オペレーティング・システム 56－2
(1992. 8. 19)

分散トランザクションシステム IXI における トランザクション管理方式

奥村 康男† 大久保 英嗣† 大野 豊† 白川 洋充††

†立命館大学理工学部情報工学科

††近畿大学理工学部経営工学科

分散トランザクションシステム IXI は，分散トランザクションを構築するためのプラットホームである．即ち，IXI は分散アプリケーションの開発コストを軽減することを目的としている．本論文では，IXI の構成要素の一つであるトランザクション管理部の機能と構成について述べる．IXI のトランザクション管理部では，トランザクション処理の並行性を高めるために入れ子トランザクション (nested transation) の機能を提供している．さらに，トランザクション処理の柔軟な記述を可能とするために，従来の一貫性 (robust consistency) の概念を拡張した弱い一貫性 (weak consistency) の概念を導入している．本論文では，これらの機能を実現するための具体的な処理方式について述べる．

A Transaction Manager in Distributed Transaction Sysytem IXI

Yasuo Okumura† Eiji Okubo† Yutaka Ohno† Hiromitsu Shirakawa††

†Department of Computer Science and Systems Engineering,
Faculty of Science and Engineering, Ritsumeikan University
56-1 Tojiin Kita-machi, Kita-ku, Kyoto 603, Japan

††Department of Industrial Engineering,
Faculty of Science and Engineering, Kinki University
3-4-1 Kowakae, Higashi-Osaka, 577, Japan

The distributed transaction system IXI is a platform to construct distributed transactions. Namely IXI aims at decreasing the development cost for the distributed applications. In this paper, the functions and construction of the transaction manager of IXI are described. In the transaction manager of IXI, the nested transaction is supported in order to enhance the concurrency of transactions. Furthermore the weak consistency is introduced, which is an extension of the robust consistency, in order to improve flexibility of describing transaction processings. In this paper, the processing scheme for the nested transaction and weak consistency are described.

1 はじめに

これまでに開発された分散トランザクションシステムとしては，CMU の Camelot [1] や，MIT の Argus[2] などがある．これらのシステムでは，トランザクション処理の並行性を高めるために，入れ子トランザクション (nested transaction) の概念を導入している．しかし，これらのシステムではトランザクション処理に関する一貫性が強い一貫性 (robust consistency) に限られているため，子トランザクションは親トランザクションに従属した存在となり，柔軟な処理が行えないといった問題がある．また，並行処理制御をロッキング方式で行っているため，アクセス競合が多くなると処理効率が低下するといった問題もある．

我々は，以上の問題点を解決するために分散トランザクションシステム IXI を開発している．IXI は，分散トランザクションシステムを構築するためのプラットフォームであり，以下の特徴を有している．

(1) 並行処理制御の方式や，コミットメント処理の方式を一方式に限定せず，トランザクションの性質に合わせて選択し，処理効率を向上させることができる．

(2) トランザクションに入れ子構造と弱い一貫性 (weak consistency) の概念を導入し，柔軟なトランザクション記述を可能としている．

本論文では，現在，Mach 上で開発中の分散トランザクションシステム IXI における入れ子トランザクションと弱い一貫性の機能を実現しているトランザクション管理部の処理方式について述べる．トランザクション管理部は，トランザクションに関する情報を管理し，クライアントからの処理要求を受け付ける窓口となるモジュールである．その処理内容は，トランザクションの開始から終了までに関係し，システム全体を制御する部分である．

以下，2章で IXI の概要を述べた後，3章で入れ子トランザクションについて，4章で弱い一貫性の概念について述べる．最後に5章で IXI におけるトランザクション管理方式について述べる．

