柔軟な復旧要件を満たすデータベース接続層データ複製手法の提案

中村暢達[†,††] 藤山健一郎[†]
河合栄治[††] 砂原秀樹[††]

災害発生時に，IT システムが運用停止となっても，バックアップサイトで代替運用可能となるようなディザスタリカバリー機構に関し，その IT システムのサービスレベルに応じた，柔軟なデータ複製機能を実現する手法を提案する．具体的には，アプリケーションのデータベースへのアクセスパターンをデータベース接続層で監視し，そのパターンに応じて，データベース，OS で識別可能なマーカーを挿入することで，バックアップサイト間との同期，チェックポイントを可能とする．そのアクセスパターンを変更することで，柔軟なデータ複製を実現する．本稿では，このような柔軟な復旧要件を満たすデータベース接続層データ複製手法の構想およびディペンダブルシステムへの展望について述べる．

A Proposal of a Replication Mechanism in a Database Connection Layer with High Flexibility of Recovery Levels

Nobutatsu Nakamura,[†,††] Ken'ichiro Fujiyama,[†] Eiji Kawai[††]
and Hideki Sunahara[††]

Achieving a disaster recovery system which allows the backup site to take over the primary site's IT services while the primary site is down because of disaster is a vital issue. We propose a flexible replication mechanism for various recovery levels. This mechanism monitors the application's database access patters in the database connection layer and puts markers in the database access requests, which are recognized by the lower layers: the database layer and the OS layer. The high flexibility of the recovery levels is achieved by customizing the policy to insert the markers in the database access requests according to the various requirements such as processing speed, recovery time, and recovery points.

1. はじめに

現在，IT システムは企業活動には不可欠となっている．人為的破壊行為，自然災害等の災害に遭い，現用サイト（プライマリサイト）が運用停止となっても，別サイト（バックアップサイト）で代替可能となるような備えが重要視されている．データセンター運用企業や社会インフラ企業では，複数のバックアップサイトを用意し，それらのサイト間で常にデータを同期するように通信し，災害時には，バックアップサイトでサービスを復旧，継続できるようなディザスタリカバリー機構を構築してきている．しかし，一般的な IT システムでは，常時同期はコストがかかりすぎると言われている．その理由は，以下のものがあげられる．

- バックアップサイトにも，プライマリサイトと同じ規模の HW 機材が必要である．
- サイト間の通信で，常時同期のために広帯域の通信回線を確保する必要がある．

常時同期では，災害発生後も完全にサービスを継続するが，実際には，災害発生時にはサービスレベルがいくらか低下しても利用者は許容可能である．つまり，常時同期型でなく，各システムの必要とするサービスレベルに応じた低コストかつ柔軟なデータ複製機能を有するディザスタリカバリー機構というニーズがある．本稿では，このような普及レベルのディザスタリカバリー機構について議論する．

2. 従来のデータ複製技術

バックアップサイトにおいて，プライマリサイトのサービスを復旧する場合，バックアップサイトは，プライマリサイトと同じデータを保持している必要がある．サービスをいかに連続して，迅速に復旧できる

† NEC インターネットシステム研究所
Internet Systems Research Labs, NEC Corp.
†† 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

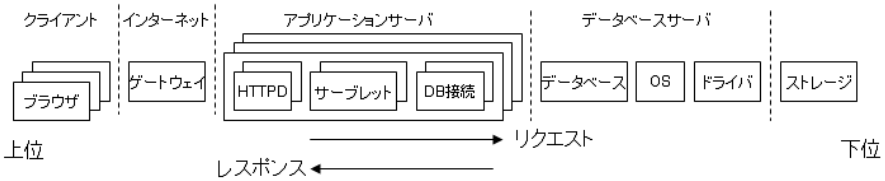


図 1 サービスの処理の流れ
Fig. 1 Service Request/Response Flow

かは，プライマリサイトからバックアップサイトへのデータ複製を，確実に，いかに遅滞なく処理するかによる．本章では，まず従来のデータ複製技術について述べる．

2.1 複製処理の位置

サービスを継続しながら，データを同期的に複製するには，サービスの処理フローの経路上のいずれかの位置でデータもしくは処理コマンドを複製する必要がある．ここで，**図 1** のように，一般的なクライアントーサーバ型の情報サービスを考える．複製元と複製先で，ストレージが同一状態を保持するためには，リクエストの処理順序が確定的でなければならない．もし，リクエストの順序が複製元と複製先で異なれば，ストレージは，異なる状態となる可能性がある．当然ながら，上位で多重にリクエストを発行したからといって，下位のストレージで，物理的にまったく同じ更新になる保証はないが，論理的に同一状態になれば，サービスの観点からは問題ない．

