システムソフトウェアと
オペレーティング・システム 65−12
（1994．7．21）

分散トランザクションシステムIXIの設計と実現

松高 雄一[†] 中村 素典[†] 大久保 英嗣[†] 大野 豊[†] 白川 洋充[††]

[†]立命館大学理工学部情報学科

[††]近畿大学理工学部経営工学科

我々は，分散環境上でのアプリケーション開発を支援するためのプラットホームとして，分散トランザクションシステムIXIを開発している．IXIは，分散環境上でのさまざまな透過性を保証している．また，入れ子トランザクションの処理機能や複数の並行処理制御方式を選択できる適応型時刻印方式を提供している．本論文では，IXIを構成する各機能モジュールの処理方式と，オペレーティングシステムMach上に構築したプロトタイプシステムについて述べる．

A Design and Implementation of Distributed Transaction System IXI

Yuichi Matsutaka[†] Motonori Nakamura[†] Eiji Okubo[†] Yutaka Ohno[†]
Hiromitsu Shirakawa[††]

[†]Department of Computer Science,
Faculty of Science and Engineering, Ritsumeikan University
1916 Noji, Kusatsu, Shiga 525, Japan

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

We have been developing the distributed transaction system IXI. IXI is a platform to support the application development on a distributed environment. By means of functions of IXI, it becomes easy to attain various transparency. IXI provides with functions for the nested transactions and the adaptive time-stamp ordering which is used as a strategy of concurrency control. In this paper, the outline and features of components which consist of the prototype system of IXI on Mach operating system are presented.

1 はじめに

近年，LAN(Local Area Network) の普及により，複数の計算機をネットワークで結合した分散システムが数多く構築されており，またこれらのシステム上でデータベース，CAD(Computer Aided Design)，グループウェアなどのアプリケーション機能を実現するための分散システムの研究がなされている．

分散システムにおいては，処理の複雑さをユーザから隠蔽するトランザクションの概念が重要となる．しかし，複数のサイトにあるトランザクションの管理を行うことは容易ではなく，データベースや CAD などのアプリケーション毎にプログラマがトランザクション処理を実現することは効率的ではない．

現在，我々はこのような分散トランザクションを用いたアプリケーションプログラムの開発を支援するために，オペレーティングシステム Mach 上に分散トランザクションシステム IXI を開発中である [4]．

従来の分散トランザクションシステムでは，並行処理制御方式として 2 相ロックに基づく方式を採用しているが，IXI では適応型時刻印方式を用いることにより，トランザクションの並行性を向上させている．また，入れ子トランザクションに弱い一貫性の概念を導入することにより，アプリケーションの柔軟な記述を可能としている．

本論文では，分散トランザクションシステム IXI の設計と実現およびその評価について述べる．

2 トランザクション・モデル

2.1 トランザクションの定義

トランザクションとは，BEGIN と END で囲まれた一連のデータ操作であり，以下の 4 つの性質を保証しなければならない．

(1) **原子性** (Atomicity)

トランザクションはそれ以上分割できない単位であり，すべての処理は完全に実行されるか，または全く実行されないかのどちらかでなければならない．トランザクションが，すべての処理を完了することをコミット (commit) と言い，途中で処理が失敗することをアボート (abort) と言う．

(2) **一貫性** (Consistency)

トランザクションの実行は，資源をある一貫した状態から別の一貫した状態に遷移させなければならない．

(3) **孤立性** (Isolation)

トランザクションはそれぞれ独立しており，処理途中の状態は他のトランザクションから隠蔽される．

(4) **永続性** (Durability)

一旦コミットされた結果は速やかに恒久化され，それを破棄することはできない．

これらの性質を保証するためには以下に述べる処理が必要となる．

(1) 並行処理制御 (concurrency control)

分散システムでは，性能を向上させるために複数のトランザクションが並行に実行される．この時オブジェクトの一貫性を保つためには，トランザクションのスケジュールが直列可能 (serializable) でなければならない．これを保証するために，何らかの規則によってオブジェクトへのアクセスを制限することを並行処理制御という．

(2) 信頼性制御 (reliability control)

