PGECons
PostgreSQL Enterprise Consortium

PostgreSQL エンタープライズ・コンソーシアム 技術部会 WG#2

システム構成調査編

製作者
担当企業名
日本電気株式会社

改訂履歴

版	改訂日	変更内容 ※表の列はスタイル「94.表内」を選択セルの背景は網掛け 25%灰色
1.0	2013/03/28	新規作成

はじめに

■本資料の概要と目的

本資料では、データ移行にあたって、目的にあった構成パターンを選択していただくための参考資料として、
RDBMS(PostgreSQL およびその他の RDBMS)の一般的な構成パターンとその特性を記載します。

■資料内の記述について

資料で記述する構文や例題について記述します。 ※スタイル「2.本文(中見出し)」を選択

目次

1. 背景

エンタープライズ領域のデータベースシステムには様々な要件が求められます。検索、更新、バッチといった各種の性能、導入コスト、高い可用性など、これらのニーズに応えるため、データベースシステムはシステムの規模や目的に応じて様々な構成を採用します。

本書は、既存のリレーショナルデータベースシステムから PostgreSQL へ移行する際に、システム構成を検討するための参考資料です。ただし、現在のシステム構成から PostgreSQL のシステム構成を機械的に決定するのは難しく、また危険なことです。移行元と同じシステム構成がPostgreSQLでは採用できない場合や、同じようなシステム構成でも達成している要件のレベルが異なることが往々に存在します。逆に、現在のシステム構成で達成していても、本当は対象のシステムであまり重要ではなく、システム移行の際には優先度を再検討する場合もあります。

このため、本書は、移行元のシステム要件と、リレーショナルデータベースの一般的な構成、その特徴を提示し、各構成について PostgreSQL が採用可能なシステム構成を紹介することで、システムのニーズにあったシステム構成を検討するための参考資料として活用していただくことを目的としています。

既存のリレーショナルデータベースシステムから PostgreSQL へ移行する際、現在のシステム構成がどのような要件に対応しており、PostgreSQL を利用するシステムで要件を満たすにはどのような構成を採用すべきか検討する場合の参考になれば幸いです。

なお、特別な記載がな限り、本書が対象としているデータベース管理システムとバージョンは以下の通りです。

- PostgreSQL 9.1
- MySQL 5.5
- Oracle Database 11g R2
- Microsoft SQL Server 2012
- IBM DB2 9.7

2. DBMSの要件とシステム構成

2.1. 求められる要件

データベースの移行にあたっては、どのような構成をとるかを検討する必要がありますが、データベースに対するニーズは様々で、現在のシステムにどのような要件が求められているのかを確認する必要があります。その要件の中でどれが重要で、必要性が高くないものは何かを見極めたうえで、採用するシステム構成を比較検討することが必要です。本章では、システム構成として求められるデータベースシステムへの要件と、本書で取り扱う範囲について確認していきます。

2.1.1. 可用性

多くのデータベースシステムは企業や団体の活動における最も重要なインフラであり、データベースの停止は業務の停止に直結します。特に社会インフラに影響を与えるようなミッションクリティカル性が高いシステムでは、業務継続やデータの安全性確保が最も重要な要件となります。なんらかの原因でデータベースシステムに障害が発生しても、できるだけ短時間でかつデータが保全された状態でサービスを再開できるようなシステム構成を採用することが必要です。また、地震や火災などの災害が発生した際にも遠く離れたバックアップサイトを用意することで、システム全体の運用を継続したりデータを保護するなどのディザスタリカバリへの要求も高まっています。

以上の内容からデータベースの可用性を以下の４つの項目に要約することができます。

- データベースシステムが停止しないこと
- 障害時の復旧時間が短いこと
- データの破壊、トランザクションの消失がないこと
- 災害に対する耐性(ディザスタリカバリ)

それぞれの要件は一つの方法で解決できることではなく、それぞれことなる対策を組み合わせてシステムを構成することによって満足させることができます。今回、本書は障害復旧とディザスタリカバリを中心に記載していきます。各システム構成の可用性レベルについては３章から７章までを参照してください。

2.1.2. 機密性

データベースには企業活動を行う上での重要な情報や、個人情報など重要な情報が格納されており、セキュリティの確保は必要不可欠ですので以下のような事態を回避するための対策を立てる必要があります。

- 情報の漏えい
- 情報の改ざん
- 資格がない人による利用

データベースシステム機密性を確保するための技術としては、ネットワークレベルのアクセス制御、ユーザの認証、ユーザやロールなどによるアクセス制御、データベースの暗号化、通信路の暗号化やデータベース監査ログ等が挙げられます。

ただし、これらの機密性確保に関する機能については、データベースシステム毎の機能や認証サーバとの連携に依存する部分が大きく、本書では取り扱いません。

2.1.3. 性能

データベースの性能指標としては以下のようなものがあり、どの性能指標を重視するかによって適したシステム構成が異なります。

- 応答時間
- スループット
- クライアントの同時接続数
- 同時実行トランザクション数

例えば、同時実行トランザクション数が増大し、1台のデータベースサーバだけでは処理できず、スループットを向上させる必要がある場合は複数台のデータベースシステムによる負荷分散クラスタ構成を検討する必要があります。

しかし、負荷分散することにより性能が逆に劣化する場合もあります。例えば、サーバ間のメモリバッファの同期や、レプリケーションを行う際の差分データ送信や同期による遅延、２フェーズコミットによる遅延等データベース性能劣化の原因となる場合ががあります。この性能劣化については可用性の為にデータベースの冗長構成をとる場合についても同様に発生する場合があります。

また、システムがデータベースシステムに要求する負荷のタイプによっても必要とされる構成は異なってきます。例えば、OLTP(On Line Transaction Processing)では、更新系処理が中心のシステムが多く、応答時間の即時性

が重要になります。分析処理を行う OLAP（On-Line Analytical Processing)は大量データを対象にした参照系が多く、応答時間の即時性はあまり重要視されない例になります。

2.1.4. 拡張性

システムやサービスの拡大によってデータベースシステムも処理量やデータ量が増加し拡張をしなければいけない場合があります。この時に行われる手法としては、大きくスケールアップとスケールアウトの 2 つが存在します。

スケールアップはデータベースシステムのプラットフォームを増強することで容量や処理能力の向上を実現します。システム構成を大きく変更することなく性能向上や容量の拡張が容易にはなりますが、本書では、特定のハードウェアに依存するなど本書が検討するポイントとは異なりますので、説明を割愛します。

スケールアウトには、要件によって様々な手法が存在します。例えば、本社で管理されるデータベースを支社から参照するシステムにおいて、支社が今後も増加する可能性があるといった場合には、マスタサーバのレプリケーションサーバを支社に置くといった対応方法でスケールアウトをすることができます。

一方で、蓄積したデータを分析して意思決定に使うシステムでは、分析するデータの量が膨大になることが想定されます。この場合は複数のサーバにスケールアウトができるデータベースシステムを検討する必要があります。

2.1.5. 運用性

一般的にデータベースのシステムを運用するには、様々な運用の観点があります。その中でもシステム構成に影響するものと言えば、監視（統合運用監視）やバックアップの取得、運用の自動化を行うためのジョブ管理、データベースに必要なメンテナンス処理（REINDEX や PostgreSQL の VACUUM 等）などの項目を検討する必要があります。一般的にはシステム構成が複雑になればなるほど検討項目が増え、運用が難しくなります。

2.1.6. 接続性

データベースシステムはあらゆるシステムと接続する可能性があります。アプリケーションはもちろんのこと、他のシステムのデータベースや DWH との連携、データの連携を行う ETL ツールなどや認証サーバ、その他ミドルウェア等と接続することが考えられます。移行に際しては、これらのツールとの接続実績や対応レベルなどを検討・調査対象にする必要があります。それだけではなく、基本的なレベルでは SQL の対応レベルや ODBC や JDBC ドライバなどの対応状況も異なることがあります。アプリケーションの影響が非常に大きくなるところですので、対応状況を確認する必要があります。

ただし、これらの条件は個別のデータベースシステムの対応レベルに依存するところが大きいため、本書では取り扱いません。接続性についての詳しい情報はアプリケーション移行調査編、SQL 移行調査編、組み込み関数移行調査編を参照して下さい。

2.1.7. 初期導入コスト

コスト削減を求められるのはいつの世でも変わらないことでしょう。コストと言っても、システム移行を行うにはいろいろなことに関わってきますが、ここでは初期導入コストを考えてみたいと思います。いわゆるサーバやストレージなどのハードウェアのコストやソフトウェアライセンスの費用です。当然、コストをかければかけるほど高機能になり、性能や可用性が高まる可能性が広がりますが、現実的にはプロジェクトの規模に応じたコストの上限があります。敢えて記載するまでもありませんが、システムの要件の優先順位を見極め、どこを重点的に対応していくかを検討する必要があります。

