データベース・システム 84−13
（１９９１．７．１８）

データベース復旧後のAP−DBMS間同期方式

森畠秀実

ＮＴＴ情報通信処理研究所

WS／PCの低価格化、高機能化に伴い、これまでセンタの汎用機上で行われていた業務処理は、LAN上に配置されたWS／PCに分散化される傾向にある。このような環境に業務システムを構築する際には信頼性の確保が重要な課題である。しかし、WS／PC用DBMSは汎用機のそれと比べ、障害時のトランザクション保障機構が不十分であり、信頼性に欠けるという問題がある。本論文では、異種DBMS統合環境を対象とし、上記問題を解決するためのアプリケーションとDBMSの同期確認を行う手法について述べる。さらに、本手法を著者らが研究開発を行っているミドルソフト”Medi”で実現する方法について述べる。

A Method for Synchronization between Application Program and DataBase Management System

Hidemi Moribatake

NTT Communications and Information Processing Lab.

This paper describes a method to assure transactions, using a synchronization mechanism between application program (AP) and database management system (DBMS). To assure transactions is important for database systems, but the DBMS for work stations or the personal computers don't care it. This method can apply to several DBMS, because it does not depend on kinds of DBMS.

In this paper, I show the method for synchronization and how to realize this method, by using the example of "Medi", which I research and develop to realize a manager for distributed information processing systems.

1. はじめに

ワークステーション（WS）、パーソナルコンピュータ（PC）の低価格化、高機能化は、業務処理の形態に変化をもたらしつつある。すなわち、これまで汎用機を用いて行われていた業務を、WSやPCを用いてローカルネットワーク環境を構築し、分散して実行しようとする動きが強まってきている。

著者らは、このような分散環境を支援するために、WS／PC上のミドルソフトMedi [Mediator／Manager for Distributed Information Processing Systems]の研究開発を進めている。Mediは、(1)ベンダや機種による機能・インタフェースの差異の隠蔽、(2)業務処理共通機能の提供、(3)複数業務間での端末共用、を可能とするための業務共通プラットフォームの実現を狙いとしている[1]-[3]。

Mediが提供するサービスの１つであるＤＢアクセスサービスは、図１に示す様に、様々な種類の市販DBMSをサーバ上に置き、ネットワークに接続された複数の端末から利用する環境（異種DBMS統合環境）において、アプリケーションがデータベースにアクセスするためのインタフェースを共通化し、さらに、各DBMS固有のスキーマ情報を共通化してアプリケーションに提供するものである。これまでの汎用機におけるDBMSと比較した場合、WS／PC用のDBMSはシステムの信頼性という点で課題が残っている。

本論文では、システムの信頼性確保という観点からPC／WS用の市販DBMSを業務システムに利用する際の問題点を明らかにし、DBMSの種類に依存することなくこの問題を解決するための手法について述べる。さらに、Mediにおける本手法の実現方法について言及し、本手法の評価をおこなう。

2. 問題の所在

2.1 データベースシステムにおける信頼性

データベースシステムにおいて、信頼性の確保について考察する際には、

(1) 障害発生時にデータベースを復旧できること

(2) アプリケーション側が行おうとする処理が確実に実行できること

の２点について検討する必要がある。

(1)は、ハードディスクの故障等の障害が発生した場合にも、データベースの情報を復旧させることができる機能を有することであり、(2)については業務アプリケーション側から、データベースがどこまで復旧したかを認識し、どこから業務を再開すべきかを確認する機能を有することである。たとえば、データベースを利用して伝票処理を行っている最中