2 IXI の概要

分散環境におけるトランザクションの概念は，共有資源に対するアクセスを制御するための単位として用いられている．分散処理の信頼性を保証するために，トランザクションには，原子性 (Atomicity), 一貫性 (Consistency), 孤立性 (Isolation), 永続性 (Durability) の 4 つの性質が要求される [4]．この 4 つの性質 (ACIDity) を分散環境で保証するシステムが分散トランザクションシステムであり，競合するトランザクションの並行処理制御や，障害が発生した場合の回復処理を行わなければならない．

IXI は，クライアント (トランザクション) と，それを実行するサーバの媒介となるシステムサーバである．トランザクションシステムとして必要な機能を IXI では，以下に示す 3 つのタスクによって実現している（図 1参照）．

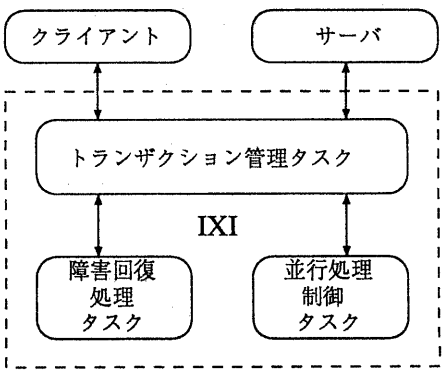


図 1 IXI の構成

(1) 並行処理制御タスク

競合するトランザクション間で，並行処理制御を行い，資源を矛盾のない状態に保つための処理

を行うタスクである．並行処理制御方式としては，従来から2相ロック方式，時刻印方式，楽観的制御方式などが提案されている．しかし，分散環境で性質の異なるトランザクションを扱う場合，どの方式にも一長一短があり，どのようなトランザクションについても柔軟に対処できる万能な並行処理制御方式は存在しない．このため，IXIでは，複数の並行処理制御方式を用意しておき，クライアント設計者がトランザクションの性質に合わせて選択することを可能としている．

(2) 障害回復タスク

トランザクション処理の履歴を管理し，システムに障害が発生した際のトランザクションのredo/undo処理を行うタスクである．IXIでは，回復機能を実現するための手法として，資源の更新操作に先立って，操作の内容を記録しておく書き込み先行ログ方式（write-ahead logging）を用いている．

(3) トランザクション管理タスク

トランザクションを管理し，コミットメント処理を行うタスクである．クライアントが記述したトランザクションは，トランザクション管理タスクに処理要求を出す形で処理を進める．トランザクション管理タスクは，クライアントからの処理要求を受け付け，各システムサーバに指示を出しながら，トランザクションの処理を行う．IXIではコミットメント処理に，2相コミットメント方式を用いている．この方式は，実際にコミットメント処理を行う前に，あるサイトが調停者となって処理に関係した他のサイト（従事者）と同期を取っておくため，システム内でコミットメント処理についての一貫性が維持できる．さらに，IXIでは，耐障害性を重視したい場合には3相コミットメントによる処理も可能となっている．

各タスク間の通信は，メッセージ通信を用いて行われる．システムサーバは，分散環境を構成しているすべてのサイトに一つずつ配置される．各サイトに存在するシステムは，自サイト内のトランザクションと資源とを管理している．IXIでは，自サイトにおける処理も他サイトにおける処理も区別なく，システムサーバにメッセージを送信する形で行われる．このため，クライアントはシステム内部での複雑な処理から解放され，クライアントにとって本質的な処理のみを記述するだけで柔軟なトランザクション処理を行うことができる．

3 IXIにおける入れ子トランザクション

トランザクションに入れ子構造を導入した場合，並行処理制御や親トランザクションと子トランザクションとの間の同期の取り方など特別な処理が必要となる．本章では，IXIにおける入れ子トランザクションの管理方式について述べる．

3.1 入れ子トランザクションの実行

IXIでは，トランザクション内部での処理に並行性を持たせるために，トランザクション内で複数のサブトランザクションを同時に実行することを可能にしている．このようなトランザクションの入れ子構造は，親子の関係で表され，親のトランザクションによって生成されたサブトランザクションは，その子供となる．原則として，子トランザクションの処理内容は親トランザクションに従属しているが，IXIでは，弱い一貫性の機能（4章参照）を用いることにより，子トランザクションを独立したトランザクションとして扱うことも可能である．