2.2. 本書の検討対象について

これら全体を取り扱うことは難しいため、本書としてはこれらの要件の中で、主に以下の要件を中心に検討します。

- 可用性
- 性能
- 拡張性
- 運用性
- 初期導入コスト

これらのニーズを満たす構成として、本書では以下の構成を紹介します。但し、これらの構成は組み合わせることもできるものであり、それぞれが独立な構成でないことをご注意ください。

- シングルサーバ構成
- HA クラスタ構成（Active-Standby 構成）
- データベースレプリケーション構成
- マルチマスタ負荷分散クラスタ構成

上の構成の特徴について、データベースシステムに依存する部分もありますが、大まかにまとめると表 2.1 のようになります。

表 2.1 システム構成による一般的な特性

		シングルサーバ	HA クラスタ	データベース レプリケーション	マルチマスタ 負荷分散クラスタ
可用性	サービス継続性	サーバ等の障害により サービスが停止する	クラスタ構成でのノード切替えによりサービスが継続可能	アプリケーション側でレプリカのDBに切り替えて業務を継続する実装が必要、クラスタ製品と組み合わせることで上記の問題は解決できる	シェアードエブリシング型は非常に短い切り替え時間を達成、シェアードナッシングはデータも含めた冗長構成が必要
	ディザスタリカバリ	バックアップ等の非同期的な手法実現する。同期的なディザスタリカバリは難しい。	HA クラスタ構成のみで実現することは原則は難しい。 レプリケーション機能と組み合わせることで実現可能	遠隔地へのレプリケーションすることで可能 レプリケーションの方式（同期・非同期）でデータをロストする範囲が異なる	原則ディザスタリカバリの実現は難しい、 レプリケーション機能と組み合わせることで実現可能
性能	参照系	データ量や同時トランザクション数が多くない場合は十分	シングルと同等。 レプリケーション機能との組み合わせで Act-Act にすれば参照負荷分散も	構成により参照負荷分散が可能	スケールアウトが可能 データ量に対する負荷分散にはシャーディングの設計が必要な場合がある
	更新系	データ量や同時トランザクション数が多くない場合は十分	シングルと同等。 ストレージ層の選択で性能が低下する可能性有り	差分データの送信や 同期処理によりシングル構成と比較して性能は劣化する可能性が高い	スケールアウトが可能 負荷分散にはシャーディングの設計が必要
拡張性		スケールアップにより対応	スケールアップにより 対応	参照系は スケールアウトが可能	参照系・更新系ともに スケールアウトが可能
運用・設計面		シンプルな構成であるため、運用設計項目やは複雑になりにくい	採用する方式により、クラスタのロールバック・ロールフォワードの設計、ストレージ層の同期設計などが必要	同期の設計、 参照負荷分散時のトランザクションの設計などが必要	シャーディングやリバランスなどを含めた設計が必要。自動チューニング機能を保有する場合もある
初期導入コスト		シンプルで 最も安価な構成	ノード間のデータ同期方式の選択によって異なる	ディザスタリカバリも検討すると、回線の費用やバックアップサイトの構築が必要	ハードウェア台数やソフトウェアライセンス費などで高価になりがち

上記の表で紹介した通り、記載してある適性もかなり幅があります。これは、それぞれの構成要素は幅を持っているため、組み合わせることでお互いの弱点をカバーすることができますし、選択したソフトウェアやハードウェアの特徴や成熟度などによって、特性も変化します。

データ処理量が少ない場合はシングルサーバ構成であっても、スケールアウト構成でも変わらない基礎性能を達成することもあり、まずはシステム構成を検討する前にそれぞれの基本的な特性を理解することが重要です。

それでは次の章から具体例を見ながら、各データベースシステム構成の特徴および PostgreSQL での構成方法について、解説していきます。

3. シングルサーバデータベース

シングルサーバデータベースとは1台で構成されるデータベースシステムのことです。データベースサーバとしては基本の構成になり、シンプルな構成で運用自体も簡単です。サーバ等の障害に対する可用性はありませんが、システム構成としては一番安価に構成ができます。シングルサーバ構成自体にはご紹介するものはございませんが、他の構成でも参考にもなりますので、本章では主にストレージ構成毎にの特徴を紹介します。

3.1. 内蔵ストレージ [Direct Attached Storage (DAS)]

サーバの内蔵ストレージでデータベースを運用する構成です。シンプルなハードウェア構成であるため、最も安くデータベースを構築することができます。ストレージの専門知識は必要なく、構築が簡単で導入価格から比較すると性能も高いですが、サーバに格納できるストレージ数を超えて拡張することは難しくなり、将来の容量追加がある場合は外部ストレージを検討したほうがよいでしょう。また、複数のサーバではストレージの共有ができないため、利用効率が下がるのも欠点です。また、企業全体でストレージを管理できないので別々にバックアップを行う必要あることや、障害検知などを行う必要があり、運用面で煩雑になりやすい問題があります。

3.2. SAN(Storage Area Network)

ストレージを FC(Fibre Channel)などのネットワーク上でシリアル SCSI プロトコルを用いてネットワーク化して運用します。このため、ストレージ専用のネットワークを通してストレージを共有することができます。メリットとしては、SAN はブロック単位アクセスをするため、テータ転送のコストが低く、性能面で優れていることが挙げられます。さらに、レプリケーション機能があるストレージの場合、レプリカ側のストレージを用いてバックアップを取得することで、マスタ側へのバックアップ実行時の性能の低下を最小限に食い止めることも可能になります。デメリットとしては高価のハードウェアが必要であるため、導入コストが高価で、ストレージ上にファイルシステムがないため、専門的な技術が必要な点が挙げられます。

3.3. NAS(Network Attached Storage)

NAS は SAN と違ってストレージ上にファイルシステムがあるため、ネットワークを介してボリュームを共有することができます。そのため、SAN とは違って専門知識が無くても管理することができます。また、特殊なハードウェアが必要ないため、SAN より安価で構成することができます。しかし、NAS はファイルシステムを介して接続するためブロックレベルでアクセスする内蔵ストレージや SAN より遅延時間(Latency Time)が長くなります。このため、内蔵ストレージ、SAN より性能が劣るので、データベースのストレージとして利用するには、性能面の工夫が必要です。また、設計時には NAS と接続するネットワークが独立されてない設計の場合、他の通信の影響をしないように注意する必要があります。

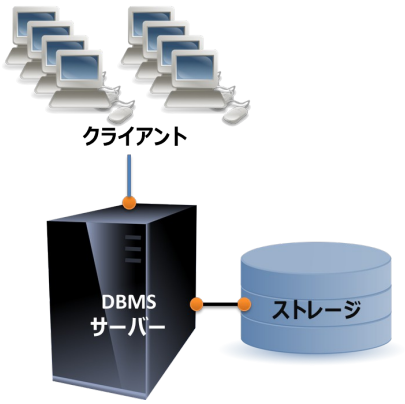

図 3.1: シングルサーバデータベース

3.4. 比較表

次の表は、各ストレージの特徴を表したものです。

表 3.1 各ストレージを用いた場合のシングルサーバ構成の特徴

	内蔵ストレージ	SAN	NAS
導入コスト	安価	比較的高価	SAN より安価
可用性	サーバの障害等でサービス停止	サーバの障害等でサービス停止	サーバの障害等でサービス停止
性能	良い	良い	工夫が必要
扱いやすさ	簡単	専門知識が必要	簡単
拡張性	サーバ筐体の物理的な制限を受けやすい	良い	良い
運用性（バックアップ・監視等）	それぞれのサーバで運用を考慮する必要がある	システム全体で一元管理が可能	システム全体で一元管理が可能
システム内のディスクの共有	-	可能	可能

3.5. まとめ

シングルサーバデータベースは1台で構成されるため、もっとも安価に実現できますす。

シングルサーバであるためサーバが単一障害点になり、可用性は高くありませんが、RAID 構成、バックアップなどを利用してデータの保全や業務の継続を一定のレベルで実現することは可能です。また CPU やメモリといったリソースの拡張性（スケールアウト）も難しい構成です。十分な可用性を確保することが困難なため、ミッションクリティカル性が高いシステムの場合は他の構成を考慮することを推奨します。

4. HA クラスタ(Active-Standby 構成)

HA クラスタ(Active-Standby 構成)とは現用サーバ(Active サーバ)に障害が発生したとき、障害を検知し、予備サーバ(Standby サーバ)にフェイルオーバして運用を続けることで可用性を確保するシステムのことです。HA クラスタ自体はアプリケーションサーバや Web サーバ、認証サーバなど、ありとあらゆるシステムで用いられており何もデータベースシステムに特化した構成ではありません。そこで、本章ではHAクラスタ構成のデータベースシステムにおいて、検討が必要となるデータ共有方式を中心に特徴について記載します。