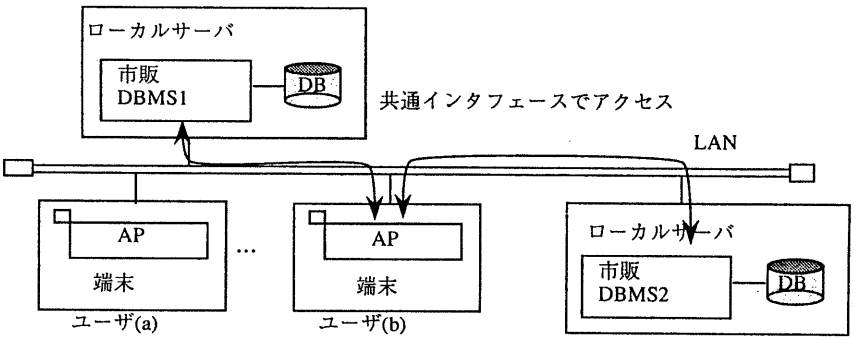

図１　異種DBMS統合環境

にデータベースの内容が破壊される、あるいは、端末側のアプリケーションがアボートする等の理由により処理を一時中断した場合を考える。データベースの内容が破壊された場合にはバックアップ・ジャーナル・ファイルの利用等によりデータベースを復旧し、端末側アプリケーションのアボートの場合にはアプリケーションを立ち上げ直した後、伝票処理を再開する。ここで、アプリケーション側が望むデータベース操作を実現するには前回どの伝票まで処理が完了しているかを確認し、次にどの伝票から処理すべきかを決定する必要がある。このように、アプリケーション側から、データベースの更新状態を認識するための機能も、データベースシステムの信頼性確保のために欠かすことはできない。

2.2 データベースシステムの信頼性の現状

上記の(1)、(2)についての現状について考察する。(1)についてはオンライン処理中のバックアップ機能やジャーナル取得機能など、リカバリ機能の拡張が進んでおり[4][5]、これらの運用によりデータベース自体の復旧のための機能は向上してきているといえる。(2)については、市販DBMSではトランザクションを単位としてアプリケーションとの同期を取っている。DBMSはトランザクションの原子性を保障し、トランザクションの途中で障害が生じた場合は直前のトランザクションが完了した時点の状態にデータベースをロールバックする。アプリケーション側は、前回完了したトランザクションの次のトランザクションから処理を再開する。

2.3 解決すべき問題点

トランザクションは、アプリケーションからの要求によるコミットの実行により完了し、完了したトランザクションはDBMSによってデータが保障される。図2にコミット要求及び完了通知の流れを示す。

この流れの中で、障害が発生した場合について以下で考察する。図3に、(i)アプリケーションがコミット要求後、DBMSによるコミット実行以前に障害が発生した場合を示し、図4に、(ii)アプリケーションが要求を出したコミットがDBMSによって実行された後、アプリケーションがコミット完了通知を受け取る以前に障害が発生した場合、を示す。これらの場合のうち、(i)についてはDBMSによって完了しているトランザクションの直後までロールバックが実行され、これまでのデータ仮更新は取り消される。(ii)についてはデータは実更新したものとして扱われるため、データの更新は保存される。したがって、(i)の場合にはこれまで行ってきたトランザクション中のデータの仮更新を再度行い、コミット要求文を再送出しなくてはならない。これに対して(ii)の場合にはこれまで行ってきたトランザクションの、次のトランザクションから実行されなくてはならない。

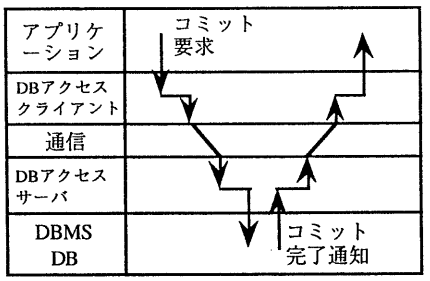

図2 コミット処理の流れ

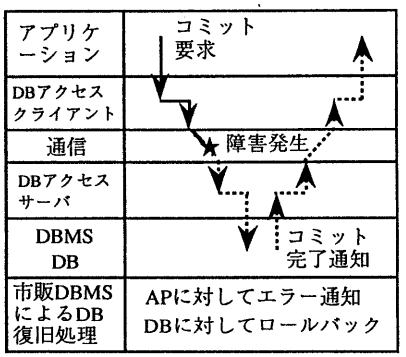

図3 コミット実行以前に障害が発生した場合

このように、(i) の場合と、(ii)の場合ではアプリケーションはデータ更新作業再開時における処理開始のトランザクションが異なるにもかかわらず、アプリケーションから完了トランザクションを確認する手段を備えていないDBMSが多い（表１）。