通常，上位層におけるリクエストの複製では処理順序は確定的とならない．例えば，クライアントからのアプリケーションサーバへのアクセス要求を多重化して，複数のサーバに送信したとしても，各サーバへの到達順序を同一となるように制御することは困難である．また，サーバ内でのアプリケーションの多くは負荷分散構成となっており，分散して並行処理された場合に，複製先と複製元で処理順序が同一であると限らない．

もし，このように処理順序が確定的でない位置で複製をすると，整合性を厳密に管理するような特別な機構が必要となり，システムの構築コストは多大となる．このような機構の適用は，金融サービスなどのミッションクリティカルな業務サービスに限定されている．

サービスの処理フローの経路上で，処理順序が確定的となる位置としては，ストレージ層，もしくはデータベース層が相当する．

ストレージ層のデータ複製には，ストレージデバイス固有のデータアクセス処理を複製する方式，電気信号的に多重化する方式などがあり，データの同期複製としては，現在最も広く使われている．しかし，通常は各サイトのストレージは同一の構成となるため，ほぼ倍の構築コストを要するという問題がある．また，ストレージアクセスのデータ・信号を同期するために，遅延やパケットロスの少ない高品質のネットワークを必要となる．

別の問題として，災害および障害発生時の復旧の信頼性問題がある．サービスは，データとプログラムの両方が必要であり，サービスを継続するためには，データが確実に保存されると同時に，プログラムとの整合性が不可欠である．つまり，障害発生時に停止したプログラムを再開するためには，再開可能な状態で保存されたデータが必要であるということである．プログラム再開に適切なデータ状態を保存するために，プログラムを一時停止することが一般的に必要とされている．

データベース層のデータ複製は，データベース内部と外部の複製に大別できる．これまではデータベース内部の複製機構が多く開発され，いくつかのデータベースには，レプリケーション機能[1)2)] が備わっており，データベース固有のトランザクション処理を複製し，バックアップサイトのデータベースに送信，データ更新を同期して行うことを実現する．しかしながら，レプリケーション機能は，高価なエンタープライズ版[3)4)] でのみの提供である．

MySQL Cluster[5)] は，４ノードまでの限定的なレプリケーション機能を MySQL に付加する技術である．この機能を利用するには，アプリケーションプログラムのソースコードを一部書き換える必要があるため，導入には手間がかかる．Postgres-R[6)] もまた，PostgreSQL にレプリケーション機能を付加するが，やはりアプリケーションプログラムのソースの書き換えを必要とする．

以上に述べたデータベース内部のデータ複製手法では，データベースの負荷が増大するという問題が大きい．現在のサーバの多くは，アプリケーションーデータベースの２層構造（もしくはウェブ層を加えた３層）であるが，アプリケーション層は負荷分散構成となる

ため，データベースがボトルネックとなるケースが増えてきているためである．

一方，データベース外部の複製に関しては，プロキシサーバを付加する手法が知られる．例えば，pgpool[7] は PostgreSQL に対するプロキシとして存在し，データベースアクセスを複製することで，レプリケーション機能を付加する．しかし，プロキシ部分が新たな単一故障点および性能ボトルネックとなる問題がある．

以上の複製技術を鑑み，低コスト，柔軟性の高い同期複製を実現するためには，データベース層データベース外部で複製する方式が有効であると考える．その理由は，(1) データベースの種類に依存しない，(2) データベースサーバの処理負荷を増大させない，(3) プログラムの変更が不要であるだけでなく，アドオンが容易である，等による．従来のプロキシ方式では，単一故障点および性能ボトルネックという課題があるため，プロキシ方式でなく，データベース接続ライブラリ拡張方式を提案する．

2.2 データベース接続ライブラリの拡張

データベース接続ライブラリとは，データベースとアプリケーションの間にあって，アプリケーションからデータベース固有の接続手順を仮想化し，データベースの種類によらずに統一的な API でデータベースアクセスを可能とするもので，ODBC[9]，JDBC[8]，ADO.NET[10] などが知られている．これらは，アプリケーション（サーブレット）が発行するデータベースに非依存のアクセス要求を，各データベース固有のアクセス要求に変換し，データベースに送信する．さらに，データベースから受信した固有のアクセス応答を，データベース非依存のアクセス応答に変換し，アプリケーションに返す．