IXIでは，複数の子トランザクションの実行は，以下の関数によって行われる．

(1) cobegin()

子トランザクションとそれらの一貫性のモード（4.1節参照）を指定する．

(2) coend()

親トランザクションと子トランザクションとの間の同期を取る役目を果たす．親トランザクションは，coend()ですべての子トランザクションの

終了を待つが，それまでは，子トランザクションと並行して実行される．

3.2 並行処理制御方式

IXI における並行処理制御は，2相ロック方式に基づく場合と，時刻印方式に基づく場合それぞれについて処理が異なる．

(1) 2相ロック方式

2相ロックを用いた場合のサブトランザクション間の並行処理制御には，資源に対するロックの hold と retain のモードを設ける [3]．ある資源に対するロックの hold とは，資源に対するアクセス権を得たことを示す．一方，ロックの retain とは，他の (一族以外の) トランザクションからのアクセスを禁止することを意味する．子トランザクションの処理が終了すると，そのロックは親トランザクションによって retain される．retain されたデータは，親トランザクションの子孫 (descendants) 以外のトランザクションが hold することはできない（図 2参照）．

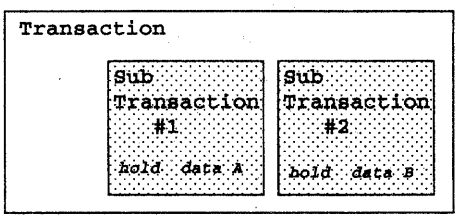
(2) 時刻印方式

IXI では，各トランザクションは処理を開始した時刻を時刻印として持つ．トランザクションが入れ子になっている場合には，子トランザクションは先祖の時刻印も継承する．この場合，以下のアルゴリズムにしたがって並行処理制御に用いられる時刻印が決定される (図 3参照)．

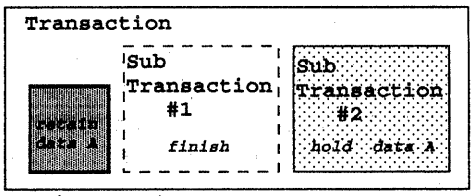
(a) トランザクション T が入れ子レベル k で実行されている場合，T を含んでいるトランザクションの時刻印を $TS_i(T)$ $(1 \leq i \leq k$; 入れ子レベル$)$ とする．(T を含むトップレベルのトランザクションの時刻印は $TS_1(T)$ となり，T 自身の時刻印は $TS_k(T)$ と表される．)

(b) 2つのトランザクション T_1(入れ子レベル l) と T_2(入れ子レベル m) のトランザクションが競合している場合，$i = 1 \rightarrow min(l,m)$ について，$TS_i(T_1)$ と $TS_i(T_2)$ とを順次比較し，最初に $TS_i(T_1) \neq TS_i(T_2)$ となった時点の時刻印を並行処理制御に用いる．

従って，子トランザクションに与えられた時刻印は，共通の親トランザクションを持つ兄弟トランザクション間の並行処理制御に用いられる．



(a) 子トランザクションによるロックの hold



(b) 親トランザクションによるロックの retain

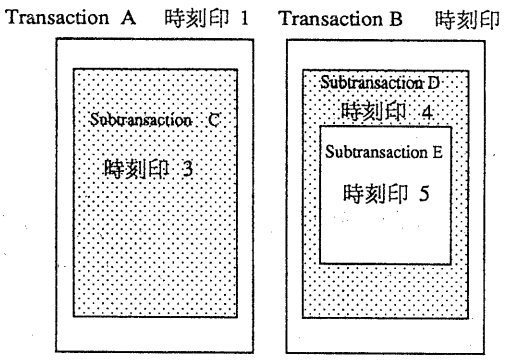
サブトランザクション#1が終了した場合、#1によってアクセスされていたデータ A は、親トランザクションによって retain され、一族以外のトランザクションからはデータ Aにアクセスできない。
資源の解放は、親トランザクションの終了時に行われる。

