術について解説を行う．3.1 節で PG-CALS のシステム構成を示し，続く各節で，拡張が必要となる主要なデータベース技術について解説を行う．

3. 1 PG-CALS の構成

PG-CALS では，クラスタ環境において，異なるノード上の PostgreSQL インスタンスがデータベースを共有し，トランザクションの同時実行制御を行う．CALS とは，ノード間でキャッシュ共有を行う機能を拡張したグローバル・ロックマネージャである．PG-CALS の特長は，CALS によって管理されるシェアードページキャッシュにあり，これによって，効率的なキャッシュ一貫性を実現する．

図 1 に PG-CALS の構成を示す．OCFS2[*2] は，クラスタ・ファイルシステムである．CALS はソフトウェアであり，PostgreSQL と同一ノード上に配置することも可能である．

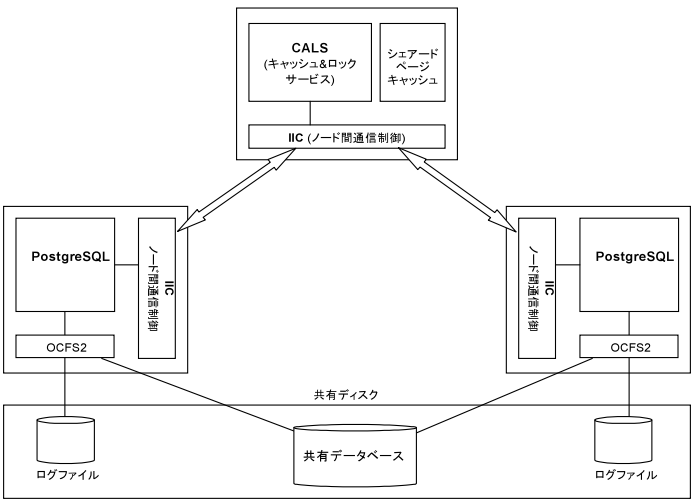

図 1　PG-CALS によるクラスタ構成例

各モジュールの機能は次のとおりである．

1）　拡張版 PostgreSQL

PostgreSQL の共有データベースアクセスを可能とするため，PostgreSQL のトランザクションマネージャ，ロックマネージャ，バッファマネージャ，IPC 管理，バックグラウンドプロセス（postmaster，bgwriter）等に CALS による共有リソース制御のための機能拡張を行う．

2） CALS（キャッシュアンドロック・サービス）

ノード間におけるロック排他制御，キャッシュ一貫性を管理し，シェアードページキャッシュ機能によって，共有データをキャッシュ上に保持する．

3） IIC（インター・インスタンス・コミュニケーション）

ノード間のメッセージ管理を行う．また，ハートビート監視を行う．

グローバル・ロックマネージャの実装として，一般に分散ロック管理と集中ロック管理とがあるが，PG-CALS では後者を採用している．集中ロック管理の利点として，ロックマネージャの 2 重化構成が可能であること，ロックマネージャを分離して，運用性が高められること，デッドロック処理の効率が良いことなどがある．逆に，懸案としては，ホットスポットとなり易いこと，障害単一点を回避するために冗長化の考慮が必要であることが挙げられる．

3. 2　グローバル・トランザクション管理

3. 2. 1　トランザクション情報管理

PostgreSQL では，トランザクション識別子によって，トランザクションのステータス管理やレコード（タプル）の可視性判定を行うため，CALS によるグローバルなトランザクション情報の管理が必要である．

CALS では，PostgreSQL インスタンスからトランザクション開始要求を受け付けると，トランザクション識別子（XID）の発行を行う．また同時に，グローバルに管理するアクティブトランザクションリストへの登録を行う．XID は，ノード間における一意性と順序を保障するために，CALS によるグローバル情報として管理される．トランザクション終了時も同様に CALS へ通知し，コミット状態の管理を行う．コミット状態の管理は，CALS の管理する COMMIT ログ（CLOG）に記録される．

