分散データベースの同期制御の一方式†

池田 哲夫††

本論文は，分散データベースの同期制御方式を提案するものである．従来提案されている方式には，デッドロック検出用の通信量が多い，トランザクションのアボート発生頻度が大きい等の問題点があった．本検討では，グローバル・トランザクションのデータベース更新能力に，更新は起源サイトに限るという制約を課し，その制約の下で，上記の問題点を解決する方式の検討を行った．検討の結果，2相ロック方式と多版時刻印方式とを修正統合した，新たな同期制御方式を提案した．提案方式は，従来方式と比較して，グローバル・トランザクションのトラフィックが低い環境において，グローバル・トランザクションのアボート発生頻度が0であり，かつ同期制御に要する通信量が少ない優れた同期制御方式であることを示した．更新能力に前記の制約を課したグローバル・トランザクションの典型的な使用形態としては，他サイトのデータベースの情報を収集し，自サイトに格納し，格納データを意志決定支援システム等で使うといった使用形態が考えられる．筆者は，既存のデータベースからボトムアップに構築される分散データベースにおいては，グローバル・トランザクションの更新能力に前記の制約を課し，かつグローバル・トランザクションの高トラフィック性をも制限しても支障ないケースが少なからぬ割合を占め，本研究の成果はそのようなケースに有用であると考える．

1. まえがき

分散データベースの同期制御方式は，分散データベース実現のための最重要な課題の一つであり，従来より，2相ロック方式，時刻印方式，楽観的方式，先読みスケジューラ方式など多くの方式が提案されている[1),2)]．これらのうち，実現されている主な方式は，2相ロック方式と時刻印方式の2つである．2相ロック方式は，トランザクションのアボートの発生頻度が小さいという長所を持つが，デッドロックの検出および解決のために大きな通信量を要するという問題点を持つ．一方，時刻印方式はトランザクションの並列走行性が高く，同期制御の決定をローカルに行えるという長所を持つが，トランザクションのアボートの発生頻度が高いという問題点を持つ．

本研究は，グローバル・トランザクションのデータベース更新能力に，更新は起源サイトに限るという制約を課して，その制約の下での同期制御方式を提案したものである．(以下，前記の制約を課されたグローバル・トランザクションを更新制約付グローバル・トランザクションと呼ぶ.) データベース更新能力に制約を課すことにより，上記の，汎用同期制御方式の持つ問題点を解決する同期制御方式を導くことを狙いとした研究である．

更新制約付グローバル・トランザクションの典型的な使用形態としては，他サイトのデータベースの情報を収集し，自サイトに格納し，格納したデータを意志決定支援システム，診断形エキスパート・システム等で使うといった使用形態が考えられる．筆者は，分散データベース，特に，既存のデータベースからボトムアップに構築される分散データベースにおいては，グローバル・トランザクションを更新制約付きグローバル・トランザクションに制約しても支障ないケースが少なからぬ割合を占め，本研究の成果はそのようなケースに有用であると考える．

検討においては，集中形データベースの同期制御方式の主流である2相ロック方式と，時刻印方式の一種で，READ 操作にアボートを生じさせない方式である多版時刻印方式とに着目して，2相ロック方式と多版時刻印方式とを統合した新たな同期制御方式を提案した．従来提案されている方式と比較した結果，この方式は，グローバル・トランザクションのトラフィックが小さい環境において，グローバル・トランザクションのアボート発生頻度が0であり，同期制御に要する通信量が小さいという長所を持ち，同期制御に伴うオーバヘッドの少ない，ローカルにもグローバルにも優れた同期制御方式であることを示した．

以下，2章では，従来提案されている同期制御方式の概要と問題点とを整理する．次いで，3章では，2相ロック方式と多版時刻印方式とを統合した新たな方式の検討を行う．4章では，従来提案されている同期制御方式との比較を行う．

† A Concurrency Control Technique for Distributed Databases by TETSUO IKEDA (NTT Communications and Information Processing Laboratories).

†† NTT 情報通信処理研究所

2. 従来の研究の経緯

分散データベースの同期制御方式は，一般に，ロックを用いる方式，時刻印を用いる方式に大別される[1),2)].