図 2 ロックの継承

4 IXI における弱い一貫性

IXI では，入れ子トランザクションの処理が柔軟に行えるように，従来から用いられている一貫性に加えて，弱い一貫性の概念を導入している．

従来のシステムでは原子性によって，各トランザクションの処理が完了しなければ当該トランザクションのコミットは行われなかった．このことは，子トランザクションを並行処理させている場合も同様であった．しかし，現実にはこのような強い一貫性を必要としない処理も存在する．IXI では一貫性の概念を拡張し，子トランザクション



サブトランザクションCとサブトランザクションDとが、競合している場合は、時刻印1と2が比較される。
サブトランザクションDとEとが競合している場合は、各々のサブトランザクションに与えられた時刻印が用いられる。

図3 時刻印の継承

として定義された処理のコミットおよび破棄に関する制限を緩和している．

4.1 弱い一貫性の種別

IXIでは，子トランザクションの実行において，以下に示す四種類の一貫性に関する種別を設けることによって，弱い一貫性を実現している．

(1) コミット先行モード

親に先行してコミットすることを認めるモードである．ただし，親が破棄する場合には，先行してコミットした内容は，破棄されなければならない．

(2) 破棄可能モード

子トランザクションが破棄した場合でも，親トランザクションはその影響を受けない．このモードにおいて子トランザクションがコミットする場合，コミットメント処理は，親トランザクションがコミットするまで延期される．

(3) 独立モード

子トランザクションにおける処理結果は，親トランザクションから完全に独立している．親トランザクションと子トランザクションは完全に切り離された扱いをする．

IXIでは，これら三種類の弱い一貫性のモードに加えて，従来の強い一貫性のモードも用意している．

(4) 通常モード

子トランザクションのコミットメント処理は，親トランザクションがコミットメント処理に移行した時に初めて行われる．もし，子トランザクションと親トランザクションのうち，いずれかのトランザクションが破棄された場合，すべてのトランザクションが破棄される．

4.2 補償トランザクション

上記のコミット先行モードに関しては注意が必要である．コミット先行モードで実行される子トランザクションAがその処理を終了し先行コミットした後に，親トランザクションが破棄したとする．この場合，Aによって更新された内容を実行前の状態に戻す措置が必要になる．undo処理により，Aによる更新は破棄されたとしても，既にAによって更新された資源に他のトランザクションがアクセスしていることが考えられる．この場合，このトランザクションは，汚染された資源にアクセスしていることになり，トランザクションAの破棄が連鎖的に他のトランザクションの破棄を引き起こすことになる．

しかし，Aが行った処理を無効にしても，資源に影響を及ぼさない場合は，何ら問題は生じない．しかしこのためには，Aで行った操作をすべて無効にする補償トランザクションが定義できなければならない．例えば，以下のような操作は補償可能であると考えられる．

(1) データの参照のみで更新を行わないトランザクション．

(2) カウンタに対する操作のように，実行順序ではなく実行回数が最終的に意味を持つ操作を行うトランザクション．

5 実現方式

IXI のトランザクション管理部は Mach 上のタスクとして実現されており，以下に示す 3 つの処理から構成されている．

(1) トランザクション管理
トランザクション毎に識別子を割り当て，トランザクションを管理する．

(2) テーブル管理
トランザクションの管理に必要な情報を保持しているテーブルを管理する．

(3) コミットメント処理
他のサイトと協調して，コミットメント処理を行う．

複数のトランザクションを並行実行するために，IXI ではトランザクション開始要求を受け取るたびに，各トランザクション専用のスレッドを生成し，それ以降の処理は各スレッド内で行っている (図4参照)．トランザクションの管理に必要となる各テーブルは，スレッド間の共有資源としてトランザクション管理タスク内に存在する．以下，トランザクション管理に必要となるデータ・テーブルの内容と，トランザクション管理の方法について述べる．

