分散処理システム3－10
（1979. 11. 22）

分散データベースにおける障害処理とデータコンシステンシイ維持に関する一考察

A Consideration on Crash Detection/Recovery Scheme and Preservation of the Data Consistency for Distributed Data Base

山崎晴明　川上英　松下温
Haruaki YAMAZAKI　Suguru KAWAKAMI　Yutaka MATSUSHITA
沖電気工業株式会社
OKI Electric Industry Co., Ltd.

1．序

近年の通信ネットワーク技術の進展は、コンピュータ処理の様々な分野においても、従来の集中型データベースに比し、数多くの利点を持つ分散データベースというアプローチを可能とするに至っている。

CCA（Computer Corporation of America）社のJ. Rothnie 等は、このような分散データベースの持つ利点として、1)高信頼性，2)通信コストの節減、3)拡張の容易性を指摘しているが〔1〕、解決すべき技術的問題点も数多く残されている。特に分散データベースにおいて、データが重複して配置された場合、これらのデータのConsistencyを維持する方式は、分散データベースにおける主要な技術的問題の一つとなっており、そのための方式も数多く提案されている〔2〕〔3〕〔4〕〔6〕〔8〕〔9〕。本稿では、特に分散データベースにおける障害生起時に、このようなデータのコンシステンシイを維持する方式について一考察を行う。

この問題で特に重要なことは、ネットワークパーティションニングが生起したか否かの判定である。一般にパーティションニングが生起したか否かの判定、および生起したとすればどのようなネットワークの分割が生じたかを診断するプロトコルは、複雑なものでありまた通信コストも大きなものとなる〔6〕。分散データベースが大規模になればなる程、単一サイトの障害をもすべて潜在的なパーティションとして扱っていたのでは、通信コストは膨大なものになってしまう。本稿で述べる障害の検出／復旧方式は、このようなパーティションの判定を効率良く行い、通信コストをできる限り少くすることに焦点をあてている。

2．コンシステンシイ維持のための同期制御

本節では、障害処理を論ずる前に、正常処理として実行されるプロトコルについて、この概略を述べる。

2.1. 分散データベースのモデルおよび用語の定義〔8〕〔9〕

データベースモデルとしては、リレーショナルモデルを想定する。次に、リレーションのリストリクションをとったものをフラグメントと呼び、このフラグメントが分散配置されるデータの単位とする。分散ネットワークのノードをデータベースサイトと呼ぶ。したがって各データベースサイト内には、いくつかのフラグメントが保持される。また、あるフラグメントは、いくつかのサイトで重複して保持されるものとする。フラグメントの更新は、各サイト内に生成されるフラグメントプロセスというプロセスにより実行される。このフラグメントプロセスは、対（フラグメント識別子，サイト識別子）によって識別されるものとする。

同一のフラグメントではあるが、異なるサイト内に冗長に格納されたフラグメントに対するフラグメントプロセスの集合のことを、

“クローズド・アップデート・グループ”と呼ぶ。

一般に、複数のフラグメントを更新するキュエリが与えられると、それはトランザクションと呼ばれる各フラグメント毎のオペレーションに分割される。このとき、クローズド・アップデート・グループ内のすべての重複フラグメントはすべて同一の値を持たなければならない（シンタクティック・コンシステンシイ）、また各フラグメント間の意味的なコンシステンシイも正しく維持されなければならない（セマンティック・コンシステンシイ）[8][9]。このとき、異なるフラグメント間の意味的なコンシステンシイを保つため形成されるグループを“リレーティド・アップデート・グループ”と呼ぶ。図2.1に2つのアップデート・グループ間の関係を例示する。

ある会社の従業員の給料を10%上げるというキュエリがターミナルから与えられたとする。このユーザからのリクエストを受付けるプロセスをソースと呼ぶ。今、この従業員リレーションが、東京地区，名古屋地区，大阪地区という3つのフラグメントに分割されており、それぞれが複数サイトに重複して保持されていたとすれば、図2.1のような3つのクローズド・アップデート・グループが形成されることになる。ソースは、与えられたキュエリをトランザクションと呼ぶ各フラグメント毎のデータ操作に分解する。次に各クローズド・アップデート・グループからマスタと呼ばれる代表フラグメントプロセスを選び、トランザクションをそのマスタに送出する。ここでクローズド・アップデート・グループ内の他のフラグメントプロセスは“スレーブ”と呼ばれる。