4.1. 共有ストレージ方式

NAS や SAN のような共有ストレージを利用して HA クラスタを構成します。共有ストレージを利用する場合はクラスタを構成する為にレプリケーションを利用する必要が無いため、更新性能面では有利な構成です。また、ストレージの機能で運用性を高めることができるのもメリットです。例えば、ディスクアレイ内で行うレプリケーション機能をもつストレージを採用した場合には、バックアップをレプリカ側から取得することで、バックアップ中のマスタ側への性能の影響をほとんど与えない運用なども可能となります。しかし、一般的に共有ストレージはコストが高いため、他のデータ共有方式よりも導入コストが高くなります。データは1箇所にあるため、現用サーバと予備サーバ間のデータの整合性を気にする必要はありませんが、システム構築時には共有ストレージが単一障害点(SPOF)にならないように注意する必要があります。

なお、フェイルオーバ時には現用サーバが使用していたストレージをそのまま利用するため、スタンバイ側がファイルシステムの整合性を確認する必要があります。ext2 等のファイルシステムではこの整合性の確認にかなりの時間を要していましたが、ext3 や ext4 などのではジャーナリングファイルシステムの搭載や機能改善により切替時間が短縮しています。

4.2. ストレージレプリケーション方式（ミラーリング）

ストレージレベルでレプリケーションするソフトウェアなどを利用して HA 構成クラスタを構成します。共有ストレージが必要ないので安価に導入ができますが、大規模の更新処理が発生するシステムではレプリケーションのオーバーヘッドにより、共有ストレージを利用するシステムより更新性能が劣化します。また、何らかの障害が発生してフェイルオーバした後では、片方のノードにだけ更新が進んでデータ同期が崩れていきますので、クラスタを再構成する際にはデータの再同期を行う必要があります。この再同期処理でさらに性能が劣化する可能性があるため、運用面では注意が必要です。

なお、これらのミラーリングはディスクのブロックレベルでの同期になり、共有ストレージ方式の際に説明した、ファイルシステムの整合性確認が必要になる場合があります。

PostgreSQL では、オープンソースソフトの DRBD を利用して実現している例が多く報告されています。

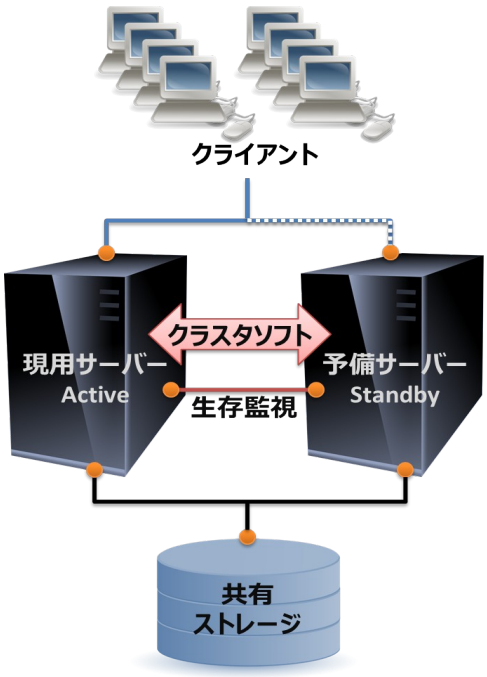

図 4.1: 共有ストレージ方式

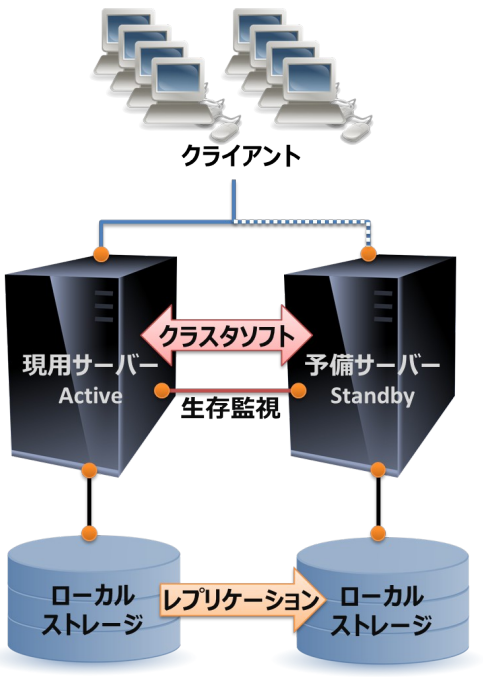

図 4.2: ストレージレプリケーション方式

4.3. 比較表

比較表は表 5.1 にデータベースのログシッピングレプリケーションと共に記載していますのでそちらを参照してください。

4.4. まとめ

Active-Standby 方式の HA クラスタ構成は後述する負荷分散クラスタより比較的に安価で技術的にも簡単にシステムを構成することができますが、比較的にフェイルオーバにかかる時間が長いため、ミッションクリティカル性が非常に高くフェイルオーバのかかる時間に対して極めて厳しい要件があるシステムの場合は別の方式を検討する必要があります。PostgreSQL に本構成を適用する場合、DBMS 自体に自動フェイルオーバの機能がないため、他のクラスタソフトを利用して、自動フェイルオーバクラスタを構築する必要があります。

5. データベースレプリケーション

データベースレプリケーションはデータを複数のサーバにデータの複製(レプリカ)を作ることでデータの冗長性を確保する手法のことです。ここで紹介するレプリケーションは 4.2 ストレージレプリケーション方式（ミラーリング）で記載した、ストレージレベルのレプリケーションではなくデータベースシステムの機能を用いて行うレプリケーションのことです。データベースレプリケーションを利用すると、データベース自体を二重化して保有することができるため 5 章で述べた HA クラスタを構成することが可能なほか、ディザスタリカバリ、参照負荷分散の目的に利用することも可能です。本章ではレプリケーション手法の一つであるログシッピングレプリケーションを利用した負荷分散クラスタについて記載します。

5.1. ログシッピングレプリケーション

ログシッピングレプリケーションとはマスタサーバがデータベースの更新ログをスレーブサーバに転送してスレーブサーバからロールフォワードすることによりデータベースのレプリケーションを実現する方式です。この機能はデータベースエンジンが作成する更新ログを利用しているため、多くの場合データベースシステム自身の機能として搭載されています。

ログシッピングレプリケーションは、ストレージレベルの共有と違い、マスタサーバ・スレーブサーバのファイルシステムは独立しているため、更新処理中にマスタサーバで障害が発生しても、ファイルシステムのチェックを行う必要はありません。また、更新ログが順次スタンバイサーバで適用されておりデータベースとしては健全な状態になりますので、更新データ量などの状況にもよりますが、ストレージレベルの共有よりフェイルオーバが早くなる可能性が期待できます。

ただし、ディスクミラーリングと同様に障害が発生したマスタサーバを構成から外し、スタンバイサーバをマスタとして利用すると、レプリケーション構成を復旧するためにはデータベースの再同期が必要です。

ここではレプリケーションの同期レベルによる違いについて述べます。

5.1.1. 同期レプリケーション

同期レプリケーションはクライアントから発行されたCOMMIT処理に対して、データベースの更新ログがスレーブに転送されたことを確認した時点で、クライアントに結果を返却します。このため、マスタサーバの障害でCOMMIT済のデータが失われることはありませんが、COMMIT実行時の更新ログの転送完了までのオーバヘッドが発生するため更新性能への影響が大きく、性能劣化を最小化するためにサーバ間のネットワークやスレーブのストレージに対して高い性能がが求められます。

5.1.2. 非同期レプリケーション

非同期レプリケーションはマスタサーバに出力された更新ログをを非同期にスレーブに転送します。このため、同期レプリケーションと比べるとレプリケーションのオーバーヘッドは比較的に小さく抑えることができますが、マスタサーバの障害が発生したとき、COMMIT済みのデータが失われる可能性があります。また、スレーブサーバを参照系で利用するときにはマスタサーバとのデータ反映の時間差があるため、一時的にマスタサーバとスレーブサーバの参照結果が異なる可能性があります。

5.2. その他のレプリケーション

ログシッピングレプリケーション以外にもトリガベースレプリケーション、クエリベースレプリケーションが広く使われています。トリガベースレプリケーションはマスタサーバのすべての更新情報をスレーブサーバに非同期に反映するレプリケーション方式です。この方式では、SQL のレベルでレプリケーションが実装されていますので、外部ツールとして開発がしやすいことや異種のデータベースでのレプリケーションが実装できるなどのメリットがあります。PostgreSQL での代表的な実装例としては、Slony- I が挙げられます。