5.1 データ・テーブル

トランザクション管理タスクは，トランザクションの処理に必要な情報を以下に示すテーブルとして保持する．

(1) トランザクション・テーブル
自サイト内に存在するトランザクションに関する情報を管理するテーブルである．テーブルの各エントリには，トランザクション名，トランザクション識別子，実行開始時刻印，並行処理制御モードなどの情報とともに，子トランザクションに関する情報，及び親トランザクションに関する情報が登録される．各トランザクションは，親トランザクションと (直接の) 子トランザクションに関する情報のみを保持し，孫 (およびそれ以降の子孫) トランザクションについての情報は保持しない．

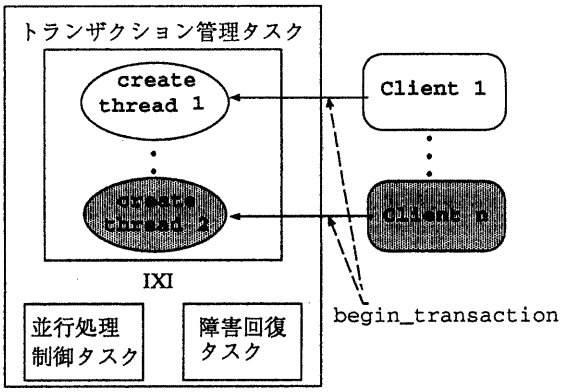


図 4 トランザクション開始時における処理

(2) サーバ・テーブル
システム内にどのような実行サーバが存在するのかを管理するテーブルである．

(3) ファイル・テーブル
IXI では，子トランザクションは，あらかじめ実行形式のファイルとして存在している．そのため，実行ファイル名がそのままトランザクションの名前となる．ファイル・テーブルとは，システム全体にどのようなトランザクションが存在するのかを管理するテーブルである．

5.2 トランザクション開始時における処理

IXI において，トランザクションの記述は，C 言語で記述されたインタフェース・ライブラリを用いて行われる．IXI で提供しているトランザクション記述インタフェースを表 1に，トランザクションの記述例を図 5に示す．

トランザクション開始時における処理は以下のように行われる．

(1) クライアントからのトランザクション開始要

求(begin_transaction())を受け付けるたびにトランザクション管理タスクは，スレッドを生成する．そのトランザクションの処理は，生成したスレッドにおいて行われる．

(2) (以下各スレッド内で処理が行われる)
トランザクション・テーブルを操作し，トランザクション識別子などの情報を登録する．

(3) その後のデータの参照・更新要求も，一旦，トランザクション管理タスクを経由して，各実行サーバに処理要求が出される．

子トランザクションを実行する場合，親トランザクション内で cobegin() を呼び出し，並行実行するトランザクションを明示しなければならない．関数 cobegin() は，指定された子トランザクションの数だけタスクを生成し，各子トランザクションの処理は，そのタスク内で実行される．IXI では，記述形式上，親と子のトランザクションの区別はなく，子トランザクションに対する開始操作も親トランザクションと同じように行われる．親と子のトランザクションの区別は，トランザクション実行時の引数によって区別される．トランザクション A が子である場合，まず A にトランザクション識別子を割り当てる．さらに，A の親トランザクションのテーブルにも，A の情報を登録する．この時に，子トランザクション A の一貫性のモードも登録される．

表 1 トランザクション記述インタフェース

begin_transaction()	トランザクションの開始
end_transaction()	トランザクションの終了
cobegin()	子トランザクションの並行実行
coend()	子トランザクションの終了待ち
call_server()	実行サーバに対する処理要求