ここで，**図 2** に示すように，データベース接続ライブラリを拡張し，アプリケーションが発行するアクセス要求を複数のデータベースに送信し，各データベースからのアクセス応答を基本的にはすべて受信し，アプリケーションに返すようにする．

これまでに，各種データベース，アプリケーションで提案方式を実装し，データ複製の信頼性，性能オーバーヘッドの評価を進めてきた[11][12]．しかしながら，実際にディザスタリカバリー機構を，通信品質が保証されないインターネットを利用して構築する場合，データ複製に遅延が生じ、処理性能が大きく低下する可能性がある．

サービスおよびアプリケーションごとに，処理性能，復旧時間（Recovery Time Objective: RTO），復旧ポイント（Recovery Point Objective: RPO）の各要

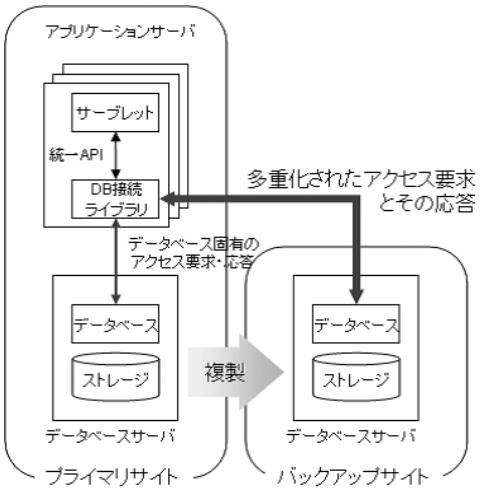


図 2 データベース接続ライブラリ拡張によるデータ複製
Fig. 2 Data Replication Using an Extended Database Connection Library

件は多様であり，信頼性を若干落としても，バックアップサイトへのデータ複製による性能低下を小さくしたいというニーズも多い．つまり，バックアップサイトへのデータ複製では，完全同期でなく，バッファリングやキャッシュを利用して処理性能の向上を図る一方，障害発生時の RTO が長くなったり，RPO が過去に遡ったりすることを許容するということである．

我々は，処理性能，RTO，RPO を可変とし，将来的には最適制御可能なディザスタリカバリー機構の実現に向けて開発を進めている。すべてのデータベースアクセスに関して同期複製する従来方式と異なり，必要なデータベースアクセスに関してのみ同期処理，あるいは処理順序を確定することで，例えバックアップサイトとの通信に，遅延の大きいインターネットを利用している場合でも，データ複製のオーバーヘッドを軽減し，スループットを向上させることを目標とする．

本稿では，１つのアプリケーションサーバと２つのデータベースサーバからなる基本構成での方式について議論する．

3. 方 式 提 案

本節において，まず提案システム構成を示し，そのシステム構成で，どのようにサービス要件に応じて RTO・RPO を可変とするか，障害・災害発生時にはいかにサービスを継続させるかについて述べる．