ロックを用いる方式は，参照/更新(READ/WRITE)要求ごとに対象データにロックをかけ，コミットあるいはアボード時にロックを解放するという2相ロックの原理[3)]でトランザクションの直列化可能性を保証する方式である．ロックを用いる方式では，デッドロックが生じうる．デッドロックへの対処法としては，トランザクション待ちグラフの中の有向サイクルの有無を判定することによってデッドロックを検出し，サイクル中のトランザクションをアボートすることによってデッドロックを解消する方式が一般的である[4),5)]．この方式には，アボートの発生頻度が少ないという大きな長所がある．そのため，集中形データベースの同期制御方式の主流となっている．しかし，分散データベースの同期制御方式としては，トランザクションを待ちグラフの作成に多くの通信量を要するという問題点がある．

時刻印を用いる同期制御方式は，競合するトランザクションを時刻印順に実行させることによって，直列可能性を保証する方式である．時刻印方式は，基本時刻印方式と，多版時刻印方式とに大別される．

基本時刻印方式[6)]は，トランザクションの時刻印 T_t と，データの READ 時刻印 T_r と，WRITE 時刻印 T_w とを用いてトランザクションの実行を制御する．READ時は，$T_t > T_w$ の時は読み込み可，それ以外は，トランザクションのアボートになる．WRITE時は，$T_t > \max(T_r, T_w)$ の時は書き込み可，それ以外は，トランザクションのアボートになる．この方式は，トランザクションの並列走行性が高く，同期制御に要する通信量がロック方式より少ないという長所を持つが，アボートが多発するという問題点を持つ．

多版時刻印方式は[7),8)]，データの WRITE ごとに新たな版を作成する点が異なる．各版ごとに READ 時刻印と，WRITE 時刻印とが付随する．WRITE 時刻印は，その版を生成したトランザクションの時刻印であり，READ 時刻印は，それを読んだトランザクションのうち，最新のトランザクションの時刻印である．トランザクションがデータを READ する時は，$T_t > T_w$ を満たす最新の版を選択する．該版がコミット済みの場合は，直ちに読み込みが行われ，未コミットの場合は，ウェイトする．WRITE 時は，最新版の時刻印と比較して，$T_t > \max(T_r, T_w)$ の時は，書き込みと新たな版の作成が可能となり，それ以外は，トランザクションのアボートとなる．性能特性は，基本時刻印方式とほぼ同様であるが，READ 時にアボートが発生しない分，アボート発生頻度が小さくなり，ウェイトが発生する分，並列走行性が低くなる．この方式の著しい特徴は，READ 時に，アボートが発生しないという点である．

上記の方式は，いずれも，トランザクションがグローバルに参照も更新も行う一般的なケースについて扱ったものである．

グローバル・トランザクションを更新制約付きグローバル・トランザクションに制約した場合の同期制御方式については，研究が行われた例はないと思われる．

3. 方式の検討

3.1 検討の前提条件

ローカルデータベースとしては，現状の集中形データベースの技術動向を考慮して，以下の条件を満たすものを前提とする．本検討で提案する同期制御方式は，ローカル・トランザクションのアボート発生頻度，並列走行性に大きな影響を与えるものであってはならないものとする．

- トランザクションの原子性，信頼性保証には，2相コミットメント・プロトコルを使用
- トランザクションの同期制御には，2相ロック方式を使用
- デッドロックは，トランザクション待ちグラフの解析により検出

グローバル・トランザクションは，データベースの更新に関する以下の制約を満たすものに限定する．(更新制約付グローバル・トランザクションの正確な定義)

- グローバル・トランザクションがデータベースを更新できるのは以下の場合に限る．
  ①該トランザクションがルート・トランザクション（トランザクションの親子関係で，他のいかなるトランザクションの子孫にもならないトランザクション）である．
  ②更新対象データは該ルート・トランザクションの起源サイトに存在する．

3.2 方式の提案

子孫グローバル・トランザクション（あるルート・グローバル・トランザクションの子孫であるグローバル・トランザクション）は，READ オンリートランザクション（更新を含まないトランザクション）である．また，READ オンリートランザクションに関しては，2章で述べたように多版時刻印方式が，アボートを生じさせない同期制御方式であることが知られている．そこで，ルート・グローバル・トランザクション（更新がありうる）とローカル・トランザクションについては2相ロック方式の適用を原則とし，子孫グローバル・トランザクションについては多版時刻印方式の考え方をとりこむことによって，新たな方式を作成，提案することを検討する．