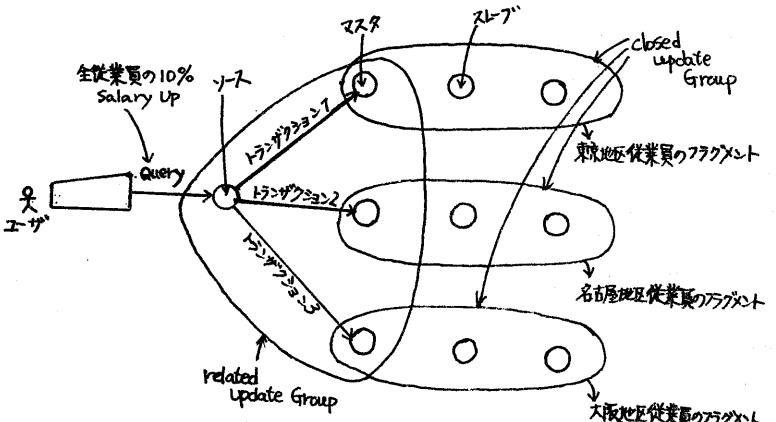

図2.1 Related Update Group と Closed Update Group の構成例

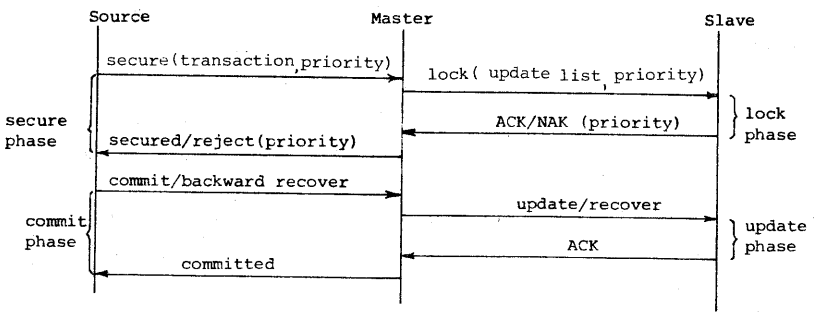

図2.2 階層型同期制御方式

2.2. 同期制御方式の概念

“ソース”と“マスタ”および“マスタ”と“スレーブ”間で行なわれるインタラクションは、図2.2に示すように“2-phase commit”法に基づいたものとなっている[7]。

まずソースは，与えられたキュエリを処理するためには、どのようなリレーテッド・アップデート・グループを構成すればよいかを決定し、各マスタにソース・メッセージを送る。このときソース・メッセージには、そのクローズド・アップデート・グループで実行すべきトランザクションおよびそのプライオリティ、リレイテッド・アップデート・グループのメンバ構成リスト（障害生起時に使用

される）等が含まれる。なお、このときのソースによるマスタの選出方法については、文献[8][9]を参照されたい。ソース・メッセージを受信したマスタは、トランザクションで示された操作を実行することにより、変更すべきタプルのリストである更新リストを作成する。ただし当該マスタが、既に別のトランザクションを受け取っており、現在それの実行途中であれば、そのトランザクションのプライオリティを“リジェクト”メッセージに付加してソースに返送する。

更新リストを作成し終えたマスタは、それをトランザクションのプライオリティとともに“ロック”メッセージに付加して、自身のクローズド・アップデート・グループ内の全スレーブに送信する。

このロック・メッセージを受信したスレーブは、フラグメントをロック状態とし、更新リストを一時記憶域に格納した後マスタにACKを応答する。もしロック・メッセージ受信時に、スレーブが既に他のマスタからのロック・メッセージを実行中であれば、そのプライオリティをNAK応答に付加してマスタに送信する。