3.1 システム構成

提案するシステム構成を**図 3** に示す．プライマリサイトおよびバックアップサイトは，それぞれ，アプリケーションサーバ，データベースサーバから構成さ

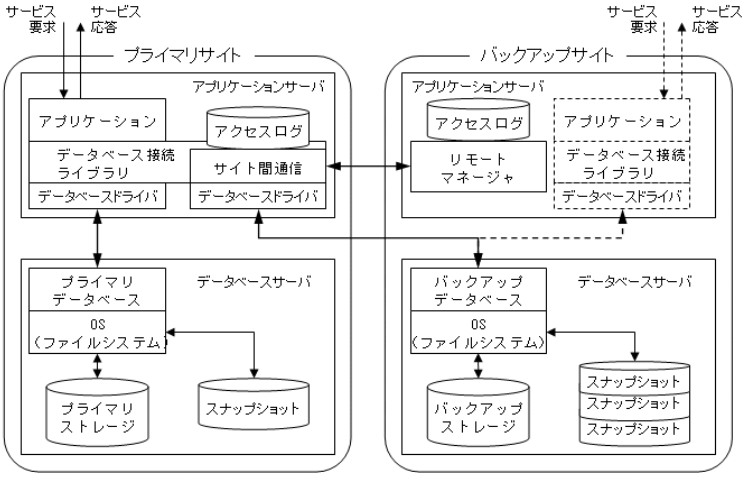


図 3 提案システム構成
Fig. 3 Proposed System Overview

れている．プライマリサイトのアプリケーションサーバには，サービス要求・応答を送受信するアプリケーション，サイト間通信モジュールを拡張したデータベース接続ライブラリ，各データベースと接続するためのデータベースドライバ，データベースサーバへのアクセスログのストレージから構成されている．バックアップサイトのアプリケーションサーバは，プライマリサイトのサイト間通信モジュールと通信するリモートマネージャ，データベースサーバへのアクセスログのストレージ，障害・災害発生時に利用するアプリケーション，データベース接続ライブラリ，データベースドライバを含む．各データベースサーバは，データベース，OS（ファイルシステム），ストレージ，スナップショット保存手段を含む．

3.2 通常時の動作

3.2.1 データ複製処理

プライマリサイトのアプリケーションサーバにおいては，データベース接続ライブラリを，標準で提供されているライブラリの API と互換であるが，その内部処理で，データベース接続を多重化するように拡張したものに置き換える．多重化されたデータベース接続の１つは，プライマリデータベース用のデータベースドライバを通してプライマリデータベースと接続し，データベースのアクセスに使われる．別のデータベース接続は，サイト間通信モジュールで管理される．基本的には，バックアップデータベース用のデータベースドライバを通して，バックアップデータベースと接続し，データベースのアクセスに使われる．このように，プライマリデータベースとバックアップデータベースに同一のアクセス要求を行うことで，複数のデータベースを論理的に同じ状態に保つようにする．

サイト間通信モジュールでは，データベース復旧のためのアクセスログをローカルに保存し，さらにバックアップサイトのリモートマネージャを介して，アクセスログをリモートにも保存する．このアクセスログの利用については，障害・災害発生時の動作の項で後述する．

次に，プライマリデータベースへのアクセスの都度，バックアップデータベースへの複製処理が行われるが，そのアクセスについて詳細に説明する．

一般的なデータベースアクセスでは，接続（connect），複数のトランザクションの実行（execute），確定（commit），切断（close）という処理の流れとなる．しかし，アクセスの種類，処理対象となるテーブルによって，同期処理，あるいは処理順序の確定が必要かどうかが異なる．本システムでは，通常処理，同期処理，順序確定処理，非処理の 4 種類にアクセスを分類し，各アクセスによって，異なるデータ複製の処理を行う．また，これらの処理の一連のかたまりをアクセスパターンと呼ぶ．アクセスパターンについては，次節において，詳細に述べる．

図 4 に，データベースアクセスの一例のシーケンスチャートを示す．データベースアクセス A〜E が順にあるとし，アクセス A は通常処理，アクセス B は非処理，アクセス C は順序確定処理，アクセス D は同期処理，アクセス E は任意の処理となっている．

通常のアクセス A は，アプリケーションからの要求があると，プライマリデータベースとバックアップデータベースの両方にアクセスを行い，プライマリデータベースの応答のみで，アプリケーションに応答

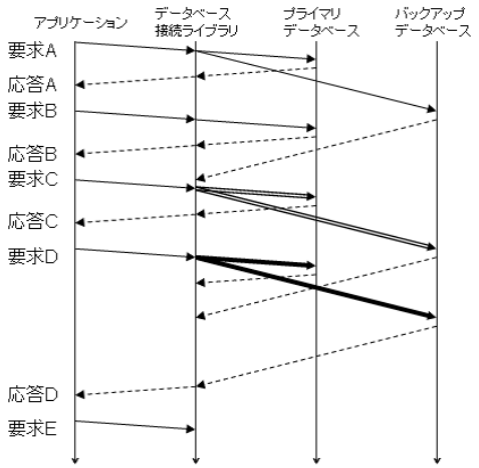


図 4 データベースアクセスのシーケンシャルチャート
Fig. 4 Database Access Sequence

A を返す．続けて次のアクセスを行う．

非処理のアクセス B では，プライマリデータベースへのアクセスのみで，バックアップデータベースへのアクセスは行わない．

処理順序確定のアクセス C は，通常のアクセスと異なり，データベース接続ライブラリにおいて，識別番号付きのアクセスであるマーカーを挿入する．識別番号は，番号サーバやトークン等で割り当てられるシーケンシャルな数値とする．この識別番号は，アプリケーションがどこまで処理を進めたかを明確に示す数値であり，障害・災害発生時の復旧動作で利用する．詳細は後述する．アプリケーションへの応答処理に関しては，通常のアクセス同様，プライマリデータベースの応答のみで，アプリケーションに応答 C を返す．