クエリベースレプリケーションはすべての更新クエリを全サーバで実行することでデータをレプリケーションします。PostgreSQL での代表的な実装例としては pgpool- II が挙げられます。

5.3. 比較表

次の表は、共有ストレージおよびストレージレプリケーション手法とレプリケーションの同期レベルによる特徴を表したものです。

表 5.1.データ共有方法による特徴

	共有ストレージ方式	ストレージ レプリケーション方式	同期レプリケーション	非同期レプリケーション
導入コスト	共有ストレージが必須で比較的高価	共有ストレージは不要であり比較的安価、ミラーリングについては商用ライセンスが必要な場合ある	データベース本体の機能で実装されているが、DBMS によってはオプション製品が必要な場合がある	同左
フェールオーバ時間	現用サーバの更新中に障害が発生した場合、ファイルチェック時間が掛かる可能性がある。	現用サーバが更新中に障害があった場合、修復に時間が掛かる可能性がある。	ファイルシステムのチェックなどが必要なく、共有ストレージ方式より短い	同左
可用性	共有ストレージが SPOF にならないような構成が必要	同期モードの場合はマスタに障害が発生してもCOMMIT済みデータの安全は保証される	マスタ障害時でもCOMMIT済みデータの安全は保証される	マスタ障害時COMMIT済みのデータを失う恐れがある
データ整合性	データが 1 箇所にあるためアクティブ・スタンバイサーバ間のデータ整合性問題は発生しない	同期モードを選択した場合は、ブロックレベルの整合性を保証する。障害発生後にレプリケーションを再構築するには再同期が必要	トランザクションレベルでマスタ・スレーブサーバ間の同期を保障する。じ(PostgreSQL では検索結果の同期は保証しない) 障害発生後にレプリケーションを再構築するには再同期が必要	マスタサーバに対してスレーブサーバの更新は遅延している可能性がある。 障害発生後にレプリケーションを再構築するには再同期が必要
性能	レプリケーションなどのオーバヘッドの影響がない	レプリケーションによるオーバーヘッドがある	更新処理において、同期によるオーバヘッドが発生するため、更新性能の劣化する可能性がある	レプリケーションによる更新処理のオーバーヘッドはあるが、同期レプリケーションよりその影響は小さい

5.4. まとめ

データベースレプリケーション構成は HA クラスタ、参照負荷分散などの目的で利用することが可能です。

参照負荷分散を目的としたログシッピングレプリケーション構成ではデータが 2 箇所以上に存在するため、目的によってデータの整合性(同期、非同期)のレベルを考慮する必要があります。

データベースレプリケーションを利用した HA クラスタを構成する場合、同期レプリケーションを利用すると、共有ストレージを利用した Active-Standby 構成よりフェイルオーバの時間を短縮できる可能性があります。[1]しかし、性能劣化があるため、目的により同期、非同期のレベルを考慮する必要があります。

1 海浦 隆一(2012), 「PostgreSQL 9.1 でつくる高可用システムにまつわるエトセトラ」 http://www.postgresql.jp/events/pgday12files/M3slide

6. ディザスタリカバリ

データベースシステムにおけるディザスタリカバリとは地震、火災などの災害などによる被害からデータを保護してサービスを継続させるような方法を採用することを言います。データを保護する代表的な方法としては、RAID を用いてストレージの冗長性を確保する方法などがありますが、地震や火災などの災害によりデータセンター全体の問題が発生した場合にはデータを保護することができません。このような災害からデータを保護してサービスを続けるためには、4.2 節で紹介したストレージレプリケーション、または 5.1 節で紹介したログシッピングレプリケーションを利用することが一般的です。

ストレージレプリケーションやログシッピングレプリケーションの場合、ネットワークを利用して遠隔地に複製(レプリケーション)を作ることが可能であるため、災害によりデータセンターが破壊された場合でもデータを保護してサービスを継続することが可能です。ディザスタリカバリを考慮したシステムは災害による障害を考慮しているため、各サーバが地理的に分散されていることが特徴です。このため、サーバ間のネットワーク性能(帯域幅や信頼性)が高くない場合が多く、非同期レプリケーションの手法を採用することが一般的です。また、最近では転送するデータを圧縮することで帯域幅が少なくてもいいようにする手法も出てきています。

7. マルチマスタ負荷分散クラスタ

マルチマスタ負荷分散クラスタとは複数のサーバにまたがってデータを保持して各サーバで並行に処理することで負荷分散を実現したデータベース構成で、特に更新系も含め負荷分散を行うことができるのが特徴です。今まで説明したログシッピングレプリケーションなどでも、参照系のクエリでは複数台のデータベースサーバを利用し負荷分散を行うことができましたが、更新系のクエリについては 1 台に集約する必要がありました。

また、マルチマスタ負荷分散クラスタにはシェアードエブリシング負荷分散クラスタ、シェアードナッシング負荷分散クラスタと大きく2つのパターンに分かれますが、どのパターンを採用しているかはデータベースシステムによってバラバラなのも特徴で、達成している機能や範囲も大きく異なります。

本章では、これら2つの方式について全体的な特徴を記載します。

7.1. シェアードエブリシング負荷分散クラスタ

シェアードエブリシング負荷分散クラスタは複数のサーバに共有ストレージを同時にマウントして運用するデータベースです。各サーバは同時に SQL を実行できて、1 つのサーバに障害が発生しても無停止で運用を続けることができます。

この構成では更新データをどのサーバに割り当てるかといったシャーディングの設計が必要になる場合が多いですが、更新系、参照系の両方のクエリの負荷分散が可能であるため、高い性能を発揮できます。また、ストレージが 1 か所に集約しており、更新した情報も各サーバ間で共有できるようなしくみを保有しているため、切り替え時間が非常に小さくなり、またアプリケーション側からもシームレスに行われます。さらに、サーバを追加することでデータベースの性能がスケールアウトすることができるなど非常に多くのメリットがある構成です。しかし、共有ストレージと高価のソフトウェアが必要になるので導入コストが非常に高くなる可能性がある構成です。システム構築時には共有ストレージが単一障害点(SPOF)にならないように注意する必要があります。

PostgreSQL では、マルチマスタ型のシェアードエブリシング負荷分散クラスタは使われていません。

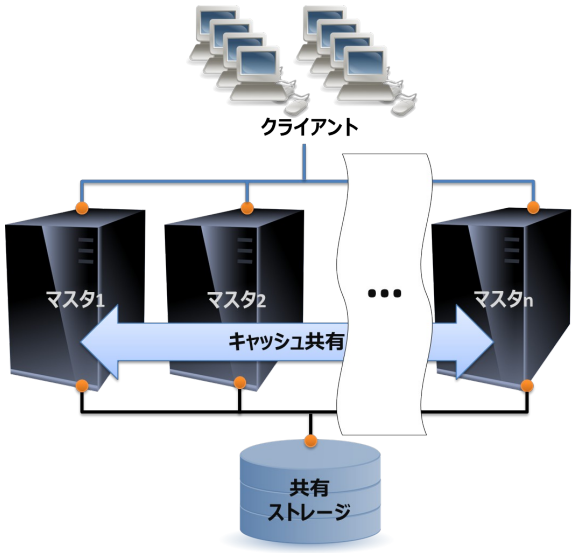

図 7.1: Oracle RAC

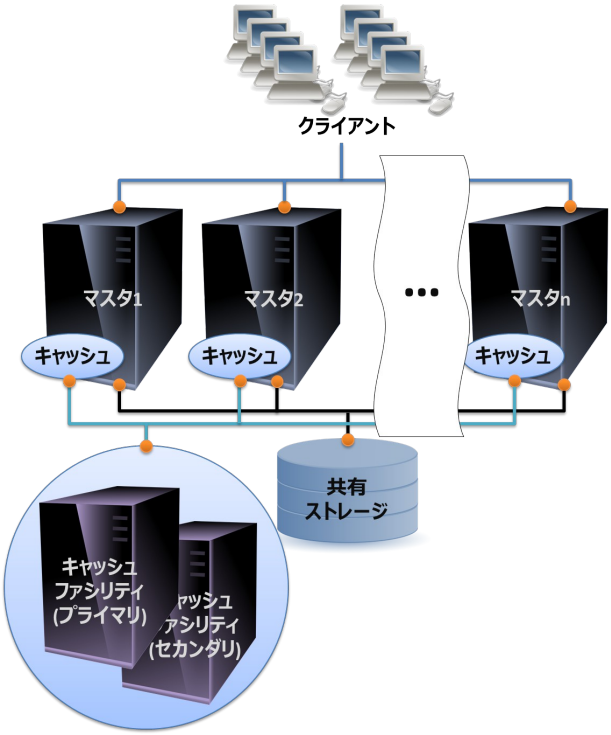

図 7.2: DB2 pureScale

シェアードエブリシング負荷分散クラスタの構成例

7.2. シェアードナッシング負荷分散クラスタ

シェアードナッシング負荷分散クラスタは前述したシェアードエブリシング構造とは違ってサーバ間で共有する資源を持たないデータベースです。設計にも依存しますが、基本的にはリソースを共有しないため、理論上シェアードエブリシング負荷分散クラスタより柔軟なスケールアウトができます。なお、通常のデータベースサーバを複数台立ててアプリケーション側でデータ分割し、結果の集計などもアプリケーションが意識して吸収する方式もありますが、本書ではシェアードナッシング負荷分散クラスタの対象外とします。