2.3.1 市販DBMSにおける同期確認方法の問題点

上記のような完了トランザクション確認機能を備えていないDBMSにおいては、コミット中に障害が発生した場合、データ更新処理再開時に、更新を行っていたデータベース中のデータを、データベースを利用しているユーザが参照し、実更新が行われたか否かを確認するという方法が取られている。この方法は確認結果に誤りが生じる可能性があるという問題点がある。以下に、確認結果が誤りとなる例を示す。図１に示すようなマルチユーザ環境でリモートから利用されているDBMSにおいて、ユーザ(a)とユーザ(b)がデータベースにアクセスしている場合を考える（図５）。ユーザ(a)がデータベース中のデータをAからBへと変更する処理を行い、コミット要求をDBMSへ送信したのち、アプリケーションからDBMSへの送信中に通信障害が発生した場合、データAをBへと更新する処理は仮更新の状態であるため、DBMSはタイムアウト等の手段で通信障害を検出し、ロールバッ

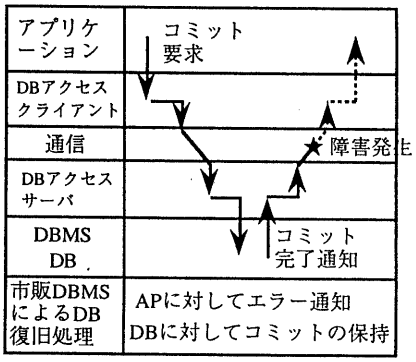

図４　コミット実行以後に障害が発生した場合

表１　トランザクション完了の確認方法

DBMS名	トランザクション完了確認方法
ORACLE	ユーザはトランザクション完了の記録を参照することはできない
SYBASE	ユーザはトランザクション完了の記録を参照することはできない
INGRES	ログスタッドというツールを用いてトランザクションの記録を参照可能

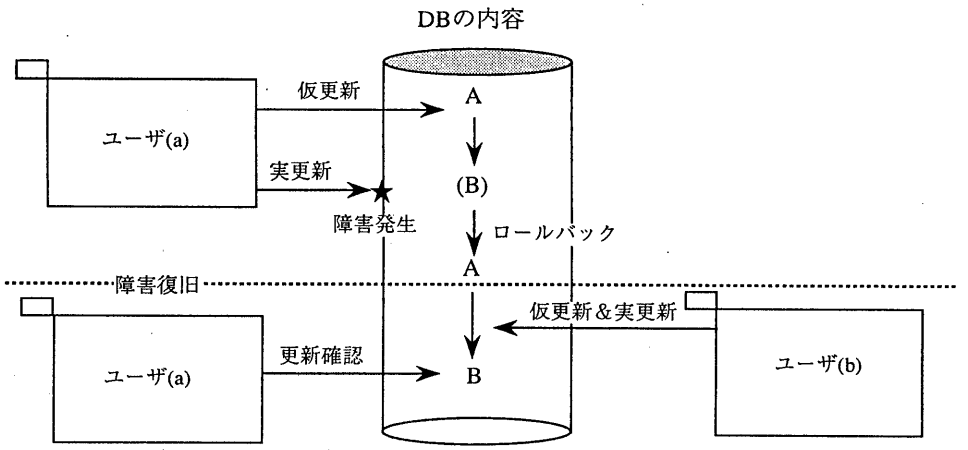

図５　更新データの確認によるトランザクション完了確認の問題点

クを実行する。ロールバックが実行されると仮更新でBへと変更されたデータはAへと戻される。ロールバックが完了した時点でDBMSはユーザ(a)によってロックされていた資源を解放する。

ここで、ユーザ(a)がデータベースにアクセスし、トランザクションが完了したか否かを確認する以前に、ユーザ(b)がデータAをBへ仮更新・実更新を行うと、ユーザ(a)は「トランザクションが完了した後の障害である」と、誤った判断を下す恐れがある。

