C-031

DB 接続 API 拡張による異種 DB 間データ同期複製方式

Enhanced Database Connection API
for Synchronous Data Replication between Different Database Softwares

藤山 健一郎　中村 暢達　平池 龍一
Ken-ichiro FUJIYAMA　Nobutatsu NAKAMURA　Ryuichi HIRAIKE

1. はじめに

情報システムにおいて、障害によるデータの損失は致命的である。障害からデータを守る方法として、データを格納するストレージにおいてデータを逐次複製する、同期複製という手法がある。しかし、複製元と複製先とに物理的に同一構成の機器や、複製元～複製先間に広帯域なネットワークが必要となる。また、障害時にデータが欠落しないように、各ストレージは高速かつ高信頼の記憶装置から構成されるため、ストレージにおける同期複製は、非常に高価となる。

一方、ストレージで物理的に同期複製を行うのではなく、データを管理するデータベース（DB）において、論理的に同期複製を行う同期レプリケーション[1]という手法がある。これは、プライマリ DB が行ったデータ処理のログ等を、バックアップ DB に転送し、同じデータ処理を行い、論理的にデータの同期複製を行う技術である。しかし、同期レプリケーションは、商用 DB の高価なエンタープライズ版のみでしか提供されていない。したがって、同期レプリケーションを行っていない既存システムに、これらの機能をアドオンする場合、DB をエンタープライズ版に移行し、さらにバックアップ DB のライセンスを追加する必要がある。

そこで、最小限の構築コストで、同期複製による高い耐障害性を得たいという要求に対し、予備であるバックアップ DB には、機能等の面で劣るが、その分コストパフォーマンスの優れた異なる DB を用いるという方法が考えられる。しかし、レプリケーションは、前述のように DB 内部におけるデータ処理を複製するため、異なる種類の DB 間では、データ処理（あるいは処理ログ）の形態が異なるため、レプリケーションを行うことはできない。

そこで、本稿では、DB の種類に関わらず汎用的にレプリケーション機能を提供することで、異なる種類の DB 間でも同期複製を実現し、低コストで高い耐障害性を実現する DB 同期複製方式を提案する。

2. DB 接続 API 拡張による同期複製

2.1. 課題と提案方式

図 1に一般的なレプリケーション構成のシステムにおけるデータ処理の流れを示す。従来のレプリケーションは、DB 内部における固有のデータ処理そのものを複製するため、異なる DB 間で同期複製を行うことができなかった。それに対し、システムのデータ処理経路上において、DB の種類によらない共通部分において複製処理を行えば、汎用的にレプリケーションを実現できる。つまり、アプリケーション（AP）と DB の両方に非依存で共通部分である DB 接続 API 部分に我々は着目した。

NEC インターネットシステム研究所
Internet Systems Research Laboratories, NEC Corp.

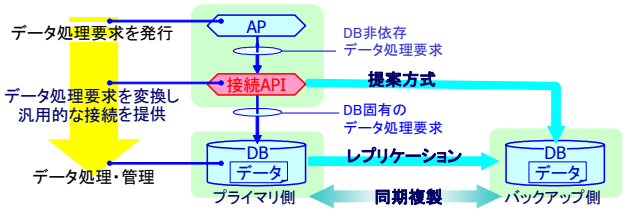

図 1：従来のレプリケーションと提案方式

DB 接続 API は、AP に標準的なインタフェースを提供し、そのインタフェースに対応する AP が発行した DB 非依存なデータ処理要求を、DB 固有のものに変換し、DB に発行する。変換部分はライブラリとして DB 毎に入れ替えることで、DB の種類を問わず、AP からの汎用的な DB アクセスを可能としている。

我々は、この DB 接続 API で複製処理を行う方式を提案する。すなわち DB 接続 API を拡張し、AP が発行する DB 非依存なデータ処理要求を、まず複製する。複製後、DB 毎に、それぞれの変換ライブラリを介して固有のデータ処理要求に変換、発行することで、異なる DB に論理的に同じデータ処理を行わせることができる。その結果、異なる種類の DB 間でのレプリケーションを実現できる。