一方で、今回の議論の対象からは少し話はそれてしまいますが、スケールアウトがしやすく、参照系の処理に強い負荷分散クラスタであるため、一般的に DWH を指向しているデータベースシステムに多く採用されている構成になります。

この構成は特殊なハードウェアが必要なく、大規模構成の割には安価でシステムを構成することができます。しかし、データが分散されているため、データパーティショニングやルーティングなどを考慮する必要があります。また原則的にはデータを分散してデータを保有しているため、可用性の面では、データを冗長化させるような構成を検討する必要があります。

PostgreSQL では、pgpool- II 、Postgres-XC でマルチマスタ型のシェアードナッシング負荷分散クラスタを実現することができます。

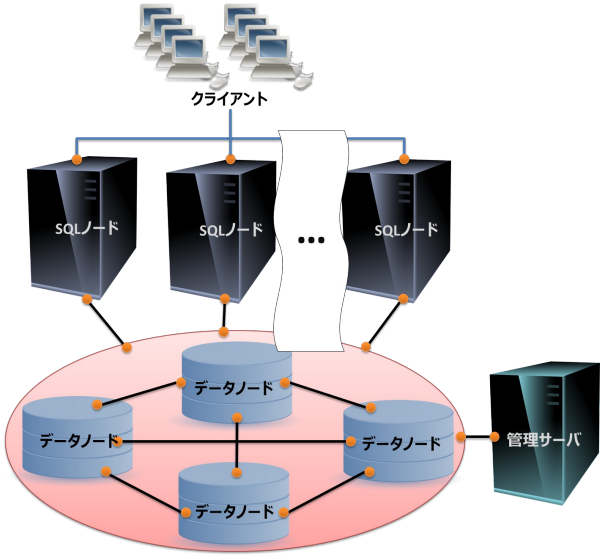

図 7.3: MySQL Cluster

シェアードナッシング負荷分散クラスタの構成例

7.3. 比較表

次の表は、シェアードエブリシング、シェアードナッシングの特徴を表したものです。

表 7.1.シェアードエブリシング方式とシェアードナッシング方式の特徴

	シェアードエブリシング	シェアードナッシング
導入コスト	サーバのスペックや台数に応じて、高価な商用ライセンスのソフトウェアが必要 また、共有ストレージも用意する必要がある	共有ストレージが無くても構成可能だが、大規模なスケールアウトを行う場合は HW の規模が大きくなる ライセンス面では OSS ベースが多くメリットが出やすい（除く DWH 向けの商用 DBMS）
フェイルオーバ時間	短い、障害によってはアプリケーション側にほとんど影響を与えず運用を続けることも可能	通常の HA 構成と同等のフェールオーバの仕組みを用いることが一般的で、ある程度の時間が必要 障害があった構成要素のみフェイルオーバを行う
可用性	障害があったサーバのトランザクションを引き継いで処理することも可能	HA 構成やレプリケーションなどによってデータの冗長性を取る方式が一般的 可用性の考え方については 4、5 章を参照
データ整合性	データは 1 箇所のみにあり、更新情報も共有できる仕組みを持っている	シャーディング、レプリケーションによるデータ整合性を考慮する必要がある
更新スケールアウト	可能だがキャッシュ共有があまり発生しないように設計を行う必要があり、あまり大規模なスケールアウトには向かない場合もある	シャーディング、レプリケーションの方式に依存 サーバ別にシャーディングした場合、スケールアウトするが、レプリケーションによるデータ重複度が高い場合は基本的にスケールアウトしない
参照スケールアウト	可能	可能

7.4. まとめ

シェアードエブリシング負荷分散クラスタは高可用性とスケールアウトによる性能向上など良い特性を持ちます。しかし、導入コストが高くなりがちで、適用するシステムの可用性や性能要求などを踏まえて採用を判断するとよいでしょう。また、更新系の処理を効率よく負荷分散するには、原則的にパーティショニングやハッシュといったシャーディングの設計を十分にして、できるだけクエリに対して複数のサーバでのデータ通信などのオーバヘッドが発生しないような設計が必要です。

シェアードエブリシングタイプの構成では、共有ストレージの障害がシステム全体に影響を与えるため、単一障害点(SPOF)にならないように構成する必要があります。

シェアードナッシングタイプの構成では原則的には共有リソースがないため、シェアードエブリシング構成よりスケールアウト特性は優れていますが、データが異なるサーバに分散されている場合、結合などの処理に時間がかかります。また、データが分散されているため動的な負荷分散をするためにはシャーディングの方法を工夫する必要があります。

8. PostgreSQL のシステム構成

本章では以上のデータベースシステムの構成から実際に PostgreSQL で取れる構成例とその特性要件のイメージを含めてご紹介します。本章で紹介する構成は以下になります。

- DRBD を利用した HA クラスタ構成(4.2)
- ストリーミングレプリケーションと pgpool-II を利用した参照負荷分散クラスタ構成(5.1)
- Slony- I を利用した部分レプリケーション構成(5.2)
- カスケードレプリケーションを利用したディザスタリカバリ構成(6)
- Postgres-XC を利用した HA 構成(7.2)

それぞれの構成は基本的に各項目の後ろに記載した括弧の中の章番号に対応する構成になりますが、PostgreSQL の特性、または構成の内容によって、その特徴が異なる部分もあることに注意してください。

8.1. DRBD を利用した HA クラスタ構成

8.1.1. DRBD(Distributed Replication Block Device)

DRBD とは Linux フラットフォームで利用可能な分散ストレージシステムです。ネットワーク経由で TCP/IP を利用し、ストレージのブロックレベルのレプリケーションを行います。イメージ的にはネットワーク越しの RAID1 を実現したようなものです。ブロックレベルで動作するため、ファイルシステムに依存することなくレプリケーションをすることができます。また、TCP/IP を利用するため、遠隔地でのレプリケーションにも利用できるといったメリットもあります。

8.1.2. 構成例

図 8.1 は DRBD を利用した Active-Standby の構成例です。

本構成は以下の要件を想定しています。

- システムの停止は業務の停止に直結するためシステムを停止を回避する必要がある。
- サーバ、ストレージの障害からデータの安全を保証すること。
- 初期導入費用はなるべく抑える必要がある。
- サービスの処理量が大きく増大に備える必要はない。

本構成は、アクティブ(現用)サーバに障害があった場合、オープンソースのクラスタソフトである Pacemaker が障害を検知、フェイルオーバを行い、データは DRBD でレプリケーションすることでシステムおよびデータの高可用性を実現しています。SAN のような高価の共有ストレージが無くても、一つのブロックデバイスとして認識して利用することが可能です。

デメリットとしては、レプリケーションによるオーバヘッドが存在するため、SAN と比較すると更新性能が落ちてしまうことです。また、スケールアウトなどで性能改善を行うことも難しくなってしまいます。また、細かなポイントではありますが、ストレージレベルでのレプリケーションなので何らかのファイルシステムや OS、ディスクの問題などでマスタに壊れたデータが更新されるなどゴミのデータが発生したときも、DRBD はそれを認識することができない可能性があります。これにより壊れたデータもレプリケーションすることで、障害の検知などが遅れてしまうことが考えられます。一方でレプリケーションの方式がログシッピングレプリケーションであれば、データベースシステム自体が転送されたログデータの整合性を確認するため、障害として検知される可能性が高くなります。