したがって、コミット実行中に障害が発生した場合、アプリケーション側から、アプリケーションが送出したコミット要求が完了後に障害が発生したのか、あるいはコミット完了前に障害が発生したのかを確認し、障害復旧後にどのトランザクションから処理を再開しなくてはならないのかを判断するための機能を具備する必要がある。

3. 問題解決のアプローチ

3.1 同期確認機能実現における条件

上記の問題を解決するためには、完了トランザクションを明確化し、アプリケーション-DBMS間の同期確認機能を実現する必要がある。ここで、異種DBMSを利用する際にアプリケーションに対して共通プラットフォームを提供するという観点から、

(1) 特定のDBMSに依存せず、どのDBMSに対しても適用可能である

(2) DBMS自体に変更（改造）を行わない

という条件の基で本機能は実現されなくてはならない。

3.2 アプローチ

同期確認機能を実現する際に、著者はトランザクションはDBMS中においては必ず完結しているという点に着目し、アプリケーション側にどのトランザクションまで実行が完了しているかを通知することができれば同期確認機能が実現できると考えた。シングルユーザの場合は更新を行っていたデータベースを参照することで同期確認が可能であるが、上記のように、マルチユーザの場合にはどのアプリケーションによってデータ更新が行われたかを知ることが困難であるために同期確認ができなくなる。したがって、アプリケーションとは独立に、どのアプリケーションによるデータ操作かを識別できる情報をSQL等の言語でデータベース内に書き込むことにより、3.1に示した条件を満たしながら、かつアプリケーションとDBMSの同期確認を行うことができると考え

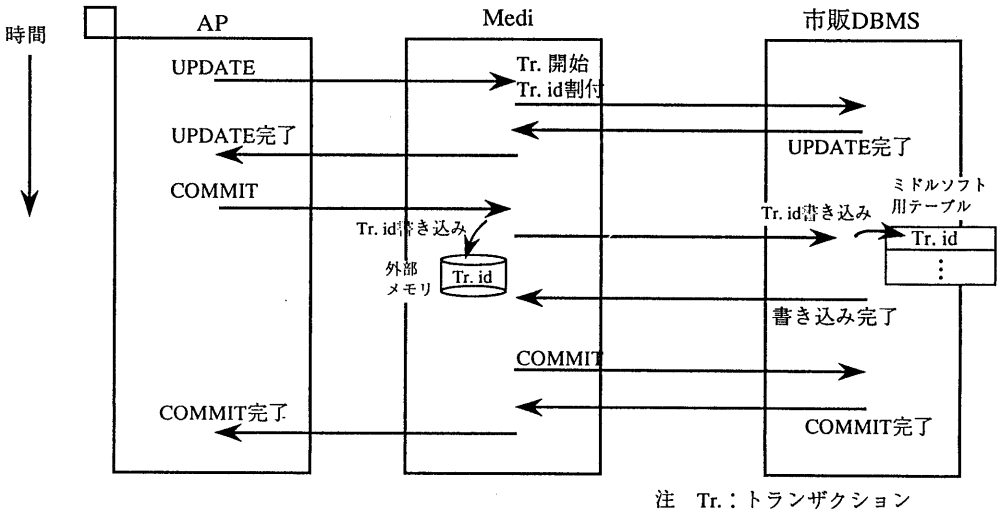

図6 確認機能のシーケンス

られる。

3.3　同期確認手法

以下に、アプリケーション・プログラムとDBMS間の同期確認手法を示す。

図6に確認機能動作時のシーケンスを示す。アプリケーションがトランザクションの開始を明示的あるいは暗示的に示すと、各々のトランザクションは識別子が付与され、アプリケーションに対して付与されたトランザクション識別子を通知される。

コミットの要求を行った時点でトランザクション識別子の情報が外部メモリに書き込まれ、さらにユーザがデータを書き込むのと同様な方法で、データベース内にトランザクション識別子情報が書き込まれる。

このような仕組みを提供することにより、トランザクションが完了後に障害が生じた場合（コミット完了後）、データベースの内容はDBMSによって保障されているため、書き込まれたトランザクション識別子が残っている。これに対して、トランザクション完了前に障害が生じた場合はトランザクション識別子情報はロールバック処理によって失われる。