以下，検討の内容を，以下の2つに分けて述べる．

・トランザクション種別ごとの同期制御方式の検討
・グローバルな同期制御方式への統合の検討

3.2.1 トランザクション種別ごとの同期制御方式の検討

トランザクション種別ごとの同期制御方式を，以下の4つに分けて検討する．

・時刻印の管理方法
・ローカル・トランザクションとルート・グローバル・トランザクションの READ 規則
・ローカル・トランザクションとルート・グローバル・トランザクションの WRITE 規則
・子孫グローバル・トランザクションの READ 規則

（1） 時刻印の管理方法

グローバル・トランザクションの競合時のアボート優先度を与える手段として，時刻印を用いる．本論文のこれ以降の検討内容と整合させるため，時刻印の管理は以下の条件を満たすものとする．

①ルート・グローバル・トランザクションの時刻印は生起時に与えられ，該サイトで生起済みのすべてのグローバル・トランザクションの時刻印よりも新しい(値が大きい)．

②親トランザクションと子トランザクションとは同一の時刻印を持つ．

（2） ローカル・トランザクションとルート・グローバル・トランザクションの READ 規則

2相ロック方式に従う．READ 時は，READ 対象データの最新版に READ ロックをかけて，READ を行う．WRITE ロックがかかっている場合は，ウェイトする．READ ロックの解放は，トランザクションのコミットあるいはアボート時に行われる．

（3） ローカル・トランザクションとルート・グローバル・トランザクションの WRITE 規則

2相ロック方式に従う．ただし，新しい版を作成する点が単純な2相ロック方式と異なる．WRITE 時は，WRITE 対象データの新たな版を作成し，WRITE ロックをかける．対象データに既に READ ロックあるいは WRITE ロックがかかっている場合はウェイトする．WRITE ロックの解放は，トランザクションのコミットあるいはアボート時に行われる．版への時刻印の付与は，その版がグローバル・トランザクションによって作成された場合のみ行われ，グローバル・トランザクションの時刻印と同一の値が付与される．

ここで，子孫グローバル・トランザクションの読み込みが矛盾なく行われるために，ルート・グローバル・トランザクションのコミット時には，該サイトで，その時点でコミット済みのすべてのトランザクションのリストを保持しておくこととする[8]．

（4） 子孫グローバル・トランザクションの READ 規則

多版時刻印方式に従う．ただし，読み込みを行う版の選択法がオリジナルの多版時刻印方式と異なる．該トランザクションの時刻印を越えない範囲で最新の時刻印を持つルート・グローバル・トランザクションをTとする．Tがコミット済みの場合は，子孫グローバル・トランザクションは，(3)で作成した，Tのコミット時点でコミット済みだったトランザクションのリストを基に，それらによって作成された版のうち，最新のものを読み込む．(これ以降，あるトランザクション T0 に関して，コミット済みでかつ T0 の時刻印を越えない範囲で最新の時刻印を持つルート・グローバル・トランザクション T1 を，T0 の READ 位置付けトランザクションと呼ぶ．) T が未コミット時の規則については後で(3.2.2(2)で) 検討する．

なお，ルート・グローバル・トランザクションの起源サイトと同一のサイトで生成された子孫グローバル・トランザクションの READ は，ルート・グローバル・トランザクションの READ と同じ規則（2相ロック方式）に従い，それぞれの READ は，同一トランザクションから発行されたものとして制御することとする．以降の検討では，特に断わらない限り，子孫トランザクションで起源サイトがルート・グローバル・トランザクションと同一のものについては，ルー

ト・グローバル・トランザクションと同一視して扱う.

3.2.2 グローバルな同期制御方式への統合

上記の方式を単純に組み合わせて適用するだけでは，正しい（すなわち，直列化可能な）同期制御方式とはならない．正しい制御方式実現のため，以下の点について検討する.

- 同一サイト起源のルート・グローバル・トランザクションの直列化
- 同一サイト内のグローバル・トランザクションの直列化

次いで，ローカル・デッドロック発生時の対処法を検討し，最後に，同期制御方式のまとめを示す.

（1） 同一サイト起源のルート・グローバル・トランザクションの直列化

以下の例に示すように，子孫トランザクションでのREAD の発行順に依存して，異なるサイトにおけるグローバル・トランザクションの直列順序が逆転し，矛盾が生じる可能性がある.

[例]

以下の前提の下での，サイトA，Bにおける実行は矛盾を生じさせる.