トランザクションがアボートした場合または何らかの障害が発生した場合には，資源を一貫性のある状態に移行させる処理が必要になる．これはトランザクションの処理内容をログとして二次記憶上に保存しておき，障害発生時にトランザクションの行った操作の無効化 (undo) または再実行 (redo) 処理を行うロギング方式によって実現される．また，分散環境上で実行される場合には，トランザクションの原子性の保証のために，処理に関連したサイト間で同期を取りながら，トランザクションの終了処理を行う必要がある．これをコミットメント処理と言う．

さらに，これらに加えて分散環境上でのトランザクション開発を低コストで可能にするためには，ユーザに対してネットワーク透過性 (network transparency) を保証する必要がある．主なネットワーク透過性には以下のものがある [5]．

(1) 位置透過性 (location transparency)

資源の分散環境上での存在位置を意識することなく，ローカルサイトでの処理と同様に処理を行うことができる．

(2) 並行実行透過性 (concurrency transparency)

共有資源に対してアクセスを行う場合，並行に実行される複数の処理の干渉を意識する必要がない．

(3) 障害透過性 (failure transparency)

ハードウェア，およびソフトウェアの障害が意識されない．　分散環境上の一部で障害が発生しても，可能な限り機能を維持しなければならない．

このように分散環境上でトランザクション処理を行う場合には，トランザクションのACID特性を保証した上で，分散環境で処理をしていることをユーザに意識させないことが重要である．

2.2 IXIにおけるトランザクションモデル

IXIが対象とするトランザクションモデルは，クライアント・サーバモデルである．クライアント (トランザクション) はメッセージ通信によってサーバに要求を出し，サーバは要求にしたがって共有オブジェクトを操作する (図1参照)．

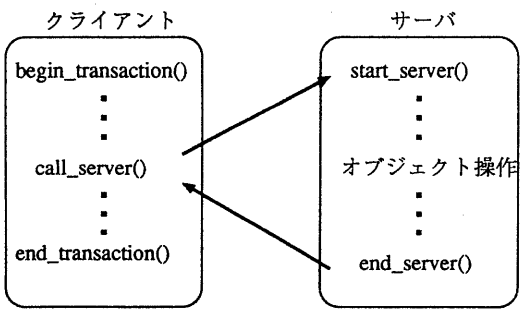

図1　IXIにおけるクライアント・サーバモデル

IXIは，クライアントが記述したトランザクションのACID特性を保証しつつ，分散環境上での複雑な処理をクライアントおよびサーバ実装者から隠蔽する．すなわち，IXIを用いることによって，トランザクションおよびサーバ実装者は，分散環境を意識することなく効率的に分散環境上でプログラム開発を行うことが可能となる．

3 IXIの処理方式

我々は，前章で示したモデルに基づき，分散トランザクションシステムIXIを開発している．本章では，IXIにおける並行処理制御方式，入れ子トランザクションと弱い一貫性の概念，信頼性制御について述べる．

3.1 並行処理制御方式

トランザクションのACID特性を保証するためには並行処理制御が必要となる．従来のトランザクションシステムでは並行処理制御方式として，アクセスする資源にロックを施し，競合するトランザクションのアクセスを制限するロック方式と，トランザクションの処理開始時にトランザクションの発行順序を表わす時刻印を与え，その時刻印に従い処理を行う時刻印方式が用いられてきた．しかし，様々な性質のトランザクションに対し，単一の方式で効率的な制御を行うのは困難である．そこで，IXIでは時刻印方式に基づいた複数の並行処理制御方式を用意している．これらの方式はトランザクション記述者がトランザクションの性質に合わせて任意に選択することが可能である．これを適応型時刻印方式といい，以下の3つの制御方式により構成されている [1][2].

・ 多重版時刻印方式

従来の時刻印方式において，アクセスするオブジェクトを多重版化することにより，トランザクションがアボートする確率を小さくした方式である．多重版時刻印方式では，時刻印に従い参照可能なバージョンを選択することにより，参照操作の失敗を少なくすることができる．この方式では，参照の対象となるオブジェクトの最新バージョンが確定していない場合に，確定するまで待つ保守的方式と，確定する前に先読みすることを認める積極的方式があり，IXIではその両方をサポートしている．