5.3 トランザクション終了時における処理

end_transaction() が呼ばれるとトランザクション管理タスクは，コミットメント処理に移行する．

IXI では，2 相コミットメントを用いている．トップトランザクションの存在するサイトが調整者となって，コミットメント処理は進められるが，子トランザクションを並行実行させている場合には，従事者サイト以外に，子トランザクションとも同期を取らなければならない．トランザクション終了時における処理は以下のように行われる．

```
main()
{
    int ret;
    begin_transaction(...);
    call_server(...);
    cobegin(subtrans_A, mode,
                subtrans_B, mode, ...);
    coend();
    ret = end_transaction();
}
```

図 5 トランザクション記述例

(1) クライアントからトランザクション終了要求(end_transaction())を受け付ける．

(2) 従事者サイトにコミットの可否を問い合わせる．<第 1 相>

(3) 従事者サイトからの返答により，コミットの可否を判断する．

(4) 子トランザクションである場合，コミットの可否を親トランザクションに通知する．先行コミットモードにおけるコミット，破棄可能モードにおける破棄，独立モードにおけるコミット/破棄は，親トランザクションと関係なくコミットメント処理の第 2 相に移行する．これらに該当しない子トランザクションは，親トランザクションからの指示を待つ．

(5) 親トランザクションである場合，従事者サイトからの返答結果と子トランザクションの状態より，コミットの可否を決定する．

(6) 親トランザクションは，従事者サイト並びにコミット待ち状態にある子トランザクションに，コミットの可否を通知する．<第 2 相>

(7) 親トランザクションは，従事者サイト及び子トランザクションのコミットを確認した後，自らのコミット処理に移行する．

5.4 テーブル管理

トランザクション管理タスクが保持する3種類のテーブルの中で，サーバ・テーブルとファイル・テーブルは，システム内の全サーバが，同じ内容のものを保持しなければならない．IXIでは，システム内で一つのプライマリ・サイトを用意しておき，そのサイトが調整者となってテーブルの内容の一貫性を維持する．以下，IXIにおけるテーブル管理のアルゴリズムを述べる．

(1) トランザクション処理を行う実行サーバおよびトランザクション実行ファイルが新しく生成または削除される場合，最初に自サイト内のテーブルの内容が更新される．

(2) 更新のあったサイトのトランザクション管理タスクは，プライマリ・サイトに更新内容を通知し，プライマリ・サイトのテーブルが更新される．

(3) プライマリ・サイトは，システムを構成する全てのサイトに更新内容をブロードキャストする．

(4) 各サイトは，自サイトにあるテーブルの内容を更新する．

6 おわりに

本論文では，分散トランザクションシステムIXIにおけるトランザクション管理方式について述べた．IXIでは，入れ子トランザクションをサポートすることで，トランザクション処理の並行性を高めている．さらに，弱い一貫性の導入により，通常のトランザクションに加え，コミット先行，破棄可能，独立といったモードを持ったトランザクションの構成が可能となっている．これにより，コミット処理のオーバヘッドの削減が可能となっている．

今後は，すでに設計を終えている並列処理制御部との連携を検討し，各部を実現することによって評価を行っていきたいと考えている．

参考文献

[1] Eppinger, J. L., Mummert, L. B. and Spector, A. Z. : *CAMELOT AND AVALON A Distributed Transaction Facility*, Morgan Kaufmann Publishers (1991).

[2] Liskov, B., Curtis, D., Jhonson, P., and Scheifler, R. : Implemantation of Argus, In *Proccedings of 11th ACM Symposium on Operating Systems Principles*, pp. 111–122 (1987).

[3] Moss, J. E. B. : Nested Tansactions : An Approach to Realiable Distributed Commputing. Technical report, Massachusetts Institute of Technology, MIT/LCS/TR-260 (1981).

[4] Özsu, M. T. and Valdiriez, P. : *Principles of Distributed Database Systems*, Prentice-Hall International Editions (1991).