T１：サイトA起源のルート・グローバル・トランザクション
T２：同上
T３：サイトB起源のルート・グローバル・トランザクション
T21：サイトBでのT１の子トランザクション
T22：サイトBでのT２の子トランザクション

〈サイトA〉	〈サイトB〉
T１：WRITE X	T22：READ Z
T２：READ X	T３：WRITE Z
（ウェイト）	
T１：COMMIT	T３：COMMIT
T２：READ X	T21：READ Z
（実行）	（T３の作成版を READ）
T２：WRITE X	
T２：COMMIT	

サイト１での直列順序	サイト２での直列順序
T１＜T２	T２＜T１

この問題を解決するためには，各サイトにおいてルート・グローバル・トランザクションを直列に（時刻印順に）走行させるようにすればよい.

このようにすれば，同一のサイトを起源に持つすべてのルート・グローバル・トランザクションと，それらのすべての子孫グローバル・トランザクションは，任意のサイトで時刻印順に直列化される.

（2） 同一サイト内のグローバル・トランザクションの直列化

以下の例に示すように，異なるサイトにおける多版READ の実行順序に依存して，異なるサイトにおけるグローバル・トランザクションの直列順序が逆転し，矛盾が生じる可能性がある.

[例]

以下の前提の下での，サイトA，Bにおける実行は矛盾を生じさせる.

T１：サイトA起源のルート・グローバル・トランザクション
T２：サイトB起源のルート・グローバル・トランザクション
T21：サイトBでのT１の子トランザクション
T12：サイトAでのT２の子トランザクション

〈サイトA〉	〈サイトB〉
T12：READ X	T21：READ Y
T１：WRITE X	T２：WRITE Y
T１：COMMIT	T２：COMMIT

サイト１での直列順序	サイト２での直列順序
T２＜T１	T１＜T２

この問題を解決するためには，子孫グローバル・トランザクションが READ を行う時には，該トランザクションの時刻印より古い時刻印を持つルート・グローバル・トランザクションがすべてコミットあるいはアボートするまでウェイトするようにすればよい.

このようにすれば，同一サイト内のルート・グローバル・トランザクションと子孫トランザクションとは時刻印順に直列化される.

（3） ローカル・デッドロックの解決方法

ローカル・デッドロックの発生時は，グローバル・トランザクションをアボートすることはせず，ローカル・トランザクションをアボートすることによってデッドロックを解決する．(1)より，ローカル・デッドロックに参加するグローバル・トランザクションは高々１つなので，デッドロック・サイクル中には，必ずローカル・トランザクションが存在するので，これは常に可能である.

（4） 同期制御方式のまとめ

- ローカル・トランザクションとルート・グローバル・トランザクションとの同期制御方式は２相ロッ

ク方式である．ただし，WRITE の時は，新しい版を作成し，その版を WRITE ロックする．ルート・グローバル・トランザクションのコミット時は，該サイトで，その時点でコミット済みのすべてのトランザクションのリストを保持しておく．

- 子孫グローバル・トランザクションの READ は多版時刻印方式に従う．ただし，版の選択法がオリジナルな方式と異なる．該トランザクションの，READ 位置付けトランザクションを T とする．多版 READ を行うグローバル・トランザクションは，T のコミット時点でコミット済みだったトランザクションで作成された版のうち，最新のものを読む．なお，ルート・グローバル・トランザクションの起源サイトと同一のサイトで生成された子孫グローバル・トランザクションの READ は，ルート・グローバル・トランザクションの READ と同じ規則に従い，各々の READ は，同一トランザクションから発行されたものとして扱われる．
- 各サイトにおいて，ルート・グローバル・トランザクションは，時刻印順に直列に実行される．子孫グローバル・トランザクションは，該トランザクションの時刻印より小さい時刻印を持つルート・グローバル・トランザクションがすべてコミット，あるいはアボートされるまでウェイトする．
- ローカル・デッドロック発生時は，ローカル・トランザクションをアボートすることによって解決する．
- ロックの両立性規則を**表 1** にまとめる．

3.3 提案方式の主な特徴

特徴 1：すべてのトランザクションは，直列化可能である．（すなわち，本論文で提案した同期制御方式は正しい制御方式である．）

表 1 ロック両立性規則
Table 1 Lock compatibility rules.

