

07-開-11

データベース構築促進及び技術開発に関する報告書

モバイルデータベースシステム
に関する調査研究

平成8年3月

財団法人 データベース振興センター

委託先 株式会社イフ・アドバタイジング



この事業は、競輪の補助金を受けて実施したものです。

序

データベースは、わが国の情報化の進展上、重要な役割を果たすものと期待されている。今後、データベースの普及により、わが国において健全な高度情報化社会の形成が期待される。さらに海外に対して提供可能なデータベースの整備は、国際的な情報化への貢献および自由な情報流通の確保の観点からも重要である。しかしながら、現在わが国で流通しているデータベースの中でわが国独自のものは1/3にすぎないのが現状であり、わが国データベースサービスひいてはバランスある情報産業の健全な発展を図るためには、わが国独自のデータベースの構築およびデータベース関連技術の研究開発を強力に促進し、データベースの拡充を図る必要がある。

このような要請に応えるため、財データベース振興センターでは日本自転車振興会から機械工業振興資金の交付を受けて、データベースの構築および技術開発について民間企業、団体等に対して委託事業を実施している。委託事業の内容は、社会的、経済的、国際的に重要で、また地域および産業の発展の促進に寄与すると考えられているデータベースの構築とデータベース作成の効率化、流通の促進、利用の円滑化・容易化などに関係したソフトウェア技術・ハードウェア技術である。

本事業の推進に当って、当財団に学識経験者の方々に構成されるデータベース構築・技術開発促進委員会（委員長 前山梨学院大学教授 蓼沼良一氏）を設置している。

この「モバイルデータベースシステムに関する調査研究」は平成7年度のデータベースの構築促進および技術開発促進事業として、当財団が㈱イフ・アドバタイジングに対して委託実施した課題の一つである。この成果が、データベースに興味をお持ちの方々や諸分野の皆様方のお役に立てば幸いである。

なお、平成7年度データベースの構築促進および技術開発促進事業で実施した課題は次表のとおりである。

平成8年3月

財団法人 データベース振興センター

平成7年度 データベース構築・技術開発促進委託課題一覧

分野	課題名	委託先
社会	1 法的データベースにおける多分野データベースの統合体的管理とオフ・オンラインの融合化に関する調査、研究	(株)日本法律情報センター
	2 新聞記事分類キーワードの標準モデル構築と自動付与に関する調査研究	(株)エレクトロニック・ライブラリー
中小企業振興 地域活性化	3 パソコンを用いた地図データベースの基礎構築	(有)朝日データサービス
	4 景観シミュレーション用樹木のデータベース構築	(株)ストウディオサカイ
	5 包装機械データベースの構築	(社)日本包装機械工業会
	6 新規事業創出支援のためのデータベース構築に向けた基礎調査	(株)日本インテリジェントトラスト
	7 高効率化先進材料ファクトデータベースのパッケージ化	(財)次世代金属・複合材料研究開発協会
	8 阪神・淡路大震災の情報デジタル・アーカイブ	神戸マルチメディア・インターネット協議会
技術	9 Mosaicの利用によるマルチメディアデータベース検索システムの構築	日本電子開発(株)
	10 大規模データベースにおける構造化情報抽出方式の調査	(株)日本総合研究所
	11 モバイルデータベースシステムに関する調査研究	(株)イフ・アドバタイジング
	12 情報収集ロボットによるInternetでのWWW所在検索データベースの構築	日外アソシエーツ(株)
	13 インハウスデータベース用CGI作成の調査研究	日本電子計算(株)

目 次

I. 課題性の研究	1
II. モーバイルデータベースシステム	3
1 章 序	3
2 章 システムモデル	6
2.1 データベースシステム	6
2.2 クライアントサーバモデル	11
2.3 モーバイル通信	13
2.4 モーバイルクライアントとモーバイルサーバ	15
3 章 検索方式	17
3.1 モーバイルオブジェクト	17
3.2 検索手順	21
3.3 レプリカ変更手順	22
4 章 トランザクション管理方式	24
4.1 概要	24
4.2 トランザクションの定義	27
4.3 実行履歴	33
4.4 独立性	35
4.5 競合演算	39
4.6 同時実行制御	42
4.7 楽観的二相ロック方式	58
5 章 データの配置方法	60
5.1 テーブル分散方法	60
5.2 垂直断片化	61
5.3 水平断片	63
5.4 冗長化	64