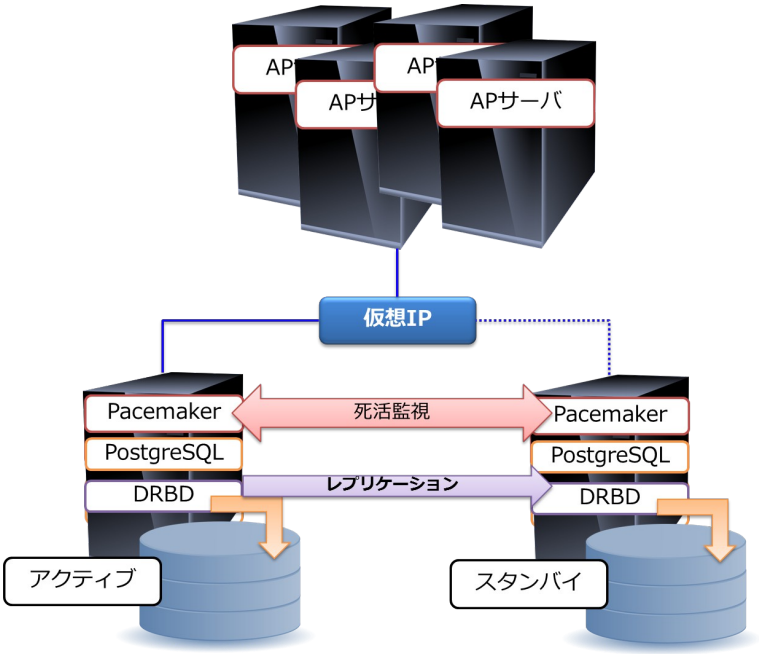

図 8.1: DRBD を利用した Active-Standby 構成

8.2. ストリーミングレプリケーションと pgpool-II を利用した参照負荷分散クラスタ構成

8.2.1. ストリーミングレプリケーション

ストリーミングレプリケーションは PostgreSQL のログシッピングレプリケーションを行う機能であり、PostgreSQL9.0 から利用できます。最初に登場した 9.0 では非同期レプリケーションだけでしたが、PostgreSQL9.1 からはログを確実に送付するまで同期するレプリケーションをサポートするようになりました。しかし、この同期の中身は、マスタサーバでCOMMIT済みのデータがすぐにスレーブサーバから参照できるという意味ではないことに注意してください。PostgreSQL9.2 までではCOMMIT済みデータをすぐスレーブサーバから参照できることを保証する機能は実装されておらず、PostgreSQL の同期レプリケーションの同期レベルを準同期と表現する場合もあります。このため、完全な同期を確保するには pgpool-II など別の方式を採用する必要があります。

ストリーミングレプリケーションの詳細については下記の PostgreSQL マニュアル[2]をご覧ください。

8.2.2. pgpool-II

pgpool-II は PostgreSQL 専用のミドルウェアで、以下のような機能を持っています。

- コネクションプール
- レプリケーションモード
- ストリーミングレプリケーションモード
- 負荷分散
- 接続数制限
- パラレルクエリ

レプリケーション関連機能にはレプリケーションモードとストリーミングレプリケーションモードの 2 つがありますが、前者は pgpool- II がアプリケーションから受け取ったクエリを複数のサーバに転送してレプリケーションをする方法で、後者は PostgreSQL の本体の機能であるストリーミングを利用したレプリケーションモードです。

pgpool- II の詳細については下記の pgpool-II ユーザマニュアル[3]をご覧ください。

8.2.3. 構成例

図 8.2 と図 8.3 はそれぞれ pgpool- II のレプリケーションモードを利用した負荷分散クラスタの構成例とストリーミングレプリケーションと pgpool-II を利用した参照負荷分散クラスタ構成の例です。

本構成は以下の要件を想定しています。

- システムの停止は業務の停止に直結するためシステムを停止できない。
- サーバ、ストレージの障害からデータの安全を保証すること。
- 初期導入費用はなるべく抑える。
- サービスが拡張される可能性があり処理量の予測が難しい。

本構成では pgpool- II でクラスタ構成をすることで高可用性を実現しています。データの冗長性については図 8.2 は pgpool- II をネイティブレプリケーションモードを、図 8.3 はストリーミングレプリケーションモードを利用して、データを冗長化しています。レプリケーション方式の場合は、共有ストレージを購入する必要がないため、初期導入費用を抑えることも期待できます。pgpool- II は障害検知やフェイルオーバ機能以外にもロードバランサとしての機能も備えているため、参照負荷分散が可能です。さらにレプリケーション数を増やすことで参照性能を向上させることも可能です。しかし、更新時に発生するレプリケーションのオーバヘッドのため、スケールアウトには限界があります。デメリットとしては全てのクエリが pgpool- II を経由するためオーバヘッドがあること。また、pgpool- II の障害の可能性があるため、pgpool- II の冗長構成が必要になることです。

図 8.2 の構成の場合は、同じ命令でもサーバによって結果が異なる処理などの場合、データは一致しなくなるため、対策が必要です。例えば、乱数を生成して更新する場合は予めアプリケーションで値を生成する必要があります。

2 「PostgreSQL9.1 最新版 日本語マニュアル 25.2.5 ストリーミングレプリケーション」 http://www.postgresql.jp/document/9.1/html/warm-standby.html#STREAMING-REPLICATION

3 「Pgpool-II ユーザマニュアル 」 http://www.pgpool.net/docs/latest/pgpool-ja.html

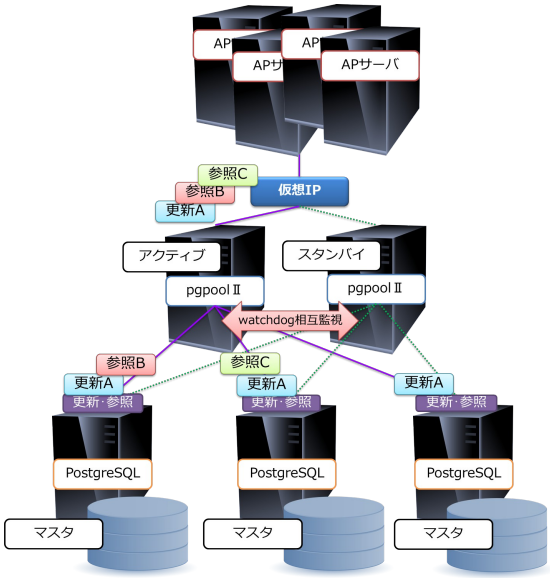

図 8.2: pgpool-Ⅱ のレプリケーションモードを利用した負荷分散クラスタの構成例

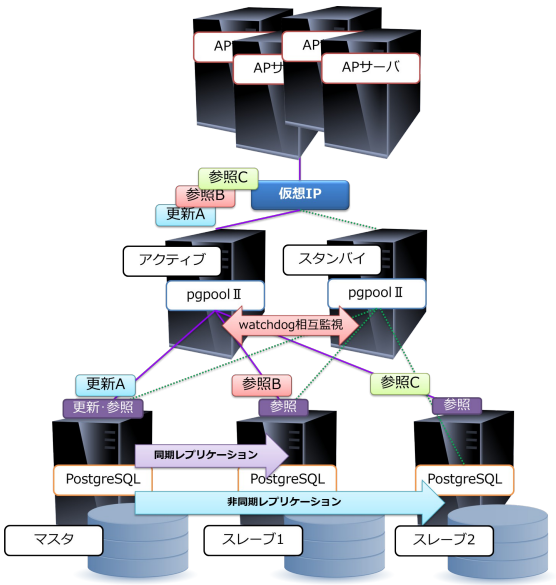

図 8.3: ストリーミングレプリケーションと pgpool-Ⅱ を利用した負荷分散クラスタの構成例

8.3. Slony-I を利用した部分レプリケーション構成

8.3.1. Slony-I

Slony-I は PostgreSQL 用のトリガーベースのマスタスレーブレプリケーションを提供するツールです。PostgreSQL は 9.0 からストリーミングレプリケーションをサポートしていますが、PostgreSQL 本体のレプリケーション機能とは異なる特徴を持つため、PostgreSQL本体にレプリケーション機能が搭載された今なお有用なツールです。以下に slony-I の代表的な特徴を記載します。

- OSや PostgreSQL のバージョンが違ってもレプリケーションが可能です。
- テーブルごとにレプリケーションすることができます。このため、ノードごとに違うテーブル構成をすることが可能になります。

ストリーミングレプリケーションと Slony-I についての議論は下記の文書をご覧ください。

- *Streaming Replication vs Slony*[4]

8.3.2. 構成例

図 8.4 は Slony-I を利用した部分レプリケーションを利用した参照負荷分散の構成例です
本構成は以下の要件を想定しています。

- システムの停止は業務の停止に直結するためシステムを停止できない。
- サーバ、ストレージの障害からデータの安全を保証すること。
- 海外拠点と本社間の帯域幅が細いためネットワーク負荷を抑えたい。
- セキュリティのため、海外拠点には必要なデータだけを公開したい。

本構成は、アクティブ(マスタ)サーバに障害があった場合、Pacemaker が障害を検知、スタンバイサーバが共有ストレージをマウントしてフェイルオーバします。データ冗長性は RAID を用いて確保します。さらに、海外拠点に必要な情報だけをレプリケーションすることで、マスタサーバのレプリケーションで発生するオーバヘッドの軽減や、また海外拠点へのネットワーク負荷の軽減、セキュリティ面の強化などの効果が期待できます。デメリットとしては、トリガーによるレプリケーションなので DDL による変更やロールの変更が反映されない問題があります。