すべてのスレーブからACKを受信したマスタは、“セキュア”メッセージをソースに返送する。もし一つでもNAKが返送されたのであれば、マスタはNAKに付加されたプライオリティと、自身がソースより渡されたトランザクションのプライオリティとを比較する。もし自身のプライオリティが高ければ、マスタはNAKを返送したスレーブに対してロック・メッセージを送出し続ける。もしNAKによって報告されたプライオリティの方が高ければ、マスタはソースに対しそのプライオリティをリジェクト・メッセージに付加して報告する。さらにマスタは、既にACKを返送してしまっているスレーブに対しては、更新リストの放棄とロック状態の解除を指示する“リカバ”メッセージを送出する。

ソースは、すべてのマスタから“セキュアード”メッセージが返送されたことを確認して、各マスタに“コミット”メッセージを送る。一方、一つでもリジェクト・メッセージを返送したマスタがあれば、他のすべてのマスタに対し“バックワード・リカバ”メッセージを送り、更新リストの放棄とロックの解除を指示する。

コミット・メッセージを受信したマスタは、スレーブに対し、更新リストの内容を2次記憶中に書き込むように指示する“アップデート”メッセージを送出する。このとき、マスタ自身の保持するフラグメントも更新リストの内容によって書き替える。

更新が終了すると、各スレーブは、マスタにACKを返送する。すべてのスレーブからACKが返送されたことを確認後、マスタはソースに“コミッテッド”メッセージを送出する。

3. 障害の検出と復旧

重複データを持つ分散データベースにおいては、サイトの障害が発生した場合、いかにしてデータのコンシステンシイを保ちつつシステム全体のサービスを続行してゆくかは、重大な問題であり、そのためのプロトコルも種々のものが提案されている[3][4][5][6]。

INGRES[3]では、障害生起時に、“Reconfigure”メッセージをブロードカストすることにより、システムの再構成を計っている。しかしながら一般に、全データベースサイトへのメッセージのブロードカストは、通信コストが大巾に増加する要因であり、できる限り少なくすることが望ましい。またこのような系の再構成プロトコルは、ネットワークのパーティショニングが生起したときに特に必要なものであって、たとえば一つのサイトの障害が検出された場合にもパーティショニングの可能性ありとして、このようなプロトコルを実行することは好ましいことではない[6]。

また[6]の方式では、ネットワークの中に一つの“primary site”を定めて冗長データの更新に関する管理を一括して行なっており、さらに系の再構成は、“primary site”を起点としてメッセージのチェイニングによる伝送方式をとっているため、分散システムの利点を充分生かしきれなく、たとえば、“primary site”とは遠く隔ったサイトからアクセスした場合には、遅延が大きいといった難点を持っている。

これらの問題を解決するため、本稿で考察した障害検出および復旧方式では、まず通信コス

トをできる限り少くし、またエンド・ユーザに対する応答性も良くすることに焦点をあてている。

このため、まずネーに考察したことは、ネ2節で述べた正常シーケンス中に組み入れるべき障害と、系の再構成を必要とする障害とを分離することであった。こうすることにより、たとえば単一サイトで障害が生起した場合には、特別の障害処理のためのプロトコルを実行させる必要がなく、通信コストの増加を押えることができ、またエンド・ユーザに対する応答性も向上させることができる。次に考察したことは、正常シーケンスの実行主体がフラグメント・プロセスであったのに対し、系構成プロトコルの実行主体を各サイトに一つ存在する診断マネジャとする方式である。こうすることにより、系構成プロトコル実行時のデータ通信量も、フラグメント・プロセスが実行主体である場合に比して大巾に節減されることとなる。さらに、診断マネジャどおしの通信は、アップデート時に使用されるコネクションとは別の、システム確立時に設定される診断コネクションを介して行なわれ、正常時のシーケンスとは独立に実行され得るという形態をとっている。

3.1 障害の検出

正常シーケンス実行時に、各サイトの障害は、フラグメント・プロセスが実施する応答時間の監視によって検出される。各フラグメント・プロセスが監視する事象を、表3.1に示す。

表3.1 時間監視

監視主体	監視事象	
	順方向	逆方向
スレーブ		・ロック・フェーズの終りからアップデート・フェーズの開始まで
マスタ	・ロック送出から応答受信まで ・アップデート送出から応答受信まで	・セキュア・フェーズの終りから、コミット・フェーズの開始まで
ソース	・セキュア送出から応答受信まで ・コミット送出から応答受信まで	