3. 2. 2　ノード間ロック制御

PostgreSQL の内部的なロック管理の処理単位として，重量ロック（Heavy Weight Lock）と軽量ロック（Light Weight Lock）がある．重量ロックは，表 2 のロックモードをサポートし，テーブル及び行レベルでの排他制御を行う．重量ロックの継続期間（Duration）は，トランザクション実行の期間であり，デッドロック検出の対象となる．重量ロックは，内部的に暗黙に確立されるが，LOCK TABLE 文や SELECT FOR SHARE/UPDATE 文によって，明示的に取得することも可能である．また，軽量ロックは，内部的な共有管理リソースのアクセスのために使用され，ロックモードとして，共有モード（Shared）と排他モード（Exclusive）がある（表 3）．取得された軽量ロックは，対象リソースに対する処理が完了した時点で直ちに解放される．

バックエンドプロセス（postgres プロセス）から CALS へ要求されたロックは，CALS 内部で排他制御が行われる．ロック解除要求は，非同期処理が可能であり，ロック解除を要求したバックエンドプロセスは，CALS からのステータスを待つことなく処理を継続する．

CALS では，PostgreSQL のロックモードを実現しているが，内部的なロック制御は，独自のロック・オブジェクト管理に基づく実装による．

表 2　重量ロック競合マトリックス

認可済みロックモード	要求ロックモード							
	AS	RS	RX	SUX	S	SRX	X	AX
Access Share (AS)	T	T	T	T	T	T	T	F
Row Share (RS)	T	T	T	T	T	T	F	F
Row eXclusive (RX)	T	T	T	T	F	F	F	F
Share Update Exclusive (SUX)	T	T	T	F	F	F	F	F
Share (S)	T	T	F	F	T	F	F	F
Share Row eXclusive (SRX)	T	T	F	F	F	F	F	F
Exclusive (X)	T	F	F	F	F	F	F	F
Access Exclusive (AX)	F	F	F	F	F	F	F	F

T：認可（コンフリクトしない）
F：待ち（コンフリクトする）

表 3　軽量ロック競合マトリックス

認可済みロックモード	要求ロックモード	
	Shared	Exclusive
Shared	T	F
Exclusive	F	F

3. 2. 3　タプルの可視性判定

PostgreSQL では，MVCC（多版同時実行制御：Multi Version Concurrent Control）を実現し，レコード（タプル）の参照と更新は互いに待たされることなく，整合性の保障されたバージョンのデータを参照することが可能である．データの履歴管理は，追記型のデータベース構造によって実現され，タプルの可視性判定は，そのタプルの挿入または削除を行ったトランザクションが活動中かどうか，活動中でない場合はトランザクションの終了状態（COMMIT か ABORT か）によって決定される．例えば挿入を行ったトランザクション識別子の終了状態が COMMIT であれば，そのタプルは可視と判定され，ABORT であればロールバックと見なされ不可視と判定される．タプルの COMMIT 情報の設定は，タプルの更新後，最初にそのタプルを検索するトランザクションによって設定される．また，PostgreSQL のトランザクションの独立性レベル（Isolation Level）として，今回の実装では，Read Committed[*3] のみに対応する．

既に述べたように，CALS では，XID 生成，アクティブトランザクションリスト及びコミット情報を管理している．これらは，PostgreSQL のバックエンドプロセスから CALS へ要求があった都度，可視性判定の情報（スナップショット情報）を作成するデータとして返される．

3. 3　シェアードページキャッシュによる効率的なキャッシュ一貫性管理

PostgreSQL のブロックは，各ノード上の PostgreSQL インスタンスによって共有される．ノード間のキャッシュの一貫性を制御するため，CALS によるキャッシュ一貫性管理とキャッシング機能を提供する．キャッシュの一貫性が保障されない状態とは，複数のノードで同一ブロックを保持している場合，いずれかのノードでそのブロックが更新され，ノード間でのブロックのバージョンが異なる状態になることを言う．CALS によるキャッシュ一貫性の管理とは，各ノードのキャッシュの状態を管理し，要求があった場合に適切なステータスと最新のブロックを返すことである．