・ 予約型時刻印方式

多重版時刻印方式において，更新操作はより大きな時刻印を持つ後発のトランザクションにより当該オブジェクトが既に参照されていた場合破棄される．このような遅い更新に対処するため我々は予約型時刻印方式を提案している．この方式は，優先して更新したいオブジェクトに予約を行うことにより，後発のトランザクションによる参照要求をブロックし，トランザクションの終了確率を高めようとする方式である．また，予約によって

待ち状態に移行するトランザクションは，より大きな時刻印を持つトランザクションのみであり，デッドロックのような循環した待ち状態が発生することはない．

・排他ロック式時刻印方式

実行時間が長いトランザクションがアボートした場合，再実行のコストが大きくシステム全体の処理効率も低下する．このため，長時間トランザクションのコミット確率を上げる必要がある．また，長時間トランザクションが予約型時刻印方式で実行された場合，予約されたオブジェクトの参照を要求するトランザクションを長時間待たせてしまい，処理効率が悪くなる可能性がある．この問題を解決するために，我々は排他ロック式時刻印方式を提案している．この方式は，時刻印方式と2相ロック方式を組み合わせた方式である．長時間トランザクションは実行に先立って更新するオブジェクトに排他ロックを施す．このロックは，同じ方式で実行されているトランザクション間において，オブジェクトアクセスについての排他処理を実現し，その他の方式で実行されているトランザクションの更新要求を破棄する．

3.2 入れ子トランザクション

トランザクションは論理的な単位であり，その大きさの捉え方はユーザに委ねられている．IXIでは，1つのトランザクション内でさらに別のトランザクションの起動を可能とする入れ子トランザクションの機能を提供している．ユーザは，入れ子トランザクションを使用することによって，柔軟なトランザクションの記述が可能となる．しかも，トランザクションにモジュール性を持たせることにより再利用性を高めることができる．また，明示的にサブトランザクションを並行実行させることにより，トランザクションの処理効率を高めることが可能となる．

入れ子トランザクションを起動した場合，新たに起動されたトランザクションを子トランザクション，起動を行ったトランザクションを親トランザクションと呼ぶ．従来のシステムでは，親と子の依存関係は密であり，子のアボートは親のアボートを引き起こし，また，親のアボートは子のアボートを引き起こすことになる．しかし，実際の処理では，このような強い一貫性 (robust consistency) が必要でなく，むしろ親と子との関係に自由度を持たせた方が処理効率がよくなる場合がある．IXIでは，入れ子トランザクションに関して，強い一貫性の概念を緩和した弱い一貫性の概念を導入している．

3.3 信頼性制御

トランザクション処理中にシステムに障害が発生した場合には，トランザクションが実行していた操作を無効化し，資源を一貫性のある状態に回復させる処理が必要になる．このため，トランザクションが行った処理内容を二次記憶にログとして記録しておく方法が用いられている．IXIでは，書き込み先行ログ方式 (write ahead logging) を用いている．これは，あるオブジェクトに対して行うトランザクションの操作内容を，実際に操作を行う前に二次記憶に記録しておく方法である [3]．

開始されたトランザクションの処理は，最終的にはコミットまたはアボート状態に移行する．分散トランザクションシステムでは，トランザクションの一貫性および原子性を分散処理環境で保証するために，トランザクションが最終的に移行する状態に関して，当該トランザクションの実行に関連したサイト間で同期をとらなければならない．これをコミットメント処理という．コミットメントプロトコルには，2相コミットメントと3相コミットメントがよく知られている．IXIでは両方式を選択的に使用することができる．

4 システム構成

分散トランザクションシステムIXIの実装対象となるハードウェアおよびソフトウェア構成と，分散トランザクションシステムIXIのモジュール構成について述べる．

4.1 ハードウェアおよびソフトウェア構成

我々が対象としているシステムは，複数の計算機がLAN(Local Area Network) によって結合された分散処理環境である．各サイトは一つ以上のプロセッサ，ローカルメモリ，ローカルディスク，通信装置および入出力装置より構成される．した

がって，サイト間には，物理的なメモリおよびクロックの共有はなく，サイト間の通信はメッセージ通信によってのみ可能となる．IXIの機能を実現可能とするために，オペレーティングシステム(以下OS)には，メモリ管理，プロセス(タスク)管理，ネットワーク通信機能が必要となる．この条件を満たすOSとして，我々はMachオペレーティングシステムを開発対象のOSとしている．