2.2. 拡張 DB 接続 API の動作

通常時の動作（同期複製）

拡張 DB 接続 API が、AP の発行するデータ処理要求を多重化することで、DB 同期複製を行う。両 DB からのデータ処理の応答は、拡張 DB 接続 API が一旦受けて、プライマリ DB の応答のみを AP に返す。AP にとっては、通常通りデータ処理要求を発行して正常な応答が返ってくるので、複製処理は意識されない。

障害発生時の動作（データ保護）

プライマリ DB に障害が発生しても、バックアップ DB には障害発生前のデータが存在し、データ処理要求の段階で多重化されているため、障害の影響は、バックアップ DB に波及することはない。つまり、バックアップ DB のデータの整合性が保たれたまま、バックアップ DB のデータを用いて、損失なくサービスを復旧、すなわち RPO（Recovery Point Objective）＝0 を実現できる。

障害発生後の動作（フェイルオーバー）

障害発生後、プライマリ DB を処理系から切り離し、バックアップ DB にのみデータ処理要求を発行する。バックアップ DB には整合性のとれたデータが保存されているため、正常なデータ処理を行い、応答を返す。バックアップ DB の応答は拡張 DB 接続 API を介して AP に返されるため、AP にはそれがどの DB の応答かは意識されない。そのため、プライマリ DB の障害は隠蔽され、無停止、すなわち RTO(Recovery Time Objective)＝0 を実現できる。

以上のように、障害時にデータの損失なく、無停止でサービスを継続することができる。

2.3. JDBC への実装

Java における DB 接続 API である JDBC を拡張し、提案方式を実装した。JDBC は、DB 固有のデータ処理要求変換を行うライブラリである JDBC ドライバと、JDBC ドライバの読み込み、標準インタフェースの提供を行う JDBC ドライバマネージャ（JDBC-DM）から構成される。

我々はこれまで、JDBC を拡張することで、高信頼化のミドルウェアを実現[2]してきており、今回は図 2に示すような JDBC の拡張を行った。この拡張したドライバを JDBC ゲートウェイ（JDBC-GW）ドライバと呼び、通常の JDBC ドライバの代わりに JDBC-DM によって読み込まれ、さらに JDBC-GW ドライバ自身が、個々の DB 用の通常の JDBC ドライバを読み込む。

JDBC-GW ドライバは、JDBC-DM に対しては JDBC ドライバとして、JDBC ドライバには JDBC-DM として振舞うため、JDBC-DM／ドライバは一切変更する必要が無い。AP へのインタフェースである JDBC-DM、また DB へのインタフェースともいえる JDBC ドライバがそのまま使えるため、AP、DB も変更の必要は無い、すなわち既存システムを変更することなく、透過的に適用することができる。

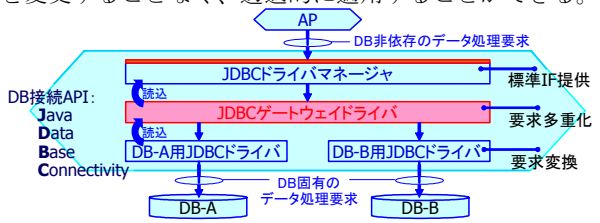

図 2：JDBC ゲートウェイドライバ

3. 評価実験

提案方式の評価では、異なる 2 種類の DB を用意し、それぞれ単一の DB からなるシステムの場合、および両方の DB からなる同期複製システムの場合とで、DB ベンチマークツールを用いてスループットを測定、比較した。

3.1. 実験環境

実験システムの構成を図 3 に示す。AP サーバでは DB ベンチマークツールとして IBM OLTP ベンチマークツールキット V2.0（OLTP-tk）を用いた。OLTP-tk は、TPC-C モデル[3]の一部である New-Order シナリオに基づき、性能評価を行う。またプライマリ DB サーバでは Oracle 10g を、バックアップ DB サーバでは PostgreSQL 7.4 を用いた。