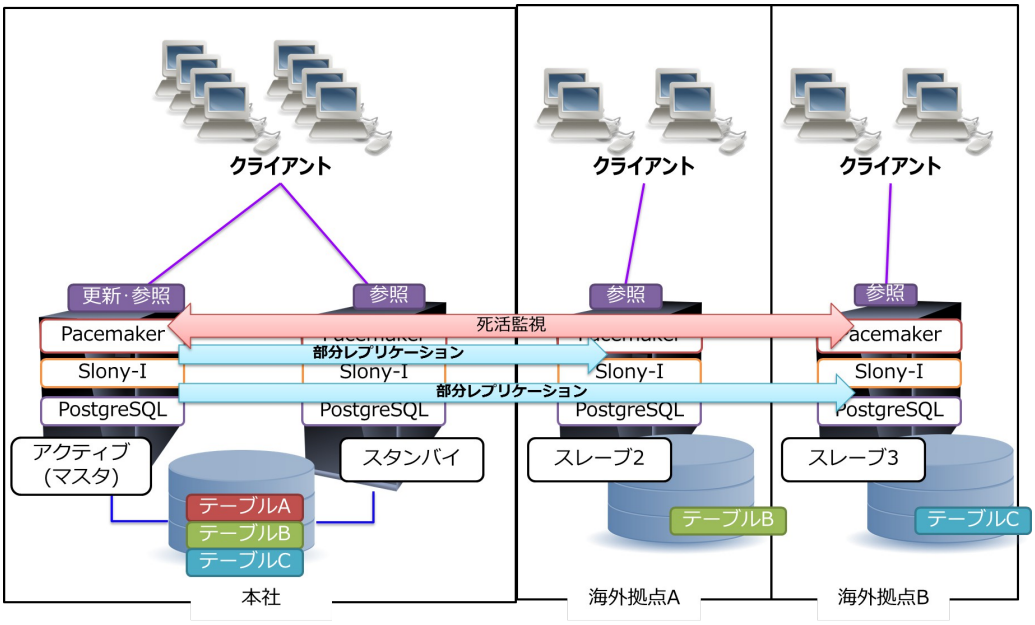

図 8.4:Slony-I を利用した部分レプリケーションを利用した参照負荷分散の構成例

4 「Streaming Replication vs Slony」
http://scanningpages.wordpress.com/2010/10/09/9-0-streaming-replication-vs-slony/

8.4. カスケードレプリケーションを利用したディザスタリカバリ構成

8.4.1. カスケードレプリケーション

カスケードレプリケーションは PostgreSQL9.2 に新しく導入された機能です。マスタサーバから全てのスレーブサーバにレプリケーションする既存のストリーミングレプリケーション機能とは異なり、マスタサーバからレプリケーションされたスレーブサーバがさらに別のスレーブサーバへと、リレーをするようにレプリケーションする機能です。これによって、複数台の PostgreSQL にレプリケーションする際に、マスタサーバのレプリケーションの負担を軽減し、さらなるスケールアウトが可能になりました。

8.4.2. 構成例

図 8.5 は地域ごとのデータベースサーバからデータが参照できるように、カスケードレプリケーションを利用した参照負荷分散クラスタの例です。

本構成は以下の要件を想定しています。

- システムの停止は業務の停止に直結するためクラスタなどで可用性を向上させる。
- 地震、火災などデータセンタの破壊が発生してもデータの安全を保証すること。
- 支店から本社のデータを参照できる必要がある。
- 本社サイトの負荷が既に高いため性能が劣化しており、負荷を分散させたい。

本構成は、マスタサーバに障害があった場合、Pacemaker が障害を検知、フェイルオーバを行うことが可能です。また、データはストリーミングレプリケーションによって、冗長化しています。

さらに、遠隔地にレプリケーションを行い災害による本社のデータセンタに問題が発生した場合にも対応ができるディザスタリカバリを実現しています。このディザスタリカバリはスレーブからカスケードレプリケーションすることで、マスタサーバへのレプリケーションオーバヘッドを最小化しています。くわえて、各支店は本社のデータを参照するために本社のデータベースに接続する必要がないため、本社サイトの負荷軽減も期待できます。

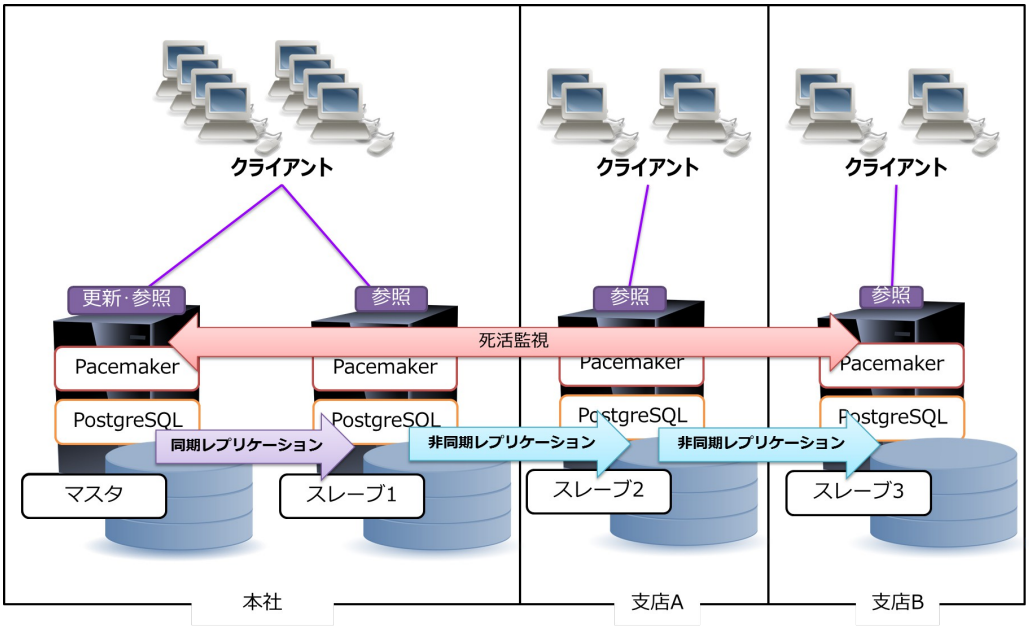

図 8.5: カスケードレプリケーションを利用した地域別参照負荷分散の構成例

8.5. Postgres-XC を利用した HA 構成

8.5.1. Postgre-XC

Postgres-XC は PostgreSQL 基盤の分散 DBMS です。Postgres-XC の代表的な特徴は以下になります。

- シェアードナッシング型クラスタ DBMS
- マルチマスタクラスタで全てのノードで更新・参照の処理が可能
- シャーディングによってデータを分散、レプリケーションすることで参照・更新ともにスケールアウトを実現

Postgres-XC は 2010 年に発表され、2012 年 4 月に Ver1.0 がリリースされた新しいマルチマスタの技術です。新しい機能がどんどん入っているところですので、今後にとても期待が持てるソフトウェアです。ただし 2013 年 3 月時点では事例がほとんどなく、まだ実際のシステムに採用する為には事前の十分な検証が必要になります。

Postgres-XC についての詳細の情報は以下の文書を参照して下さい。

- *2012 年度 WG1 活動報告書 3.4 Postgres-XC によるスケールアウト検証*

8.5.2. 構成例

図 8.6 は Postgres-XC を利用したスケールアウト型クラスタ構成の例です。
本構成は以下の要件を想定しています。

- システムの停止は業務の停止に直結するためシステムを停止できない。
- サーバやストレージの障害からデータの安全を保証すること。
- 初期導入費用はなるべく抑える。
- サービスが拡張される可能性があり、データ量、処理量の予測が難しい。

本構成は、一部のコンポーネントに障害があっても、その影響が制限的であるため残りのコンポーネントは継続してサービスを提供することが可能です。また、障害があったコンポネントは Pacemaker で検知し、フェールオーバ(GTM Proxy の場合は再起動)して復旧します。データは 2 つのサーバがお互いレプリケーションすることで冗長性をもたせます。最後の要件に関しては現在の最新版である v1.0.2 では満たしていませんが、Postgres-XC はスケーラブルクラスタを志向しているため、今後の機能追加により、最初は小規模のクラスタでサービスを始めても、処理量が増えたときスケールアウトすることで処理量を増やすことが期待できるでしょう。