5.5 視野	64
6章 モーバイルステーションによるデータベースサービス.....	66
参考文献.....	70
付記.....	76

I. 課題性の研究

1. 目的

情報通信技術の進展によって、今日、情報通信ネットワークの活用が急速に拡大しつつあるが、その多くは有線による位置固定のコンピュータ間を結合するものである。確かに、無線LANの利用も多くの方で見られるようにはなったが、それでもコンピュータおよび関連機器は、通常、設置場所が固定されている。

こうした中で、最近、無線等を利用した移動型の情報通信システムが大きな脚光を浴びており、特に、パーソナル・ハンディ・システム（PHS）等の進展によって急速に普及しようとしている。

膨大な情報を効率的に活用しなければならないユーザにとって、移動型ステーションを通じて必要な場所から、必要な時に即座に様々なデータベースにアクセスできれば、さらにサーバ内のデータの更新もできれば、ユーザの利便性が飛躍的に向上する。しかし、こうした場合、位置が固定されない移動型のシステム間での通信処理（モバイルコンピューティング）が重要となる。また移動型ステーションは小型・軽量であるだけに、サーバとクライアント間の通信量および通信時間の軽減が必要とされる。

このため、固定位置のデータベースサーバを移動型ステーションのクライアントが利用する問題について様々な側面から考察する。さらに、こうしたシステムを利用した有効なデータベースサービスについて検討を行うとともに、併せて既存の大規模ネットワーク、たとえばインターネット等の融和性についても考察する。

以上の調査・研究によって、わが国のデータベース振興に資することを目的とする。

2. 実施内容

本調査研究では、以下の項目を主たる内容としている。

1) 移動型検索方式

移動型ステーションを通じて、データベースサーバ内のデータを検索する方式について検索する。ステーションとサーバ間の通信量を減少するためにサーバ内のデータをステーション内にキャッシングすることにより、通信量を減少する方式を検討する。

2) 移動型トランザクション管理方式

複数のデータベースサーバ内のデータを更新する方式を検討する。特に、利用者が移動しながらサーバを利用するときに、サーバ内のデータの一致性を保つための同期方式を検討する。

3) データの配置方法

移動型ステーションから、データベースサーバを利用するために、データを複数のサーバにどのように配置するかについて検討する。

4) 移動型ステーションによるデータベースサービス

移動型ステーションを利用した有効なデータベースサービスについて検討を行う。

3. 実施体制

株式会社イフ・アドバタイジング内に当該事業遂行のための「特別調査・研究委員会」を設置し、事業の実施に当たった。

調査研究委員会

滝沢 誠（東京電機大学理工学部・経営工学科教授）

山鳥雄嗣（日本情報処理開発協会・調査部長）

市川 隆（A I・ファジー振興センター所長）

道田国雄（イフ・アドバタイジング調査部長）

尾崎晴雄（シネ・ジャーナルプロダクション企画部長）

Ⅱ．モバイルデータベースシステムの研究

1 章 序

インターネット、ATM光ネットワーク等の通信ネットワークの普及により、組織体内、国内のみならず世界中のコンピュータが相互接続され、利用されてきている。こうした通信ネットワークの普及のなかで、ワイヤレスの無線を用いた通信が重要となってきた。これまでの通信は、通信ネットワークに有線により接続された位置固定のコンピュータ間で行われてきた。各コンピュータには、ネットワーク内の物理的な位置を示すアドレスが与えられ、このアドレスを用いて通信対象のコンピュータを定めてきた。これに対して、通信技術の進歩により、これまでの有線を用いた通信から、無線等を利用することにより、モバイル通信が普及してきている。自動車電話、携帯型電話、PHS等の普及により、各個人が移動しながら電話を行えるようになってきている。こうした無線を用いることにより、ネットワーク内での位置の固定されない移動型システム間での通信と処理（モバイルコンピューティング）が重要となってきた。