ノード間のブロック競合による排他制御の管理は次のとおりである．

1） 参照同士のブロック共有

ブロックの参照は，共有モードの軽量ロックにより，同時に複数のノード間で同一ブロックを参照することが可能である．対象ブロックが要求ノード上に存在せず，CALS 上に存在する場合は，CALS より転送される．CALS 上に存在しない場合は，要求を行ったノードのバックエンドプロセスが共有ディスクから読み込む．

2） 参照と更新，更新と更新による排他制御

ブロックの参照と更新，及び更新同士は，軽量ロックによって排他制御される．更新による排他ロックの期間は，ノードのバッファ上でブロックが変更される間である．また，更新されたブロックは，ロックの解放要求と共に CALS に転送されてキャッシュされる．

図 2 は，ブロックのロック処理とブロック転送を示したものである．あるノードからデータブロックを更新する場合，軽量ロックにより，排他モードで CALS に対してロック要求を行うが，CALS では，全てのノードのブロックの状態を管理しており，そのブロックが最新でなければ，最新のブロックをロック認可ステータスと共に要求元ノードへ転送する（図 2 の (1)）．また，更新されたブロックは，ロックの解放要求とともに CALS へ転送される（図 2 の (2)）．

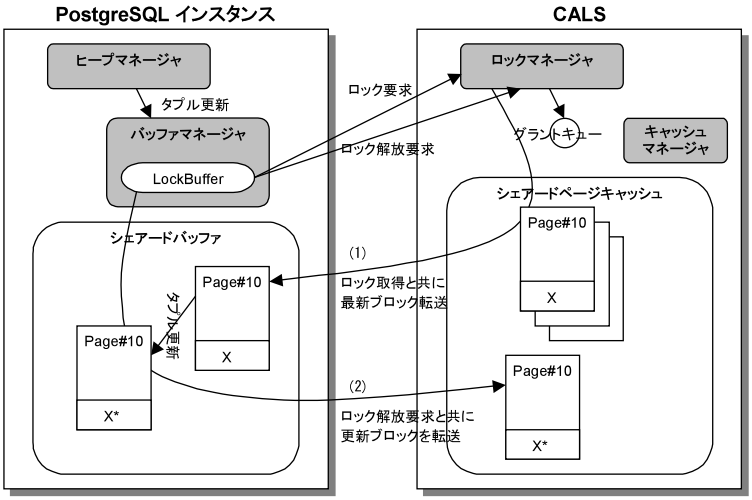

図 2　ブロックのロック処理とブロック転送

このように，PG-CALS では，CALS によるグローバル・ロックマネージャによって，ロック管理とキャッシュ一貫性を統合的に管理する仕組みを実装し，また，これらの連携によって，効率的なキャッシュ一貫性を実現している．あるノードで更新されたデータブロックは，ロック解放と同時に CALS に転送されて CALS のシェアードページキャッシュ上に保持され，ノード間のキャッシュ一貫性を効率的に保つための仕組みを実現する．

3. 4　障害回復

3. 4. 1　WAL 生成とログ順序番号

ログ（更新履歴）は，WAL（Write Ahead Logging）プロトコルに従ってノードごとに採られ，更新ブロックのイメージは，各ノードのログファイルへ書き込まれる．同一のデータブ

ロックが，異なるノード上のトランザクションから更新されるため，ログファイルから更新された順にデータを復元するためには，ノード間でログの更新順序を保障しなければならない．これは，全ノードのログからのデータベースの修復が必要となるグローバルな障害回復や，PITR（ポイントインタイムリカバリ）におけるノードごとのログのマージ処理で必要となる．

WAL 生成で付与されるログの順序番号 LSN（Log Sequence Number）は，ログファイルの物理アドレスを基準としたものであり，ノード間での順序を保障することはできない．ノード間でのログの順番を保障するために，Lamport の論理時計[1]に基づく論理時刻印（Logical Time Stamp）を適用する．データ更新を分散システムにおけるイベントとして，更新ログに論理時刻印を付与することで，ノード間のログの更新順序を保障することが可能である（図3）．