後行操作 / 先行ロック	ローカル READ	グローバル READ 注 1)	グローバル READ 左記以外	WRITE
READ	可 ロック	可 ロック	可 ロックはかけない	不可 ウェイト
WRITE	不可 ウェイト	不可 ウェイト	可 ロックはかけない	不可 ウェイト

注 1) ルート・グローバル・トランザクションの起源サイトにおける，ルート・グローバル・トランザクションあるいはルート・グローバル・トランザクションの子孫トランザクションによる READ の場合．

まず，各サイトにおいて，グローバル・トランザクションが時刻印順に直列化されるような，トランザクションの直列化が可能であることを示す．

ルート・グローバル・トランザクションとローカル・トランザクションとに関しては，2 相ロック方式を用いていることと，ルート・グローバル・トランザクション同士が直列に実行されることより，ルート・グローバル・トランザクション同士が時刻印順に並び，かつ，ルート・グローバル・トランザクションのコミット時点でコミット済みだったすべてのローカル・トランザクションを直列順で，該ルート・グローバル・トランザクションに先行させるような直列化が可能である．子孫グローバル・トランザクションについては，それらを時刻印順に並べた後，対応する READ 位置付けトランザクションの直後に位置付けることができる．この位置付けは，グローバル・トランザクションの時刻印順性をくずすことはない．以上より，各サイトにおいて，グローバル・トランザクションが時刻印順に直列化されるような，トランザクションの直列化が可能であることが示された．

ここで，すべてのグローバル・トランザクションを時刻印順に 1 列に並べる．各サイトでのトランザクションの直列化において，グローバル・トランザクションの時刻印順の直列化が可能なことより，各サイトのすべてのローカル・トランザクションを，各サイトでの直列順序に矛盾しないように，その列の中に挿入することができる．これは，すべてのトランザクションがグローバル・トランザクションの時刻印順に沿って直列化可能であることを示す．

特徴 2：グローバルなデッドロックは発生しない．

グローバルなウェイトが発生するのは，新しいグローバル・トランザクションの READ が古いグローバル・トランザクションの終了（コミット/アボート）を待つ場合だけである．古いグローバル・トランザクションが新しいグローバル・トランザクションをウェイトすることはありえないので，デッドロック・サイクルは発生しえない．

特徴 3：同期制御上の理由（矛盾あるいはデッドロックの発生）によって，グローバル・トランザクションがアボートされることはない．

ここで提案した方式は，特徴 1 の説明から明らかなように，トランザクションの実行途中に矛盾が発

生するのを許すものではない．したがって，矛盾の発生によってグローバル・トランザクションがアボートされることはない．

デッドロックによるアボートに関しては，まず，特徴2より，グローバルなデッドロックは発生しない．また，ローカルなデッドロックが発生した場合，3.2.2 の(3)で示したデッドロックの解決方法より，グローバル・トランザクションがアボートされることはない．したがって，デッドロックによって，グローバル・トランザクションがアボートされることもありえない．

特徴4：同期制御のための通信量は時刻印の分だけである．

4. 他の方式との比較

3章で提案した方式を，2相ロック方式および多版時刻印方式と比較する．

ローカルな同期制御方式としては，提案方式は，3.2.1 の(2)，(3)で説明したように2相ロック方式とほとんど同じであり，したがって，アボート発生頻度が少なく，並列走行性も時刻印方式に比べてそれほど劣らない優れた制御方式であることが分かる．

グローバルな同期制御方式としての優劣は，3つの観点，すなわち，①グローバル・トランザクションのアボート発生頻度，②グローバル・トランザクションの並列走行性，③グローバル・トランザクションの同期制御用の通信量を基に比較する．比較結果を**表2**にまとめる．

表より，各方式の優劣を判断する．

グローバル・トランザクションのアボート発生頻度に関しては，提案方式が頻度0であり，優れている．次いで，2相ロック方式が優れている．多版時刻印方式は，前記2方式に比べると，アボート発生頻度は大きい．

表2 グローバル・トランザクションに対する同期制御方式の比較
Table 2 Comparison of concurrency control techniques for global transactions.