処理再開時には、ユーザ・アプリケーションからデータベース内のトランザクション識別子を参照し、識別子が保存されているトランザクションについてはコミットが完了したものと認識し、次に行うべき処理を決定する。

4．Mediでの実現方式

Mediを用いて本機能を実現するためのシステム構成図を図7に示す。

制御部はアプリケーション・プログラムからの要求が次のうちのどれであるかの判断を行う。

(1)　トランザクション開始要求

(2)　データベース操作要求

(3)　コミットメント要求

(1)の場合、制御部はトランザクション識別子生成部にトランザクション開始の通知を行う。トランザクション識別子生成部は外部メモリに格納されているトランザクション識別子を参照し、開始されたトランザクションに対して新たなトランザクション識別子を付与し、外部メモリに格納する。表2に外部メモリに格納される情報の例を示す。

(2)の場合、制御部はDBMSに対して、アプリケーション・プログラムから送られてきたデータベース要求をDBMSに応じたプロトコルに変換した後送信する。DBMSは送られてきた要求にしたがって、データベースにアクセスし、ユーザ・テーブルの更新

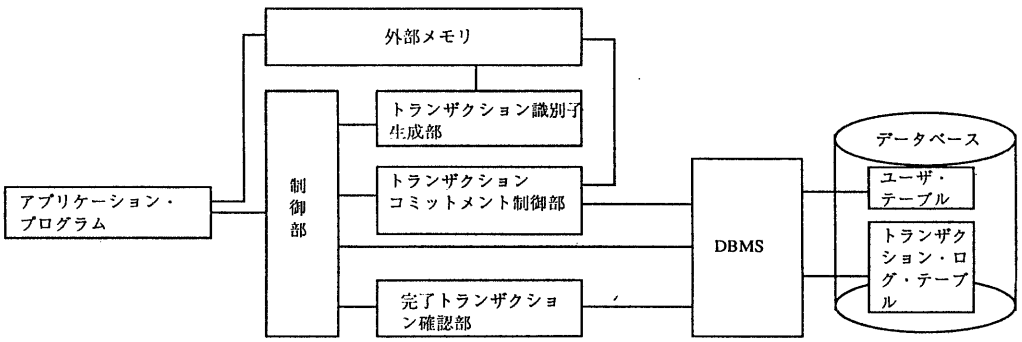

図7　同期確認システム構成図

作業を実行する。

(3)の場合、制御部はトランザクション・コミットメント制御部に対してコミット要求を送出する。トランザクション・コミットメント制御部は、外部メモリに格納されているトランザクション識別子を参照し、コミット要求されているトランザクション識別子をトランザクション・ログ・テーブルに書き込むためのコマンドを発行する。

DBMSはトランザクション・ログ・テーブルに、指定されたトランザクション識別子の書き込みを行い、トランザクション・コミットメント制御部に対して書き込み完了を通知する。通知を受けたトランザクション・コミットメント制御部は、コミット要求通知をDBMSに対して発行する。コミット要求を受け取ったDBMSはデータベースに対してコミットを実行し、コミット完了通知をトランザクション・コミットメント制御部に送信する。コミット完了通知は制御部を経由してアプリケーション・プログラムに通知される。トランザクション・ログ・テーブルはテーブルのロックされる時間を最小限に抑え、テーブルへの書き込み時間を短縮するために、アプリケーション毎に生成される。トランザクション・ログ・テーブルに格納される情報の例を表3に示す。

アプリケーション・プログラムによるコミット要求中における障害等発生後の処理再開時に、以下に示す方法で同期をとる。

(1) アプリケーション・プログラムは外部メモリに格納されているトランザクション識別子からアプリケーション・プログラムが送出したトランザクション識別子の情報を得ると共に、確認部に対してトランザクション・ログ・テーブルの内容を参照するためのSQLを発行する。