インターネット等の通信ネットワークの普及により、世界中の情報を利用できるようになってきている。通信ネットワークでは、情報の格納、管理しているデータベースシステムが重要な資源である。これまでのデータベースシステムとしては、リレーショナルモデル [CODD70] のシステムがパソコン、ワークステーション、汎用大型コンピュータまで広く利用されてきた。また、新しいデータベースシステムとして、オブジェクト指向のシステムも利用されてきている。これに対して、インターネットの世界では、単純な Unix, DOS/V, NT Windows 等のファイルを用いた WWW(world wide web) が広範に利用されてきている。これらを用いて、書籍、株、住宅、就職、大学のカリキュラム等の種々の情報を獲得できるとともに、ショッピング、ホテルの予約等の業務を行うことができる。このように通信ネットワークでは、情報の格納、管理しているデータベースシステムをどのように利用していくかが重要な課題となっている。

こうしたモバイルコンピューティングの中で、移動型のコンピュータを通じて、どのようにデータベースシステムを利用していくかが課題となってきた。例えば、位置が固定のデータベースサーバを、移動型ステーションのクライアントが利用する問題について考える。移動型コンピュータは、各個人が持ち運ぶものであることから、小型で軽量である。このために、コンピュータを長時間動作させることに十分なバッテリーを持つことができないので、サーバとクライアント間での長時間の通信を行うことができない。このために、移動型ステーション内のクライアントとサーバ間の通信量と通信時間を減少させる必要がある。どのように移動型クライアントから、位置固定のデータベースシステムサーバを利用するかについて、論文 [GRUB94, HUANG94, IMIE94, JING95, YEO94] で論じられている。

今後の情報システムでは、各個人が携帯型のコンピュータを持ち、これらに個人の種々の情報を記憶している個人データベースシステムが主要な構成要素となっていくものと思われる。ICカードのように、地下鉄、電車での切符の購入、銀行の預貯金、ビルの入退管理等に、これらの携帯コンピュータが利用されていく。こうした情報システムでは、個人の移動にともない、携帯コンピュータ内の個人データベースシステムも移動していく。即ち、クライアントのみならずデータベースサーバも、通信ネットワーク上を移動していく形態となる。本調査研究では、クライアントとともに、移動型のサーバも考えたデータベースシステムについて、以下を研究する。

- (1) 検索方式。
- (2) トランザクション管理方式。
- (3) データの分散方式。

検索方式では、移動型のクライアントから、移動型のデータベースサーバ内のデータを検索する方式について考える。トランザクション管理方式では、複数のデータベースサーバ内のデータを更新する方法について考える。特に、クライアントとサーバ間の通信コネクションがバッテリーの容量の制限と通信のノイズから切断された状況下でも、データベースシステム間のインテグリティ制約を保つ方法について論じる。複数のデータベースサーバに、データを分散する方法について考える。

以上から、本調査研究では、移動型ステーションを通じて、複数のデータベースサーバを有効に利用する方法について研究し、もってわが国のデータベースの振興に資すること

を目的とする。

本報告では、まず、第2章で、モバイルデータベースシステムのモデルについて述べる。ついで、第3章で、モバイルデータベースシステムの検索方式について述べる。第4章では、モバイルデータベースシステムにおけるトランザクション管理方式について述べる。

最後に、モバイルオブジェクトのトランザクション管理方式の資料を添付する。

2章 システムモデル

モバイル通信のもとでデータベースシステムを利用するモバイルデータベースシステム (mobile database system) のモデルについて述べる。

2.1 データベースシステム

データベースシステムは、データベースに対して、SQL等のデータ操作演算により、データ操作を複数の利用者に行わせるとともに、データベース内のデータのインテグリティ制約を保つためのシステムである。

現在のデータベースシステムの多くは、リレーショナルモデルに基づいている。利用者からデータベースシステムがどのようにみえるかを示したものがデータモデルである。現在の主要なデータベースシステムは、リレーショナルモデルに基づいている。リレーショナルモデル [CODD70] は、データ構造、データ構造に対する操作演算、データ構造が保つべき制約としてのインテグリティ制約により与えられる。

2.1.1 データ構造

リレーショナルモデルのデータ構造は、テーブルから構成される。例を図2.1に示す。従業員の情報を示すテーブル Emp は、従業員の番号 (eno)、名前 (ename)、給料 (sal)、所属している部の番号 (dno) に関する情報を持っている。これらを属性またはカラムとする。テーブル名と属性の構成をテーブルのスキーマとする。テーブル Dept は、部についての番号 (dno)、名前 (dname) の情報を持っている。テーブル内の各情報、例えば、<1, A, 150, 1> を行とする。