同期処理のアクセス D は，処理順序確定のアクセスと同じく，データベース接続ライブラリにおいて，マーカーを挿入する．ただし，アプリケーションへの応答処理に関しては，上記のアクセスと異なり，バックアップデータベースの応答を待って，アプリケーションに応答 D を返す．その後，アプリケーションは，次のアクセス E の処理に進む．

3.2.2 データベース・アクセスパターン

ここで，データベースアクセスにおいて，プライマリとバックアップの各データベース間で不整合となる可能性について，詳細に説明する．**表 1** は，あるアクセスにおいて，データベース接続ライブラリと各データベースとのアクセス処理の間で障害・災害発生，および各データベースのアクセス処理後にデータベース接続ライブラリがアプリケーションへの応答している間で障害・災害発生した場合，各データベース，アプリケーションが取る可能性がある処理状態の組み合わせを示す．

ケース１，ケース６は不整合がないので，その後サービスを継続しても問題ない．

ケース２の場合，アプリケーションは処理済と受信しているので，バックアップサイト側では当該アクセスを再投入する必要がある．しかし，データベースの動作や状態に影響がないアクセスであれば，当該アクセスは無視可能で，そのままサービスを継続しても問題はない．

ケース３は，プライマリデータベースは更新されているものの，アプリケーションまで反映されていない場合である．災害でプライマリデータベースが失われても，それをバックアップサイト側で反映させる必要はなく，正常状態として処理を継続してよい．

ケース４，５の場合，バックアップデータベースが更新されているが，アプリケーション側では更新されたことを知らない場合である．本来のデータベース処理では，アプリケーション側にあわせて，データベースをロールバックさせるが，現在の商用データベースの多くの commit 処理では，処理を確定したものとして継続する仕様となっている．よって，本提案システムでも，ケース４，５の状態は正常状態として処理を継続するものとする．

以上から，データベースの動作や状態に影響があるアクセスに関してのみ，ケース２の状態を回避する必要がある．すなわち，一部のアクセスに関し，バックアップサイトの処理確認を待って，アプリケーションに対して応答を返す必要がある．本提案システムでは，このようなアクセスに対応するアクセスパターンを登録し，アクセスパターンに合致する場合には，図 4 に示したアクセス D のように，バックアップサイトの応答を待って，アプリケーションにプライマリデータベースからの応答を返すようにする．

本システムでは，システム管理者は，いくつかのアクセスパターンをあらかじめ登録し，そのアクセスパターンと実アクセスとを照合することで，通常処理，

表 1 各サイトのデータベースおよびアプリケーションの更新状態
Table 1 Status of Primary/Backup Databases and a Application

	プライマリ	バックアップ	アプリケーション
ケース 1	処理済	処理済	応答済み
ケース 2	処理済	未処理	応答済み
ケース 3	処理済	未処理	未応答
ケース 4	処理済	処理済	未応答
ケース 5	未処理	処理済	未応答
ケース 6	未処理	未処理	未応答

同期処理，順序確定処理，非処理のアクセスを行う．

デフォルトは，通常処理である．非処理のアクセスパターンとしては，例えば，読み込み操作に相当するものを指定する．順序確定処理のアクセスパターンとしては，各トランザクションの排他ロック付き execute 操作等があり，同期処理のアクセスパターンとしては，commit 処理が典型的である．

アクセスパターンの登録内容を変更することで，処理性能，RTO，RPO を可変とする．同期処理の頻度が高いと，バックアップサイトの応答を待つ時間が長くなり，性能低下を招く．そこで，RPO および RTO の要件によっては，選択的にアクセスパターンを登録してもよい．

例えば，座席予約サービスシステムで，10 席の予備を確保しているのであれば，少なくとも 10 件のアクセス要求ごとにバックアップサイトと同期すれば十分である．また，コマースサイトで，24 時間以内の商品発送の規約があり，アクセスログの手動処理では，1 トランザクションあたり 10 分の処理がかかるのであれば，少なくとも 144 件のアクセス要求ごとにバックアップサイトと同期すればよい．

3.2.3 アクセスログのローテーション

本システムでは，データベースへのアクセスログを保存するが，サービスを継続する上で，累積したアクセスログをローテーションさせることが必要である．基本的には，ある時点のシステムをまるごとバックアップ（スナップショット）し，それまでのアクセスログを削除すればよい．しかし，スナップショットの実行タイミングは，アクセスログと保存したシステム状態との間で整合性を保持する必要があり，アプリケーションとスナップショット処理を同期する必要がある．以下に，アプリケーションと同期して，スナップショットを実行する方式について，**図 5** を用いて説明する．