4.2 モジュール構成

IXIは，以下に述べるシステムモジュール群によって構成されている．

(1) トランザクション管理部(Transaction Manager)

(2) 並行処理制御部(Concurrency Controller)

(3) オブジェクト管理部(Object Manager)

(4) 回復処理部(Recovery Manager)

(5) ユーザインタフェース部(User Interface)

ユーザインタフェースを除いた4つののモジュールは，互いに独立したタスクとして実装されている．これらのタスクはサイト毎に配置され，それぞれローカルサイトのトランザクション，サーバ，オブジェクトを管理する．図2に，システムタスクの構成を示す．各システムタスクの役割を以下に示す．

・トランザクション管理部

トランザクション管理部は，IXIのシステムタスクの窓口となり，トランザクションおよびサーバを管理するタスクである．トランザクションおよびサーバからの要求は，このモジュールにおいて解析され，並行処理制御部およびオブジェクト管理部に渡される．トランザクションのコミットメント処理においてもこのモジュールが調整の役割を果たす．トランザクション管理部では以下の処理が行われる．

(1) トランザクションおよびサーバからの処理要求の解析．

(2) トランザクション識別子の割り当てと時刻印の付加．

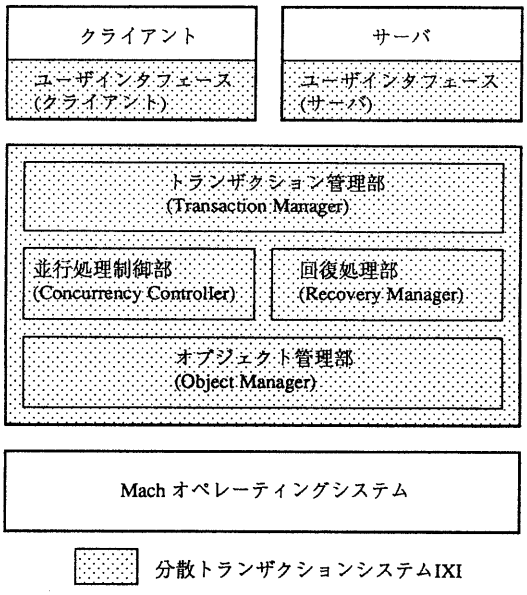

図2 システムタスク

(3) サーバの位置管理および，サーバが提供する関数の引数などの情報を管理する．

(3) コミットメント処理

(4) トランザクション毎に，トランザクション管理情報，状態，行った操作をログに記録する．

・並行処理制御部

並行処理制御部は，適応型時刻印方式にしたがい，ローカルサイトに存在する共有オブジェクトに対する並行処理制御を行う．適応型時刻印方式は多重版時刻印方式に基づいているため，ローカルサイトにあるオブジェクトのバージョンを管理するためのメタオブジェクトを管理している．メタオブジェクトには各バージョンの作成時刻印，参照時刻印，そのバージョンを参照しているトランザクションの情報など，適応型時刻印方式のために必要な情報が含まれる．

・オブジェクト管理部

並行処理の対象となるオブジェクトの新規登録，削除などを行う．また，サーバからオブジェクトのアクセス要求を依頼された場合，自サイトにオブジェクトがなければ，他サイトのオブジェクト管理部と通信を行い，オブジェクトの位置を検索する．

・回復処理部

回復処理部は，システムに障害が発生したとき，各システムタスクの起動に先立ち，トランザクションの回復処理を行う．具体的には，障害が発生したサイトのログを走査し，トランザクションが行った処理を無効化し，各資源を一貫性の保たれた状態に移行させ，トランザクション処理が可能な状態に回復させる．

・ユーザインタフェース部

IXIでは，ユーザが起動するトランザクション，サーバを含めてタスク間でメッセージ通信を行いながら処理が実行される．トランザクションおよびサーバの記述は，C 言語のライブラリとして提供するユーザインタフェースを利用することによって行われる．ユーザはこのライブラリを用いることにより，より少ない負担でトランザクションおよびサーバを記述することが可能となる．IXIのクライアントおよびサーバ記述関数を表1および表2に示す．

表1　トランザクション記述インタフェース関数

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
call_server()	サーバに対する処理要求
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