3.2 障害検出処理

3.2.1 スレーブの障害

正常シーケンス実行途中のスレーブの障害は、マスタによる応答時間の監視およびスレーブ自身の逆方向監視によって検出される。このときのスレーブの状態には、アップデート実行前と後の2状態があるが、単一のスレーブの障害の場合には、いずれの場合も、マスタはその スレーブから応答が返されたものとして、正常シーケンスの実行を継続する。

複数のスレーブのタイムアウトを検出したマスタは、ネットワーク・パーティショニングが生起した可能性が非常に高いためまだアップデート・フェーズに入っていなければ、バックワード・リカバを、入ってしまっていればコミッテッドをソースに送出した後、後述する系再構成プロトコルの実行を、自サイトの診断マネジャに依頼する。

一方、逆方向監視タイマのタイムアウトにより異常を検出したスレーブは、同一クローズド・アップデート・グループ内の任意のフラグメント・プロセスに対し、現在の状況を問合せる。この問合せに対する応答には、次の3種が考えられる。

1)、リカバリを実行した。
2)、アップデートを実行した。
3)、その他タイムアウトにより異常を検出した、または時間監視を実行中等

このうち、1)であれば、当該スレーブは、ロック状態を解除し、更新リストを放棄する。2)であれば、当該スレーブは、更新リストを用いてアップデートを実行する。3)であれば、問い合せの相手を変えて同一処理を実行する。すべてのスレーブが3)であれば、当該スレーブは、リカバリを実行する。

3.2.2 マスタの障害

正常シーケンス実行途中のマスタの障害は、ソースによる応答時間の監視およびマスタ自身による逆方向監視によって検出される。このときのマスタの状態には、コミット・メッセージの処理を実行する前と後とがある。もし一つでもコミット・メッセージを実行したマスタが存在する可能性があれば、ソースは代替マスタを選択して、コミット・メッセージを全マスタに

送出する。一方もしコミットメッセージ実行前であれば、ソースはマスタ選択リストを変更した後、バックワード・リカバを全マスタに指示し、キュエリを放棄する。

複数のマスタのタイムアウトを検出したソースは、ネットワーク・パーティショニングの生起した可能性が非常に高いため、もしコミット・メッセージ送出前であれば、全マスタにバックワード・リカバを出した後、そうでなければ直ちに処理を中止して系再構成プロトコルの実行を自サイトの診断マネジャに依頼する。

一方、逆方向監視により、異常を検出したマスタは、同一リレーテッド・アップデート・グループ内の任意のメンバに対し、現在の状況を問い合せる。

この問合せに対する応答には、次の3種が考えられる。

1)、バックワード・リカバリを実行した。

2)、コミットを実行した。

3)、その他

このうち1)であれば、該マスタは、そのクローズド・アップデート・グループのスレーブに対しリカバリを指示し、自身もロック状態を解除し、更新リストを放棄する。2)であれば、アップデートをまたスレーブに送出し、自身も更新リストを用いてアップデートを実行する。3)であれば、問合せの相手を変えて同一の処理を実行する。すべてのマスタが3)であれば、該マスタはバックワード・リカバリの処理を実行する。

3.2.3. ソースの障害

正常シーケンス実行途中でソースが障害となった場合、一般にリクエストを発生したユーザ・プロセス自体が、それ以上処理を続行することが不可能となるため、シーケンスは中断する。

ただし、コミット・フェイズにおいて、ソースが障害となった場合、複数のマスタ間でインコンシステントな状態が生起する可能性がある。

このような状態は、マスタによる逆方向監視により検出され、3.2.2. で述べた処理がマスタによって実行されるため、コンシステンシイが維持されることになる。

3.3. 系再構成プロトコル

正常シーケンス実行中に複数サイトの障害を検出したフラグメント・プロセスは、系再構成プロトコルを実行するため、自サイト内に一つ存在する診断マネジャに対しリクエストを発生する。リクエストを受けた診断マネジャは、他のサイトの診断マネジャと伴に系再構成プロトコルを実行する。なお各診断マネジャは、エラーサイト・リストと呼ばれるその時点で障害のため運用を停止しているサイトのリストを保持している。このときの診断マネジャによるプロトコルの実行は、次のようになる。