システム管理者は，スナップショット条件をあらかじめ登録する．プライマリサイトのデータベース接続ライブラリにおいて，その条件に合致した場合，スナップショット識別子付きアクセス要求としてマーカーを挿入する．このマーカーは，データベース層および OS 層で識別できるようなものであり，例えば，あるファイルに関連付けられたテーブルの更新等である．

バックアップデータベースでは，そのマーカーオペレーションを処理するが，その処理は，OS のあるファイルアクセスと関連付けされているので，ファイルアクセスをフックし，監視することで，OS 層で検知することが可能である．そして，スナップショットを実行し，バックグラウンドでスナップショットイメージを保存すればよい．スナップショットを保存後，アクセスログにおいては，スナップショット識別子付きアクセス要求より以前のアクセスログを削除することができる．スナップショットイメージと，スナップショット識別子とを対応付ければ，複数世代を管理することも可能である．

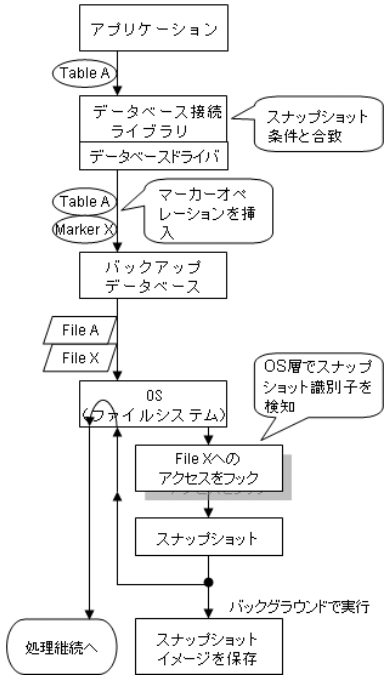


図 5 スナップショットの実行手順
Fig. 5 Snapshot Procedure

3.3 障害・災害発生時の動作

3.3.1 フェイルオーバー

プライマリサイトにおいて，障害・災害が発生した場合，バックアップサイト側で，アプリケーションを起動し，バックアップストレージのデータを用いて，サービスを継続する．しかし，これまでに述べてきたように，本提案方式では，プライマリデータベースとバックアップデータベースとで，完全に同期していない．そのため，次のようにフェイルオーバーを実行する．

まず，障害・災害を検知後，バックアップデータベースを処理が明確になっている時点までロールバックする．本提案システムでは，アクセスにマーカーを付与しているので，アクセスログに保存されている最新マーカーのシーケンシャル番号を読みとり，そのシーケンシャル番号以下となるマーカーの位置までデータベースをロールバックする．そして，アプリケーションを起動し，バックアップサイト側でサービスを再開

する．

3.3.2 再同期時の動作

バックアップサイト側に障害・災害が発生，また初期構築，ネットワーク障害，システムのアップデートなどによって，プライマリサイトとバックアップサイトのデータベース間で不整合が発生する場合がある．この場合，再同期の処理が必要となる．

軽度障害

一時的にデータベースの接続が切れたような軽度の障害の場合，再度アクセスログを順次実行するだけで十分復旧可能であり，次のような手順となる．

データベースを最新マーカー位置までロールバックする．リモートマネージャからは，そのマーカーより後のアクセスログを順次バックアップデータベースに投入する．この投入処理中でも，サービスは継続しているので，アクセスログは順次追加されている．もし，アクセスログのバックアップデータベースへの投入速度よりも，アクセスログの増加速度が大きい，もしくは速度差が小さい場合には，プライマリサイト側に通知して，アクセス要求を制限することで，アクセスログの増加を抑制する．アクセスログの投入残数がゼロとなった時点で，プライマリサイトのサービスを一時停止し，サイト間通信モジュールから，バックアップデータベースに接続し，リモートマネージャの接続を切断する．その後，通常状態の接続となる．

重度障害

ディスク障害等でデータが失われている場合，まずデータベースの再構築が必要である．バックアップサイトに保存してあるスナップショットイメージを利用するか，もしくはプライマリサイトでスナップショットイメージをバックアップサイトに転送後，スナップショットイメージを復元し，その後データベースを起動する．リモートマネージャから，データベースに接続し，復元したスナップショットに対応するマーカーより後のアクセスログを順次バックアップデータベースに投入する．その後の処理は，前述の軽度障害の場合と同様である．