表2　サーバ記述インタフェース関数

begin_server()	サーバの開始
end_server()	サーバの終了
wait_message()	クライアントからの要求待ち
refer()	共有オブジェクトの参照要求
modify()	共有オブジェクトの更新要求
new()	共有オブジェクトの登録要求
delete()	共有オブジェクトの削除要求

5　評価

我々が開発している分散トランザクションシステムIXIを用いて，トランザクション処理の評価を行った．実験は2台のオムロン製LUNA-88Kをイーサネットで接続したシステムで行った．2台のマシンは，それぞれ1つのCPUとローカルディスクを持っている．なお，使用したオペレーティングシステムはMach2.5である．実験は25回実行し，1回あたりの所要時間を求めている．

空トランザクション

トランザクション起動時のオーバヘッドを計測するために，サーバ呼び出しなどを全く行わない以下のようなトランザクションを実行した．

```
main()
{
        begin_transaction();
        end_transaction();
}
```

トランザクション発行・終了時には以下に示す処理が行われる．

(1) トランザクション管理部への処理開始要求
(2) トランザクション情報の登録
(3) 時刻印の割り当て
(4) コミットメント制御

空のトランザクションを実行し測定した結果，1回あたりの実行時間は170ミリ秒となった．

オブジェクト参照

トランザクションがリモートサイトに存在するサーバに処理を依頼するために必要な時間を計測するために，次のような実験を行った．なお，並行処理制御は保守的多重版時刻印方式で実行した．実験に使用したシステムタスク，アプリケーションサーバおよび参照するオブジェクトの構成を図3に示す．

(1) ローカルサーバ呼び出し (siteA でトランザクションを実行する)
(2) リモートサーバ呼び出し (siteB でトランザクションを実行する)

上記の (1), (2) においてサーバを複数回呼出すトランザクションを実行し，計測を行った．実験

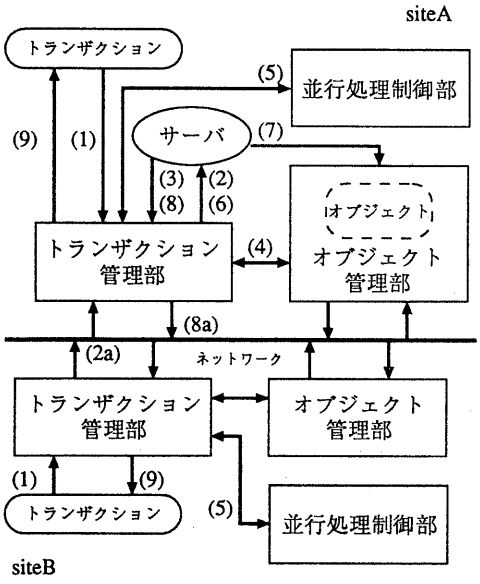

図 3　タスク構成

に用いたトランザクションプログラムを以下に示す．なお，オブジェクト送信に関して，オブジェクトサイズの影響をうけないように，サーバは 0 バイトのオブジェクトを参照する．実験に用いたトランザクションプログラムを以下に示す．ここで，"refer_server" は参照のみを行うサーバである．

```
main()
{
        begin_transaction();
        call_server("refer_server");
                    :
        call_server("refer_server");
        end_transaction();
}
```

ローカルサイトに存在するサーバを呼出す場合，すなわち図 3の siteA でトランザクションを実行させる場合，システムでは以下の処理が行われる．なお番号は図 3の矢印の番号に対応する．

(1) siteA のトランザクション管理部にサーバ呼び出しを要求する．

(2) 要求されたサーバが siteA に登録されているかを調べる．もし，登録されていれば実行する．

(3) 参照要求をトランザクション管理部に要求する．

(4) オブジェクト管理部にオブジェクトの位置を問い合わせる．

(5) 並行処理制御部に問い合わせ，アクセス許可を得る．

(6) (4) と (5) で得た情報をサーバに送る．

(7) (6) で得た情報よりオブジェクトを参照する．

(8),(9) 処理結果をトランザクション管理部を介してトランザクションに送信する．

また，リモートサーバでトランザクションを実行する場合には，これに加えて以下の処理が必要となる．

(2a) ローカルサイトにサーバが存在しない場合，リモートサイトのトランザクション管理部に要求を転送する．

(8a) リモートのトランザクション管理部から，ローカルのトランザクション管理部に処理結果を送信する．

リモートサイトに存在するサーバを実行する場合，ネットワークを介してサーバの位置検索を行うための処理が必要となる．また，コミットメント処理においては，そのトランザクションの実行に関わったサイトすべてがコミットメント制御の対象となるため，このための処理も必要となる．