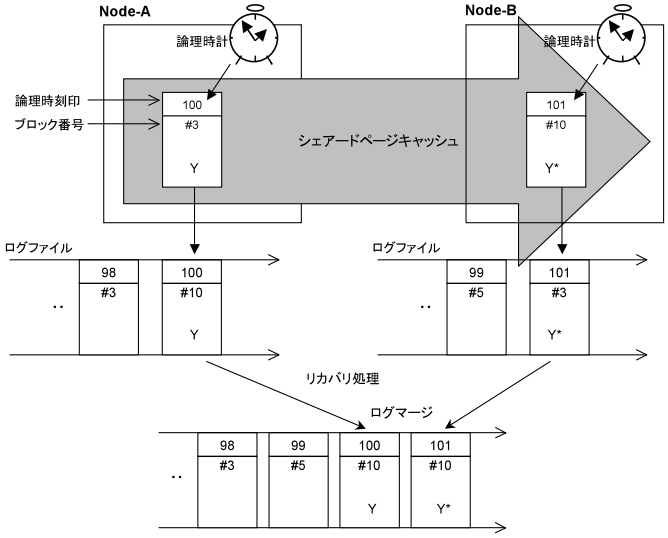

図3　ログ生成における論理時刻印の発行とログ・マージ

3.4.2　分散チェックポイント

チェックポイントとは，データベースのバッファプール上の更新ブロックをディスク上に反映させ*4，メモリとディスクを同期させる仕組みである．同期の時点は，チェックポイント・レコードによって記録され，障害回復時のリスタートで，ログの適用開始時点の決定に使用される．また，チェックポイント処理中に，データベースの更新を許すものをファジー・チェックポイント*5と呼ぶ．PostgreSQL ではファジー・チェックポイントを採用している．

分散データベースにおけるチェックポイント方式として，ノード間で同期を行うグローバル・チェックポイントと非同期チェックポイントがある．グローバル・チェックポイントは，全てのノードに対する同期を保障し，非同期チェックポイントは，ノードごとに非同期に実行される．通常，オンライン時は，非同期チェックポイントとし，障害回復や保守などで全ノードの同期が必要な場合は，グローバル・チェックポイントを使用する．

非同期チェックポイントでは，ノード間の書き出しの順序によって，実際の更新の順序とデータベースへの反映順序が逆転する現象が生ずる．つまり，最新のブロック・イメージが，古いブロック・イメージによって上書きされてしまう場合である．この問題を排除するため，対象ブロックを書き出す前に，対象ページが既に他のノードによって更新されていないか CALS

に確認する必要がある．

3. 4. 3　障害回復処理

PG-CALS における障害は，ノード障害（または DB インスタンス障害），CALS 障害，メディア障害（ディスク障害）の三つに大別される．ノード障害の場合，局所化として障害ノードのみ停止し，正常に稼働しているノードはそのまま稼働を続ける．但し，障害ノード上で稼働中であったトランザクションがロールバックされるまで，それらのトランザクションが保持するロックが残った状態となる．CALS 障害は，全ノードでアクセス停止となる．また，メディア障害（ディスク障害）では，各ノードのログを更新順序に従ってマージするための PITR（ポイント・イン・タイム・リカバリ）の拡張が必要となる．