4. 実　　験

4.1 実　　装

Java におけるデータベース接続ライブラリ である JDBC[8] を拡張し，前節までに述べたディザスタリカバリー機構の一部として，データ複製処理 (3.2.1 節)，データベースアクセスパターンに応じた複製制御機構 (3.2.2 節) を実装した．本節では，この実装，およびその動作について述べる．

JDBC は，Java アプリケーション，Java サーブレットのほとんどが利用している標準データベース接続ライブラリである．JDBC は，厳密には JDBC ドライバと JDBC ドライバマネージャから構成され，アプリケーションからは，標準 API を提供している JDBC ドライバマネージャが呼び出される．我々が拡張したのは，JDBC ドライバマネージャが個々のデータベース用に読み込む JDBC ドライバの部分である．JDBC ドライバの内部で，各種の拡張処理を行い，さらに個々のデータベース 用の JDBC ドライバを読み込み，各データベースへのアクセスを実現している．この拡張した JDBC ドライバを，我々は JDBC ゲートウェイと呼んでいる．

本提案方式ではアクセスパターンを登録しておき，そのアクセスパターンに合致するデータベースアクセスに関して，順序確定処理，同期処理を行うか，またはバックアップデータベースに対し非処理となる．今回，具体的に順序確定処理とした JDBC のメソッドは，Statement.executeQuery, Statement.executeUpdate であり，同期処理とした JDBC のメソッドは，Connection.commit である．

4.2 実 験 環 境

実験環境の構成を**図 6** に示す．各サーバの OS には Linux，データベースには PostgreSQL 7.4 を設定し，アプリケーションには TPC-C[14] に準拠した IBM OLTP ベンチマークツールキット V2.0 を用いた．また，プライマリサイトとバックアップサイトを想定し，その間のインターネットをエミュレーションするために，NIST Net[13] を用いた．実験では，NIST Net のパラメータを変化させて，ベンチマークツールを実行し，プライマリサイトでのみの性能（複製を行わない場合）と，pgpool[7] を用いて同期複製した場合の性能と，提案方式で逐次同期複製時の性能と，提案方式でアクセスパターンによる制御しながら複製した場合の性能とを比較した．pgpool のシステム構成においては，ベンチマークツールと pgpool プログラム間には．ローカルソケット通信を用い，２つのデータベースに接続した．

4.3 実 験 結 果

実験結果を**図 7** に示す．今回の実験では，NIST Net による片道遅延を 0, 10, 50msec と変化させ，またパケットロスを 0 %，0.1 %，1 %に変化させた．表の４列目は毎秒のトランザクション数，５列目は通常状態（複製なし）と比較したときの割合（%）である．通常状態では，毎秒 17.74 トランザクションの処理性能であった．pgpool による複製および逐次同期複製

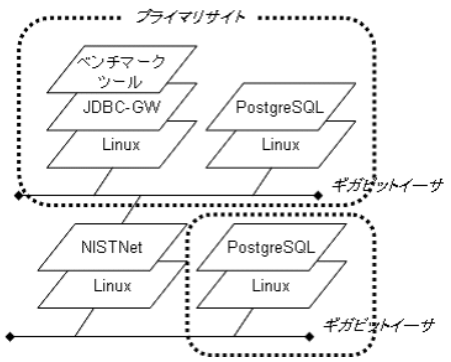
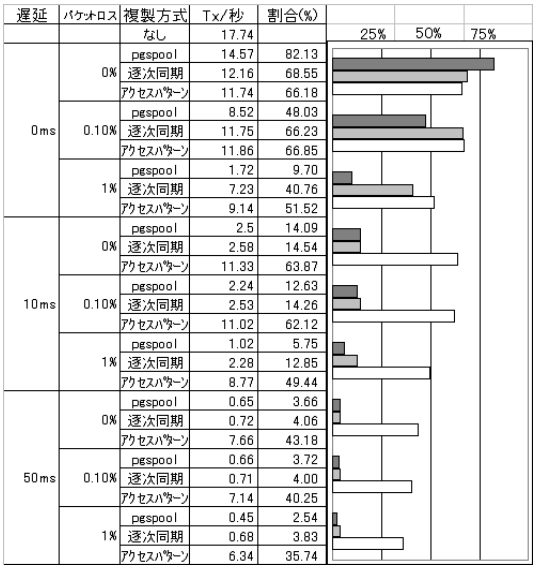