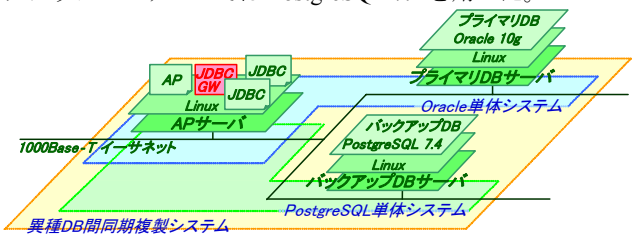

図 3：実験環境

3.2. 実験方法

OLTP-tk の実行パラメータは、スケールファクターを 10（総データ数：約 140 万件）、測定時間を 1 分とした。また、データ処理要求のトランザクション（Tx）発行間隔を 0[ms]から 1000[ms]の間で変動させて測定した。測定結果であるスループット値[Tx/sec]は、それぞれ 3 回計測した値の平均値を用いた。

3.3. 実験結果と考察

実験結果を図 4に示す。提案方式は信頼性を優先した同期複製のため、遅いほうの DB に速度が律速する。遅い DB、つまり今回の実験ではバックアップ DB と比較した場合、常に 80～90%程度のスループットが得られている。一方、速いプライマリ DB と比較した場合、Tx 発行間隔が短い状態では、両 DB の性能差のため、スループットは約 35%となる。Tx 発行間隔が長い状態では、性能差が問題とならないため、100[msec]以降は 80%以上のスループットが得られる。

現状ではフェイルオーバー時の可用性を優先し、参照データ処理要求もバックアップ側に転送している。今後、データを変更しない参照要求は、バックアップ DB には転送しないことで、バックアップ DB の負荷を軽減し、システム全体の性能低下を低減できると考える。

なお、Oracle-PostgreSQL を用いた評価実験を報告したが、Oracele 10g, PostreSQL 7.4, DB2 UDB V8.2 の各 DB 相互で同期複製の動作確認及び性能検証を行っている。

Tx間隔	各システムのスループット[Tx/sec]			同期複製系のスループット比率[%]
[msec]	プライマリ系	バックアップ系	同期複製系	20% 40% 60% 80% 100%
0	41.76	15.58	13.82	
5	26.24	14.92	12.63	
10	25.77	14.82	12.46	
20	20.98	14.45	11.73	
40	14.90	12.92	9.49	
60	11.52	10.82	7.96	
80	9.34	8.88	6.89	
100	7.86	7.46	5.95	
150	5.61	6.32	4.58	
200	4.73	4.28	3.71	
400	2.33	2.27	2.11	
600	1.59	1.56	1.47	
800	1.20	1.19	1.13	
1000	0.97	0.96	0.92	

図 4：実験結果

4. まとめ

本稿では、DB 接続 API 拡張による異種 DB 間同期複製について述べた。提案方式は、ミドルウェア層の拡張で低コストに、単一の DB からなる既存システムに同期複製機能をアドオンする。つまり、既存システムを変更することなく、元々用いられている DB と異なる種類の DB との間で同期複製を行い、システムの耐障害性を向上させる。また、本方式を Java における DB 接続 API である JDBC に実装し、評価を行った。その結果、Tx 投入密度が毎秒 10 件程度のアプリケーションサービスでは、実用上問題ないことを確認した。

AP が DB に依存したデータ処理要求を発行する場合、処理要求を各 DB に対応するよう変換する必要があるが、完全に同等な処理に変換することは困難である。しかしながら、今後 O/R マッピング等、ビジネスロジックとデータ処理ロジックの分離が進めば、このような DB 依存の問題は解消するものと思われる。

今後は、フェイルオーバー後の同期複製状態へ無停止で再復帰する機能などの開発を行う予定である。

参考文献

[1] J.Gray et al, "The Dangers of Replication and a Solution" ACM SIGMOD Conference, 1996.
[2] 藤山他, “Java アプリケーション多重実行におけるデータベースアクセス管理方式”, FIT2004
[3] Transaction Processing Performance Council, "TPC benchmark C: sandard secification rvision 5.2", 2003.