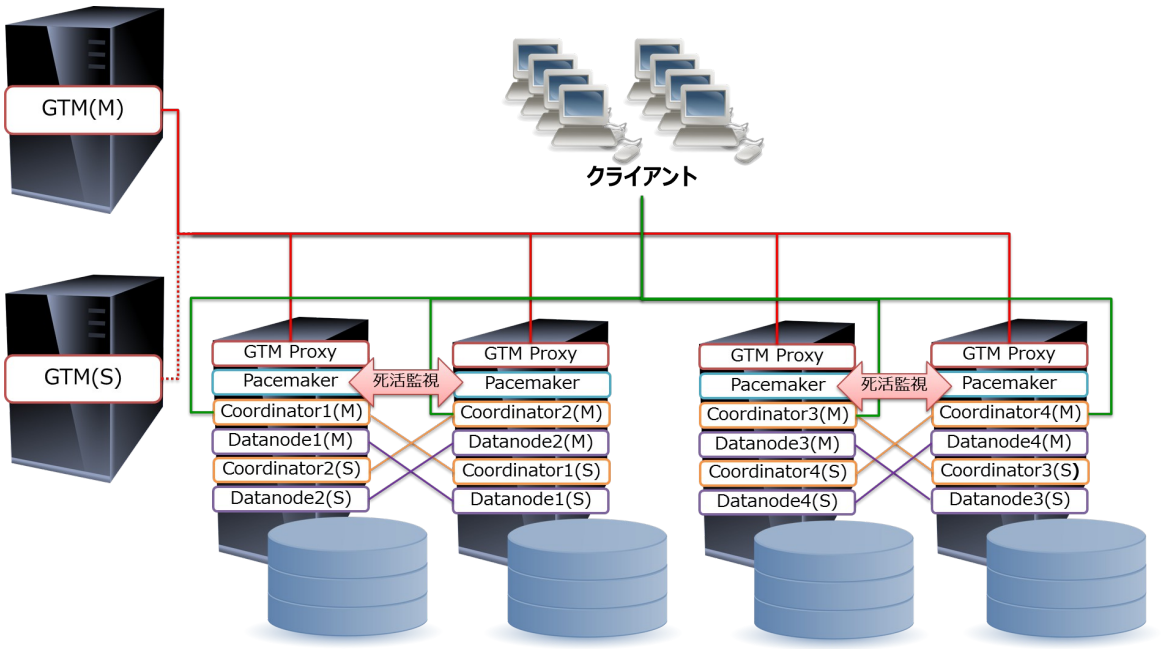

図 8.6: Postgres-XC を利用したスケールアウト型クラスタ

5

5 図 8.6 の(M)と(S)はそれぞれマスタとスレーブのことです。

9. RDBMS の対応状況

9.1. 概要

今まで見てきたデータベース構成について、各商用 DBMS の対応状況を確認してみます。。シングルサーバや HA クラスタ構成は基本的にどのデータベースシステムでも対応しています。データベースレプリケーションについてはログシッピングの対応レベルが異なり、PostgreSQL ではトランザクションの同期や一部のデータのレプリケーション機能などはクエリベースのレプリケーションで対応できているものもあります。

また、マルチマスタ負荷分散クラスタについては、同じ特徴を持ったデータベース構成はほぼ存在しないため、機能やレベルも大きく異なります。

表 9.1: 各データベースの対応状況

	シングルサーバ	HA クラスタ		データベースレプリケーション			マルチマスタ負荷分散クラスタ	
		共有ストレージ	レプリケーション	ログシッピング（同期）	ログシッピング（非同期）	クエリベース	シェアードストレージ	シェアードナッシング [6]
PostgreSQL	○	○		準同期	○	(Slony-I pgpool-II)	-	(Postgres-XC pgpool-II 他)
Oracle Database	○	○		○	○	ー	Oracle Real Application Cluster	-
Microsoft SQL Server	○	○		○	○	ー	-	(SQL Azure Federation)
DB2	○	○		○	○	ー	DB2 pureScale	DB2 Database Partitioning feature
MySQL	○	○		○	○	ー	-	MySQL Cluster

括弧つきは DBMS 本体の機能ではなく、関連する技術や別のソフトウェアと連携したものを記載しています

これらのシステムのから PostgreSQL のどの構成を選択するかを検討するに当たり、構成とその特性について、表にまとめました。記載している要件の範囲がまだまだ狭い内容ですが、移行する際の参考になれば幸いです。

6 通常のデータベースサーバを複数台立ててデータ分割して格納し、アプリケーション側で吸収する方式もありますが、本書ではシェアードナッシング負荷分散クラスタの対象外とします

表 9.2　PostgreSQL 関連での構成とその特性一覧

		シングルサーバ	HA クラスタ			pgpool-II を用いたレプリケーション	Slony-I	アプリケーションでのシャーディング	Postgres-XC
			共有ディスク	ディスクミラーリング	ストリーミングレプリケーション				
可用性	サービス継続性([7])	無し	あり	あり	あり	あり pgpool-II サーバ自体の可用性検討が必要	あり	別途クラスタを検討	あり データの冗長構成を各ノードで設定する
	ディザスタリカバリ構成	取れない	ストレージレイヤーでレプリケーションをすれば可能	取れる	取れる	pgpool-II 配下のノードのレプリケーションなどを行えば可能	取れる	各サーバのレプリケーションを取れば可能、台数が多い場合は難しい	各サーバのレプリケーションを取れば可能、台数が多い場合は難しい
性能	参照系	オーバヘッドは無し	オーバヘッドは無し	オーバヘッドは無し	スケールアウトが可能	スケールアウトが可能	スケールアウトが可能	オーバヘッドはないが結果のマージはアプリケーション側で考慮する必要がある	スケールアウトが可能
	更新系	オーバヘッドはなし	オーバヘッドはなし HA 構成の中では一元化されているメリットがある	オーバヘッドあり 同期・非同期の設定もあるが、ストレージレベルの非同期では DB の論理破壊の可能性がある	オーバヘッドあり 同期・非同期の設定で性能とトランザクション反映の信頼性のトレードオフになる	オーバヘッドあり 全ノードにクエリを発行して結果を待つ	オーバヘッドあり トリガベースで実行している	オーバヘッドはなし ただし、DBMS までの話でアプリケーション側ではシャーディングの意識が必要	スケールアウトが可能
拡張性		スケールアップで対応	スケールアップで対応 データ容量追加は行いやすい	スケールアップで対応	カスケードレプリケーションによって参照系のスケールアウトが可能	ノードを追加することで、参照系のスケールアウトが可能	カスケード接続で参照系のスケールアウトが可能	アプリケーションの設計に依存する	現段階では動的なスケールアウトはできないが将来に期待
運用・設計面		シンプルな構成であるため、運用設計項目は複雑にはなりにくい	ストレージ全体を一括管理できるメリットあり 特徴的な検討項目 ・クラスタ設計 ・ストレージの設計	以下の特徴的な検討項目がある ・クラスタ設計 ・ストレージ層の同期設計 ・障害発生後に再構築する際のデータ再同期設計	障害時の切替は比較的早い可能性、以下の特徴的な検討項目がある ・クラスタ設計 ・レプリケーションの同期や postgres の切替動作 ・障害発生後に再構築する際のデータ再同期設計	以下の特徴的な検討項目がある ・レプリケーション設計 ・ノードの追加・復旧時の同期設計	以下の特徴的な検討項目がある ・レプリケーション設計（詳細に設計できる反面煩雑） ・障害発生後に再構築する際のデータ再同期設計	独自のデータベースが複数台立っている状態であるため、煩雑になりがち また、アプリケーション側で吸収する部分が多く規模が大きくなりやすい	現段階では監視や各運用面の機能が少なく設計が難しいと想定、今後の機能拡張に期待

7　PostgreSQL ではどの構成でも、クラスタによる切替時間は必要になります。またトランザクションを保証するといったシームレスな切替はできません

初期導入コスト	シンプルな構成で一番安価	共有ディスクが必要となり比較的高価クラスタソフトも必要	汎用のサーバのみで構成を組み事が可能、クラスタソフトも必要	汎用のサーバのみで構成を組み事が可能クラスタソフトも必要	汎用サーバのみで構成が可能だが台数に依存 ソフトウェアはOSSのメリットあり	汎用サーバのみで構成が可能だが台数に依存 ソフトウェアはOSSのメリットあり	汎用サーバのみで構成が可能だが台数に依存	要件にもよるがサーバ台数が大規模になりがち
	シングルサーバ	**共有ディスク**	**ディスクミラーリング**	**ストリーミングレプリケーション**	**pgpool-IIを用いたレプリケーション**	**Slony-I**	**アプリケーションでのシャーディング**	**Postgres-XC**
		HAクラスタ						

著者

版	所属企業・団体名	部署名	氏名
システム構成調査編 第 1.0 版 （2012 年度 WG2）	日本電気株式会社		