(2) 確認部は、要求文をDBMSに応じた形式に変換し、DBMSに対してSQLを用いてトランザクション・ログ・テーブル検索要求を行う。

(3) DBMSはトランザクション・ログ・テーブルの検索を行い、その結果を確認部に送信する。

表2 外部メモリ格納情報の例

ユーザ・アプリケーション名	トランザクション識別子	トランザクション開始時刻
AP-1	Tr-1	9:15:20
AP-1	Tr-2	9:16:00
AP-2	Tr-1	9:17:35
AP-1	Tr-3	9:18:02
⋮	⋮	⋮

表3 トランザクション・ログ・テーブル格納情報の例

トランザクション識別子	トランザクション開始時刻
Tr-1	9:15:20
Tr-2	9:16:00
Tr-3	9:18:02
⋮	⋮

(4) 確認部ではDBMSの種別に依存しない共通インタフェースを通じでアプリケーション・プログラムに完了トランザクション識別子を通知する。

(5) 通知を受けたアプリケーション・プログラムは外部メモリに格納されているトランザクション識別子とトランザクション・ログ・テーブルに格納されているトランザクション識別子を比較する。

(6) 外部メモリに存在して、トランザクション・ログ・テーブルに存在しないトランザクション識別子がある場合は(7)を実行する。その他の場合は(8)へ。

(7) 外部メモリに存在して、トランザクション・ログ・テーブルに存在しないトランザクション識別子を持つトランザクションを再実行する。

(8) 処理中断前に行っていたトランザクションの、次のトランザクションから処理を再開する。

5. 評価

本論文では、Mediがトランザクション・ログ・テーブルをデータベース内に生成することによって、トランザクション完了確認によるアプリケーション-DBMS間同期確認機能をDBMSの機能に依らず、共通のインタフェースで行う手法を示した。

本手法は、DBMS自体をカスタマイズすることなく、しかもDBMSが一般的に有しているテーブルの生成、書き込みという機能で実現されているため、種々のDBMSに適用可能である。このことを、ORACLE、informix、INGRES、SYBASEの４種類のDBMS上に同期確認機能を構築することによって確認した。これによって、アプリケーション・プログラムのポータビリティを確保しつつ、データベースアクセスの信頼性を向上させることを可能とした。

本手法における問題点は、外部メモリ及び、トランザクション・ログ・テーブルにトランザクション識別子を書き込むための時間がかかるという点である。書き込みに要するオーバーヘッドは、１０回のデータベースに対する挿入、更新処理に対して数％（informixの場合の実測値）であり、通常の業務に適用しても実効上の問題は無いと考える。

6. おわりに

端末ミドルソフト”Medi”において、異種DBMSに対して共通インタフェースでアクセスし、アプリケーション・プログラムのポータビリティを確保しつつ、アクセスの信頼性を向上させる手法について述べた。

これまで、アプリケーション・プログラムが発行したコミット要求がDBMSに届く以前に障害によって失われた場合、アプリケーション側から実行されたコミットの履歴を参照する共通の手段がないために、処理再開時におけるアプリケーション・プログラムが開始するトランザクションの断定が困難であり、信頼性向上の妨げとなっていた。そこで、アクセスの信頼性向上のために、Mediでは既に実行されたトランザクションをアプリケーションから確認するための機能を実現した。本機能はDBMSが持つ基本的な機能のみを用いて実現されているため、異種データベース統合環境において、データベース操作再開時の同期誤りを防ぎ、データベースシステムの信頼性向上を図ることが可能となる。

今後の課題としては、確認機能の自動化が挙げられる。

謝辞

本研究を行うにあたり、貴重な御意見、ご討論を承りましたＮＴＴ情報通信処理研究所・分散処理研究グループの方々に深くお礼を申し上げます。

参考文献

[1] 森畠、竹内：”端末ミドルソフトにおけるMML（Micro-Mainframe Link）機能の仮想化”，情報処理学会第42回全国大会, Apr, 1991

[2] 菅沼：”垂直分散システムにおけるホストネイティブプロトコルの仮想化”：情報処理学会第41回全国大会, Sep, 1990.

[3] 菅沼、他：”WSをベースとした新しい情報システムづくりを支援”：NTT技術ジャーナル, Jun, 1991.

[4] ORACLEデータベース管理者ガイド Ver. 6, 1990

[5] INGRESデータベース・マニュアル Ver. 2, 1989