図 4 は，ノード障害時のフェイルオーバー処理の例である．

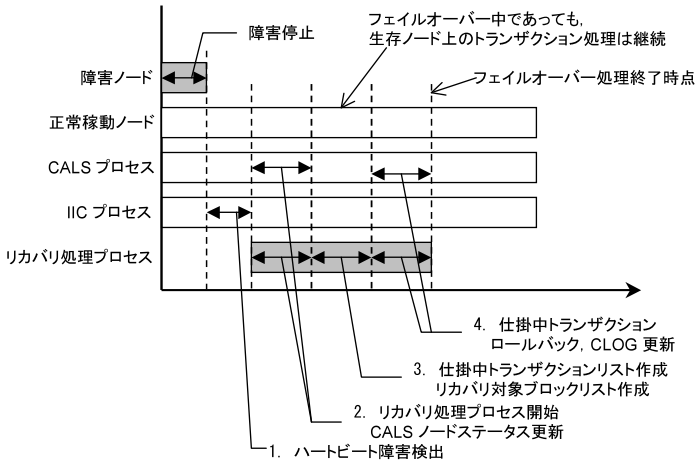

図 4　フェイルオーバー・タイムチャート

まず，ハートビートにより，障害が検知され（図 4 の 1.），CALS 上のノード情報が更新される（図 4 の 2.）．リカバリ処理プロセスは，CALS に問い合わせて，障害ノード上の仕掛り中トランザクションリストを作成し（図 4 の 3.），それらの保持するロックの解放とコミットログ情報（CLOG）の更新を行う（図 4 の 4.）．ノード障害による障害回復では，更新されたブロック・イメージは CALS 上にキャッシュされているため，障害ノードのログによる DB 回復処理は不要である．

CALS 障害の場合は，グローバルな障害回復（グローバル・クラッシュリカバリ）が必要であり，各ノードのログを使用し，適切なチェックポイント地点からデータベースのリカバリ処理を行う必要がある．また，CALS が冗長化構成に対応した場合は，CALS 障害による全面的なアクセス停止を排除することが可能となる．メディア障害については，各ノードのログをマージする点を除き，通常の PITR と同様の処理である．

4.　性能検証

4. 1　効率測定の実施

PG-CALS の効率検証として，PostgreSQL に付属している簡易版の TPC-B ベンチマークツ

ールである pgbench を使用する．効率測定で使用するソフトウェアと M/C の仕様は，表 4 のとおりである．また，PostgreSQL のデータベースは，ファイルシステム上に作成する必要があるため，クラスタ・ファイルシステムとして，Linux の標準である OCFS2 を使用する．

表 4　ソフトウェアと M/C 仕様

OS	Redhat Enterprise Linux 4.0 Update4
File System	OCFS2 1.2.5-1 (Cluster File System)
RDBMS	PostgreSQL 8.2.4
CPU	Xeon5130 2GHz×1
Memory	4GB
HDD	SAS146GB×2，EMC Ax150

図 5 に，効率測定のクラスタ構成を示す．CALS を PostgreSQL と同一ノード（Node-A）に配置する 2 台構成とする．

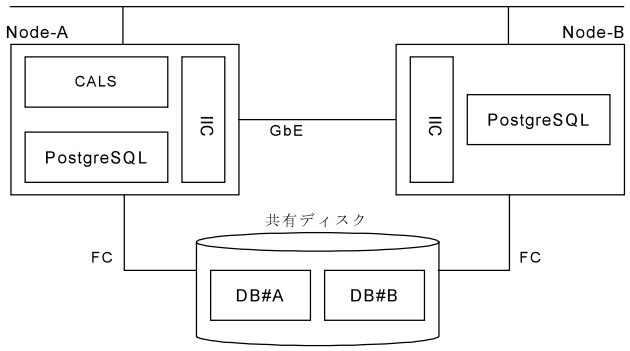

図 5　効率測定のクラスタ構成

pgbench の設定は，1 クライアントあたり，1000 トランザクションを実行する．また，初期データ件数は，1000 件とし，システムカタログ及び初期データはメモリに常駐した状態とする．測定は，3 回の実行の平均とし，クラスタ構成における TPS（Transaction Per Second：単位秒あたりのトランザクション数）は，各ノードの TPS の和とする．

また，pgbench を 2 ノードから同時に実行した場合，索引キーの重複により，ノード間での更新の競合が発生する．そのため，Node-A と Node-B のアクセスするデータベースを分割することで対処する．CALS 内でのロックの競合は，同一ノードのトランザクション同士によっても発生する．

効率測定は，参照と更新の比率を 7 対 3 とするケース，0 対 10（全て更新）とするケースのそれぞれで実施した[*6]．また，比較のため既存 PostgreSQL による測定を併せて行う．