1). フラグメントプロセスは、障害が発生したと判断されるサイトのサイト識別子とフラグメント名を診断マネジャに渡し、系再構成プロトコルの実行を要求する。

2). リクエストを受けた診断マネジャは、診断メッセージを送ることにより指定されたサイトにある診断マネジャとの通信を試みる。

3). この診断メッセージを受け取った診断マネジャは、指定されたフラグメントプロセスに対し、障害からの復旧手順を実行するよう指示した後、送信してきた診断マネジャに対しACKを応答する。

4). 診断メッセージを送信したすべての診断マネジャからACKを受信した場合、その診断マネジャは、要求元フラグメントプロセスに対し、パーティショニングが生起していないことを通知し、処理を終了する。

5). ACKを応答しない診断マネジャがあった場合には、パーティショニングが生起していると判断し、ステップ6)以下で示すような、自サイトが、分割されたネットワークのマジョリティ側に位置するか否かを判断する処理を実行する。

6). ACKを応答しなかったサイトには障害が生起しているものとみなし、エラーサイト・リストを更新して、全サイトの診断マネジャにそれをブロードカストする。

7). このメッセージを受信した各サイトは、エラーサイトリストを新しいもので置き換え、ブロードカストを実行しているサイトの診断マネジャにACKを応答する。

8). ブロードカストを実行しているサイトは、返送されてきたACKの数によって、自身が

マジョリティかマイノリティかを判断し、もしマジョリティの場合はステップ9)を，マイノリティの場合にはステップ10)を実行する。

9). マジョリティの場合、全サイトからACKが応答されたか否かをチェックし、応答しなかったサイトを新たにエラーサイト・リストにかえ、再度ステップ6)を実行しなおす。メッセージを送信した全サイトからACK応答があった場合は、プロトコルの実行を終了し、各診断マネジャは、新しいエラーサイト・リストに基づいてマスタ選択リスト等のデータを変更し、系の再構成を行う。

10). 自サイトがマイノリティであると判断した診断マネジャは、ACKを応答したすべての診断マネジャに対し"We are down"メッセージおよびACKを応答してきたサイト名のリスト（マイノリティ・リスト）を送信し、データベース・サイトとしての活動を休止する。

3.4. 復旧処理

各サイトにおける障害の復旧処理はサイト内の障害原因が取り除かれたとき、または自サイトの診断マネジャによりフラグメント・プロセスに対して復旧指示があったとき、あるいはネットワーク・パーティショニングの原因が解除されたとき起動される。以下にその概要を示す。

3.4.1. ヒストリの形式

各サイトでは、障害の復旧処理に備えて、ヒストリと呼ばれる最新のアップデート・トランザクションのリストを、各フラグメント毎に保持している。図3.1は、このヒストリの形式を示したものである。

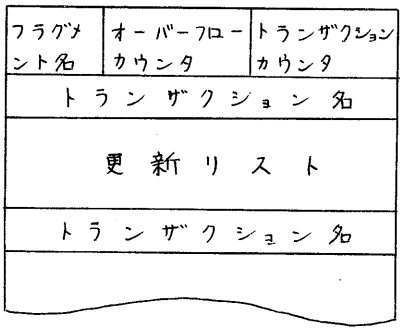

図3.1 ヒストリの形式

フラグメント名は、このヒストリがどのフラグメントに対応するものであるかを示している。またトランザクション・カウンタは、ヒストリ中に登録されているトランザクションの数を表わしている。トランザクション名は、そのトランザクションをユニークに識別するための識別子である。さらに、更新リストは、そのトランザクションの実行時に使用された更新リストである。なお、このヒストリは、登録するトランザクション数がある一定値を越えると古いトランザクションが放棄される。このようなヒストリ中のトランザクションのオーバーフローが何回起ったかを示す値が、オーバーフロー・カウンタである。なおヒストリ中に収録するトランザクションの最大数をどのようにとるかは、アプリケーションに応じて可変とする。