図 6 実験環境
Fig. 6 Experiment System



遅延	パケットロス	複製方式	Tx/秒	割合(%)
		なし	17.74	
0ms	0%	pgspool	14.57	82.13
		逐次同期	12.16	68.55
		アクセスパターン	11.74	66.18
	0.10%	pgspool	8.52	48.03
		逐次同期	11.75	66.23
		アクセスパターン	11.86	66.85
	1%	pgspool	1.72	9.70
		逐次同期	7.23	40.76
		アクセスパターン	9.14	51.52
10ms	0%	pgspool	2.5	14.09
		逐次同期	2.58	14.54
		アクセスパターン	11.33	63.87
	0.10%	pgspool	2.24	12.63
		逐次同期	2.53	14.26
		アクセスパターン	11.02	62.12
	1%	pgspool	1.02	5.75
		逐次同期	2.28	12.85
		アクセスパターン	8.77	49.44
50ms	0%	pgspool	0.65	3.66
		逐次同期	0.72	4.06
		アクセスパターン	7.66	43.18
	0.10%	pgspool	0.66	3.72
		逐次同期	0.71	4.00
		アクセスパターン	7.14	40.25
	1%	pgspool	0.45	2.54
		逐次同期	0.68	3.83
		アクセスパターン	6.34	35.74

図 7 実験結果
Fig. 7 Experiment Result

では，遅延の影響大きく受けるが，提案方式は遅延の影響が軽減されていることが分かる．東京–大阪間の RTT を 10ms として，トランザクション数が毎秒 10 程度のアプリケーションであれば，本提案方式を用いてシステムを東京と大阪に多重化することは十分可能である．

5. おわりに

本稿では，低コストかつ柔軟なディザスタリカバリー機構について述べた．特に，汎用的なデータベース接続ライブラリ内で，データベースアクセスを多重化すると同時に，アクセスパターンに応じて，順序確定処理，同期処理，非処理というように，バックアップデータベースへのアクセスを制御し，トレードオフの関係にある処理性能と RTO・RPO を可変とする方式を提案した．

実験では，遅延やパケットロスが発生するような環境においては，ディザスタリカバリー機構なしと比較すれば大きな性能低下はあるものの，逐次同期方式と比べれば，遅延 10msec，パケットロス 0.1 %の環境では，約 4.4 倍の性能向上を確認できた．

今後は，複数のアプリケーションへの対応を可能とするトランザクションの順序保証機構等を備え，より実用レベルの実装を進める予定である．

参 考 文 献

1) Gray, J., Shasha, D. et al.: The Dangers of Replication and a Solution, *Proceedings of the 1996 ACM SIGMOD International*, (1996).
2) Wiesmann, M., Pedone, F., et al: Database replication techniques: a three parameter classification, *Proceedings of the 19th IEEE Symposium onReliable Distributed Systems*, (2000).
3) Oracle: Oracle9i Advanced Replication, Oracle white paper, (2002).
4) IBM: DB2: Replication Guide and Reference, (1999).
5) Ronstrom, M. and Thalmann, L.: MySQL Cluster Architecture Overview, MySQL Technical White Paper, (2004).
6) Kemme, B. and Alonso, G.: Don't be lazy, be consistent: a new way to implement Database Replication, *Proceedings of the 26th International Conference on Very Large Databases*, (2000).
7) Ishi, T.: pgpool page, http://pgpool.projects.postgresql.org/, (2005).
8) Reese, G.: Database programming with JDBC and Java, O'reilly & Associates Inc, (2000).
9) Geiger, K.: Inside ODBC, アスキー, (1996).
10) Sceppa, D.: プログラミング Microsoft ADO.NET, 日経 BP ソフトプレス, (2002).
11) 藤山 健一郎, 中村 暢達, 平池 龍一: データベース同期複製のための JavaAPI の拡張, 情報処理学会第６７回全国大会, １Ｋ－５, (2005).
12) 藤山 健一郎, 中村 暢達, 平池 龍一: クラスタ対応高可用 DB トランザクション管理基盤, *SWoPP 2005*, (2005).
13) Carson, M. and Santay, D : NIST Net: A Linux-based Network Emulation Tool. *ACM SIGCOMM Computer Commununications Review*, Vol.33, No.3, pp.111–126, (2003).
14) Transaction Processing Performance Council: TPC benchmark C: sandard secification rvision 5.2, (2003).