図 6 と図 7 は，それぞれ参照/更新比率 7 対 3 と 0 対 10 の測定結果である．グラフの縦軸はスループット（単位 TPS），横軸は同時接続数を示し，それぞれのノードで同時接続数を増加させた場合のスループットの推移を示している．PG-CALS の同時接続数は，各ノードの和である．但し，同時接続数 1 の場合のみ，TPS が高い方のノードの結果を採用する．また，PostgreSQL の接続数パラメタ max_connections[*7] の値を 100 と設定しているため，既存 PostgreSQL での同時接続数 100 以降の TPS は測定していない．

7対3の場合，既存PostgreSQLでは，同時接続数40で最大6997tpsとなり，PG-CALSでは，同時接続数80で最大9457tpsとなる（図6）．0対10の場合，既存PostgreSQLでは，同時接続数40で最大2242tpsとなり，PG-CALSでは，同時接続数120で最大3881tpsとなる（図7）．

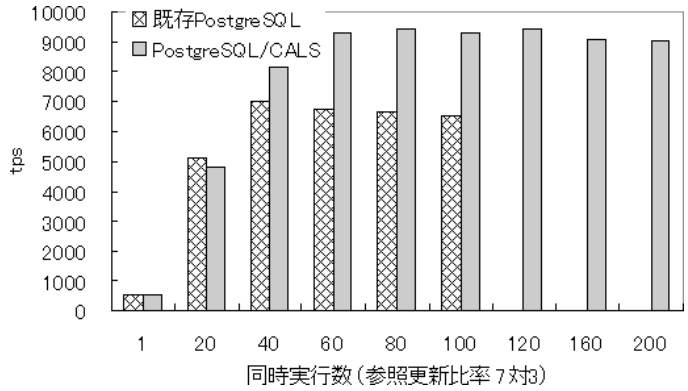

図6　スループットの推移（参照／更新比率7対3のケース）

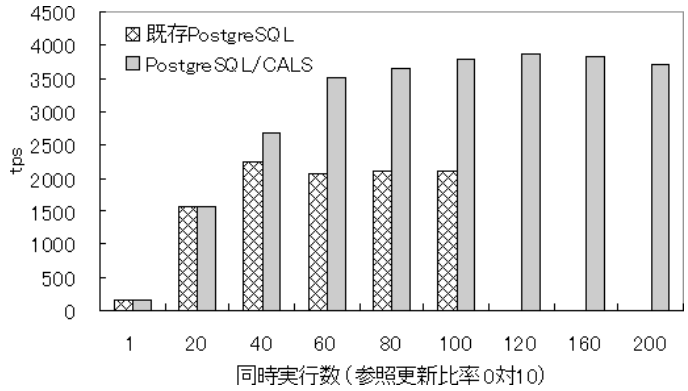

図7　スループットの推移（参照／更新比率0対10のケース）

表5は，参照更新比率7対3と0対10での既存PostgreSQLに対するPG-CALSのTPSの比率（既存PostgreSQLに対して何倍か）を示したものである．同時接続数100で，参照更新比率7対3の場合で約1.4倍，0対10で約1.8倍となる．これらの結果より，PG-CALSでは，既存PostgreSQLに対して，スケールアウトすることによって，スループットが向上することを確認した．

表5　既存PostgreSQLとPG-CALSのTPS比率

	同時接続数					
参照更新比率	1	20	40	60	80	100
7対3	1.01	0.95	1.17	1.38	1.42	1.44
0対10	1.00	0.99	1.19	1.70	1.73	1.80

参照比率の高い7対3のケースでは，採取したシステムリソース等の資料より，PG-CALS間のメッセージ通信によるオーバーヘッドが顕著となる．また，全て更新（0対10）のケース

では，WAL 処理によるログファイルへの書き込み I/O によって，CALS 処理のオーバーヘッドが目立たないという特徴が表れている．

4.2　効率改善策の検討

今回の性能検証によって，PG-CALS では，対象データがキャッシュされた条件下での単純な検索処理に効率上の問題があることが分かっている．これは，既存の PostgreSQL におけるロック管理とタプルの可視性判定の処理プロトコルに単純に従っただけの場合，PG-CALS 間のメッセージのやり取りが増大し，ネットワーク通信を含め，プロセス間通信におけるオーバーヘッドが顕在化するためである．また，プロセス間通信として，カーネル MQ[*8] を採用しているが，コネクション数の増加によってボトルネックとなり，性能が向上しないことを確認している．