3.4.2. 障害復旧手順

フラグメント・プロセスによる障害復旧手順は、次のようになる。

1). 復旧処理を実行しようとするフラグメント・プロセスは、同一クローズド・アップデート・グループ内の任意のメンバに対し、ヒストリ中のオーバーフロー・カウンタの値を問い合せる。

2). 応答されたオーバーフロー・カウンタの値と、自身のヒストリ中のオーバーフロー・カウンタの値とを比較し、もし一致しなければ、ヒストリを用いての復旧は不可能と判断される（この場合、該当フラグメント全体の転送が必要となる）。

3). オーバーフロー・カウンタの値が一致した場合該当フラグメントのヒストリ全体の転送を要求する。そのあと、受信されたヒストリにより未だ実行されていないトランザクションを検出し、更新リストの内容を書き込む。

一方、自サイトがパーティショニングによりマイノリティとなってしまったが、後になってその原因が取り除かれたと判断されたサイトでは、そのサイトの診断マネジャがマイノリティ・リストの内容に従ってエラーサイト・リストを更新し、系再構成プロトコルのステップ8)以降を実行する。次に、こうしてパーティショニング状態から復旧できたサイトでは、そのサイ

ト内の診断マネジャが、自サイト内のすべてのフラグメント・プロセスに起動をかけ、各々の障害復旧処理を実行するよう指示を出す。

4. 結論

本稿では、分散データベースにおけるデータ・コンシステンシイの維持方式について、特に障害生起時の問題に焦点をあてて考察を行なった。なお、ここでは、クリティカルな処理を実行している2つ以上のプロセスが、同時に障害を起すことはない、という仮定をおいているが、この仮定がどの程度正当なものであるかは、今後信頼性の面から定量的に考察を行って行く予定である。また、障害発生時のプロトコルの詳細については、解決すべきいくつかの問題点（例えば、マイノリティ・ネットワーク内に、あるクローズド・アップデート・グループの一つのフラグメント・プロセスを除く他のすべてのフラグメント・プロセスが含まれてしまった場合のコンシステンシイの維持方式等）が残されているが、今後、プロトコルの詳細化とともに解決してゆく方針である。

さらに、本稿で考察した障害検出/復旧のプロトコルの他にも、数多くの方式が提案されており、通信コスト、信頼度、応答遅延といった面からの定量的な比較も今後の研究課題として残されており、それによりアプリケーションに応じて最も適した方式を判定する基準といったものを考察してゆく予定である。

参考文献

[1] J.B. Rothnie, N. Goodman "A Survey of Research and Development in Distributed Management" Proc, Int Conf, VLDB, 1977, pp.48-62

[2] P.A. Bernstein, D.W. Shipman, J.B. Rothnie, N. Goodman "The Concurrency control mechanism of SDD-1: A System for Distributed Database (General Case.)" Technical Report CCA-77-09, 1977.

[3] M. Stonebraker "Concurrency Control and Consistency of Multiple copies of Data in Distributed INGRES" Proc of the THIRD BERKELEY WORKSHOP, 1978, pp235-258.

[4] C.A. Ellis "A Robust Algorithm for Updating Duplicate Data Bases" proc of the SECOND BERKELEY WORKSHOP, 1977, pp146-158.

[5] H. Hammar, D. Shipman "An overview of Reliability Mechanisms For A Distributed Data Base Systems" proc. COMPCON 1978, Spring.

[6] P. Alsberg, J. Day "A Principle for Resilient Sharing of Distributed Resources" proc. 2nd International Conference on Software engineering, 1976.

[7] Gray.J et al. "Granularity of locks and Degrees of Consistency in a Shared Data Bases" IBM Research, San Jose, Ca., RJ1849, July, 1976.

[8] Yamazaki. H et al. "A Hierarchical Structure for Concurrency Control in a Distributed Database System" proc. 6th Data Communication Symposium 1976 (To be published)

[9] 山崎, 疋田, 吉田, 川上, 松下 "分散データベースにおける同期制御のための階層型プロトコル" 情報処理学会データベース管理, 分散処理合同研究会 昭和54年9月20日