実験結果

実行結果を表 3に示す．また，表 3にはそれぞれの実行時間の差も示している．図 4は表 3をグラフとして表わしたものである．表 3および図 4から，サーバ呼び出し回数に比例して実行時間の差が増加しているのがわかる．1 回の参照操作に必要な通信回数は，ローカルサーバで実行する場合は 19 回，リモートサーバで実行する場合は 33 回である．従って，リモートサーバで実行する場合は，さらに 14 回の通信が必要となることが分かる．これには，ネットワークを介した通信も含まれる．

表 3より，リモートサーバ呼び出しでは，サーバ呼び出し 1 回につき，ローカルサーバ呼び出しに較べ，平均 170 ミリ秒のオーバヘッドが生じていることが分かる．これが 1 回のリモートサーバ呼び出しにかかる通信オーバヘッドであると考え

られる.

表 3　サーバ呼び出し回数と実行時間 (単位:ミリ秒)

回数	ローカル (1)	リモート (2)	(2)−(1)
1	536	774	238
2	899	1286	387
3	1264	1794	530
4	1534	2305	771
5	2007	2929	922

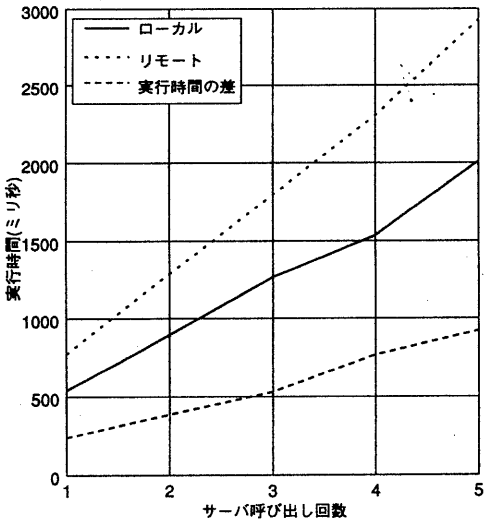

図 4　サーバ呼び出し回数と実行時間

6　おわりに

本論文では，我々が開発している分散トランザクションシステム IXI のプロトタイプシステムの設計と実現について述べた．さらに，本システムを用いて簡単な評価を行った．現在は，回復処理部以外の各機能モジュールがほぼ完成し，各種のテストを行っている段階である．本論文では，プロトタイプシステムを用いて，オブジェクトの参照操作に関して簡単な評価を行ったが，今後は IXI の特徴である予約型時刻印方式を使用して予約を行った場合のオーバヘッドの測定などさらに詳細な評価を行っていきたいと考えている．また，本システムを用いた各種アプリケーションの開発および評価も合わせて行う予定である．

7　謝辞

本研究を行うにあたり，御協力を頂きました大野・大久保・中村研究室の院生の方々に感謝いたします．特に，性能評価に関して御協力頂きました，M1 の小穴泰裕氏に深く感謝いたします．

参考文献

[1] 國枝和雄，畑田孝幸，大久保英嗣，津田孝夫：適応型時刻印方式に基づく同時実行制御，情報処理学会論文誌，Vol. 33，No. 6，pp. 802-811 (1992).

[2] 松高雄一，大久保英嗣，大野豊，白川洋充：分散トランザクションシステム IXI における並行処理制御部の構成，第 37 回システム制御情報学会講演論文集，pp. 111-112 (1993).

[3] 小穴泰裕：分散トランザクションシステム IXI における回復処理部の実現，立命館大学理工学部情報工学科卒業論文 (1993).

[4] 奥村康男：分散トランザクションシステム IXI の構築，立命館大学大学院理工学研究科修士論文 (1993).

[5] 前川守，所真理雄，清水謙多郎編：分散オペレーティングシステム UNIX の次にくるもの，共立出版 (1991).