改善策としては，不要なロック要求やタプル可視性判定を行わないための処理プロトコルの改良が必要であり，未更新ブロックの参照におけるロック要求とスナップショット情報取得の排除や，冗長なキャッシュ一貫性管理の排除の検討が必要である．また内部実装方式の改良としては，MQ に代わる効率的なプロセス間通信の手段として，ライブラリ化やシェアードメモリの利用による改善の検討を要する．

5. お　わ　り　に

本稿では，PG-CALS の実装技術について解説を行った．PG-CALS では，複数ノードからのデータリソースを共有するアクティブ/アクティブ型のクラスタを実装した．現時点では，未実装機能や改善すべき点を残すものの，実用化に向けた実装は可能であると評価している．

今後の課題としては，障害回復や可用性対応など，残された機能への対応と，将来的には基幹システムにおける DB エンジンとしての耐障害性や信頼性など，より高度なシステム要件に対応していく必要がある．また，PostgreSQL は，幾何データや地理データ，テキスト処理など，複雑なデータ型やデータ処理に特長がある．こうした PostgreSQL の機能と，PG-CALS によるシェアードページキャッシュ機能との連携により，リアルタイム性が求められるシステムへの適用など，両者の特長を生かせる新たな適用分野を探っていきたい．

＊ 1　PostgreSQL は ANSI SQL92 に準拠したオープンソースのリレーショナル DBMS であり，1980 年代に米国カリフォルニア大学バークレー校で開発された POSTGRES がベースとなっている．現在はコミュニティに引き継がれ，開発が継続されている．
＊ 2　OCFS2（Oracle Cluster File System）は共有ファイルシステムをサポートする，POSIX に準拠したファイルシステムである．カーネル 2.6.16 より Linux の標準として取り込まれている．
＊ 3　PostgreSQL では，Read Committed と Serializable がサポートされ，Read Committed がデフォルトである．Read Committed では，問い合わせの時点で COMMIT されているデータが参照可能となる．
＊ 4　データベースシステムでは，トランザクションの効率と同時実行性のため，バッファプール上の更新されたデータは，コミットとは非同期にディスクに書き込まれる．
＊ 5　同期チェックポイントでは，バッファプール上の更新ブロックがディスクに全て書き出されるまで，データの更新がブロックされる．
＊ 6　参照と更新の比率を変更できるように pgbench に改造を行った．また，デッドロックが発生しないように修正を行っている．
＊ 7　PostgreSQL の同時接続ユーザ数の上限を指定するコンフィグレーションパラメタ．
＊ 8　System V Message Queuing 機構．

参考文献 ［1］ L. Lamport, “Time, Clock, and the Ordering of Events in a Distributed Systems”, Communication of the ACM, Vol. 21, No. 7, July 1978.
［2］ C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, “ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging”, ACM Transactions on Database Systems, 1992.
［3］ C. Mohan, I. Narang, “Efficient Locking and Caching of Data in the Multisystem Shared Disks Transaction Environment”, Proc. 3rd International Conference on Advances in Database Technology, 1992.
［4］ C. Mohan, I. Narang, “Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment”, Proc. 17th International Conference on Very Large Data Bases, Barcelona, September 1991.
［5］ Oracle Corporation, 2007, http://www.oracle.com/technology/products/database/clustering/index.html
［6］ PostgreSQL Global Development Group, 2007, http://www.postgresql.org/
［7］ 阪口 喜好, “XTC-UDS の概要”, ユニシス技報, 日本ユニシス, Vol.12 No.1 通巻33号, 1992年5月.

執筆者紹介 中 山 陽 太 郎（Yotaro Nakayama）
1988年 東京学芸大学大学院教育学修士課程修了. 同年日本ユニシス(株)入社. HMP IX シリーズ, UNISYS オープン系データベース管理システムの開発保守に従事. 現在ユニアデックス(株)ソフトウェアプロダクト統括部 OSS 推進部に所属.