比較項目 ＼ 制御方式	グローバル・トランザクションのアボートの発生頻度	並列走行性	同期制御用通信量
2相ロック方式	少：デッドロック発生時のみアボート	中：競合資源での待ち有	大：デッドロック解析用の待ちグラフ情報
多版時刻印方式	多：古い時刻印のWRITEはアボート	高：待つのは読込みでのコミット待ちのみ	小：時刻印のみ
提案方式	0：デッドロック発生時もアボート不要	低：グローバル・トランザクション直列化	小：時刻印のみ

グローバル・トランザクションの並列走行性に関しては，多版時刻印方式が優れているが，それと2相ロック方式との差は，大きくない．提案方式は，以下に述べる3つの原因により，他の2方式に比べ並列走行性が劣化する．

①サイト内で2相ロックでウェイトしうる．

②各サイトでルート・グローバル・トランザクションは1度に1つしか走行できない．

③未終了のルート・グローバル・トランザクションより時刻印の新しいグローバル・トランザクションの多版READは，時刻印の古いルート・グローバル・トランザクションがすべて終了（コミット／アボート）するまで待たされる．

グローバル・トランザクションの同期制御用の通信量に関しては，2相ロック方式が悪い．他の方式については，大差はない．

以上をまとめると，本検討で提案した方式は，①ローカルな同期制御方式としては，トランザクションのアボート発生頻度の小さい優れた方式であり，②グローバルな同期制御方式としても，グローバル・トランザクションのトラフィックの小さい環境において，グローバル・トランザクションのアボート発生頻度が0であり，同期制御に要する通信量が小さいという長所を持つ優れた方式であることが示された．

5. むすび

本論文は，グローバル・トランザクションのデータベース更新能力に，更新は起源サイトに限るという制約を課して，その制約の下での同期制御方式の検討を行ったものである．

検討の結果，ローカル・トランザクションとルート・グローバル・トランザクションのREAD/WRITEの制御には2相ロック方式を修正適用し，子孫グローバル・トランザクションのREADの制御には多版時刻印方式を修正適用し，さらに，各サイトでグローバル・トランザクションを時刻印順に直列化することによって，全サイトの全トランザクションの直列化可能性を保証する方式を提案した．従来提案されている他の方式と比較した結果，この方式は，①ローカルな同期制御方式としては，トランザクションのアボート発

生頻度の小さい優れた方式であり，②グローバルな同期制御方式としても，グローバル・トランザクションのトラフィックの小さい環境において，グローバル・トランザクションのアボート発生頻度が0であり，同期制御に要する通信量が小さいという長所を持つ優れた方式であることを示した．

今後の研究課題としては，単調に増加する版の管理方法，および提案方式の性能の定量的評価等があると考える．

謝辞 本研究の機会を与えてくださった NTT 情報通信処理研究所専用システム部松永部長，伊土主幹員，ならびに内容について多くの助言，コメントを頂いた寺中主幹員，田中主幹員に深謝いたします．

参考文献

1) Bernstein, P. A. and Goodman, N.: Concurrency Control in Distributed Database Systems, *Comput. Surv.*, Vol. 13, No. 2, pp. 185-221 (1981).
2) Ceri, S. and Pelagatti, G.: *DISTRIBUTED DATABASES Principles and Systems*, McGraw-Hill, New York (1985).
3) Eswaren, K. et al.: On the Notions of Consistency and Predicate Locks in a Relational Database Systems, *Comm. ACM*, Vol. 19, No. 11, pp. 624-633 (1976).
4) Obermarck, R.: Distributed Deadlock Detection Algorithms, *ACM Trans. Database Syst.*, Vol. 7, No. 2, pp. 187-208 (1982).
5) Stonebraker, D. and Neuhold, E.: A Distributed Version of INGRES, *Proc. 2nd Berkley Work. on Distributed Data Management and Computer Networks*, pp. 19-36 (1977).
6) Berstein, P. A. et al.: Concurrency Control in a System for Distributed Databases (SDD-1), *ACM Trans. Database Syst. T*, Vol. 5, No. 1, pp. 18-51 (1980).
7) Reed, D. P.: Implementing Atomic Actions on Decentralized Data, *ACM Trans. Computer Systems*, Vol. 1, No. 1, pp. 3-23 (1983).
8) Chan, A. et al.: Overview of an ADA Compatible Distributed Database Manager, *Proc. ACM SIGMOD 1983*, San Jose, CA., pp. 228-237 (1983).

(昭和62年11月11日受付)
(昭和63年4月14日採録)

池田 哲夫（正会員）

昭和31年生．昭和54年東京大学理学部情報科学科卒業．昭和56年同大学院修士課程修了．同年，日本電信電話公社電気通信研究所入所．現在，NTT 情報通信処理研究所専用システム研究部主任研究員．この間，プログラミング言語の意味論の研究，データベース管理システムの研究開発などに従事．ACM 会員．