テーブル内の各属性の値は、原子的 (atomic) でなければならない。即ち、各属性の値として、集合、リストを取ることができず、一つの値を取れるだけである。例えば、図2.1で、ename の値として、A, B という値を取れるが、集合 {A, B, C} を値として取ることはできない。以上の性質を満足しているテーブルを正規形であるという。

Emp	eno	ename	sal	dno
	1	A	200	1
	2	B	300	1
	3	C	400	2
	4	A	250	3
	5	D	300	2

Dept	dno	dname
	1	A
	2	B
	3	C
	4	A
	5	D

図2.1 テーブルの例

2.1.2 SQL

SQL(structured query language) [SQL] は、第一階論理に基づいた非手続き的な高水準言語であり、国際標準言語である。SQL は、スキーマの定義、消去を行うデータ定義機能と、テーブル内の行の操作を行うデータ操作機能を持つ。

A. データ定義

create 文により定義できる。属性 eno、sal、dno のデータ型は整数 (int) であり、ename は文字列 (char) である。また、eno は、空値を取れない。

```
create table Emp
```

```
( eno int NOT NULL, /* 属性 eno のデータ型は整数で、空値が許されない */
  ename char(36),   sal int,   dno int )
```

また、テーブル Emp の定義を除くことは、以下のようにして行える。

```
drop table Emp
```

B. データ操作

次に、テーブル Emp と Dept に対する SQL によるデータ操作（検索と更新）の例を説明する。

(1) DDD 部員のなかで、給料が200以上の者とその給料を求めよ。

```
select  ename, sal  from  Emp, Dept
where   dname = "DDD" and Dept.dno = Emp.dno and sal >= 200
```

Dept.dno は、dno がテーブル Dept の属性であることを示す。また、SQL では以下のようにも書ける。in の右側には select... という検索式が書かれている。このような検索式を入れ子形式という。この in は英語の関係代名詞を示し、dno を修飾している。

```
select  ename, sal  from  Emp
where   sal >= 200 and
        dno in (select dno from Dept where dname = "DDD")
```

(2) DDD 部員のなかで、給料が200以上の者とその給料を、給料の昇順に求めよ。

```
select  ename, sal  from  Emp
where   sal >= 200 and
        dno in (select dno from Dept where dname = "DDD")
order by sal
```

- (3) 部毎の平均給料を求めよ。

```
select dno, avg(sal) from Emp
group by dno
```

- (4) DDD 部員の給料を1.5倍にする。

```
update Emp set sal = sal * 1.5
where dno in ( select dno from Dept where dname = "DDD")
```

- (5) DDD 部員を消去する。

```
delete from Emp
where dno in ( select dno from Dept where dname = "DDD");
```

- (6) 行<11, K, 210, 1>をテーブル Emp に追加する。

```
insert into Emp ( eno, ename, sal, dno)
values ( 11, K, 210, 1)
```

- (7) DDD 部員として、Kを追加する。

```
insert into Emp ( eno, ename, sal, dno)
select 11, "k", 210, dno from Dept
where dname = "DDD"
```

以上のように、SQLを用いると、非手続き的にデータ定義とデータ操作を行える。

2.1.3 視野

リレーショナルモデルでは、データベース内のテーブルを基本テーブルとする。利用者は、これらの基本テーブル内の一部を利用する。利用者が利用の度に、テーブルの結合を行うことは、容易でない。このために、利用者が必要とするデータを基本テーブルから集めた仮想的なテーブルを定義する機能がある。この仮想的テーブルを視野 (view) とする。例えば、Emp と Dept の上に、従業員名と彼の属している部名を与える視野は以下のよう定義できる [図2.2]。

```
create view ED ( name, dname)
as select ename, dname from Emp, Dept
where Emp.dno = Dept.dno
```

検索演算「DDD 部の部員を求めよ」は、ED を用いて次のように書ける。

```
select name from ED where dname = "DDD"
```

ここで、利用者は、Emp と Dept の結合を行う必要がない。データベースシステムは、この SQL から、視野の定義文を用いて、以下の SQL を作成し、実行する。

```
select ename from Emp, Dept
where Emp.dno = Dept.dno and dname = "DDD"
```

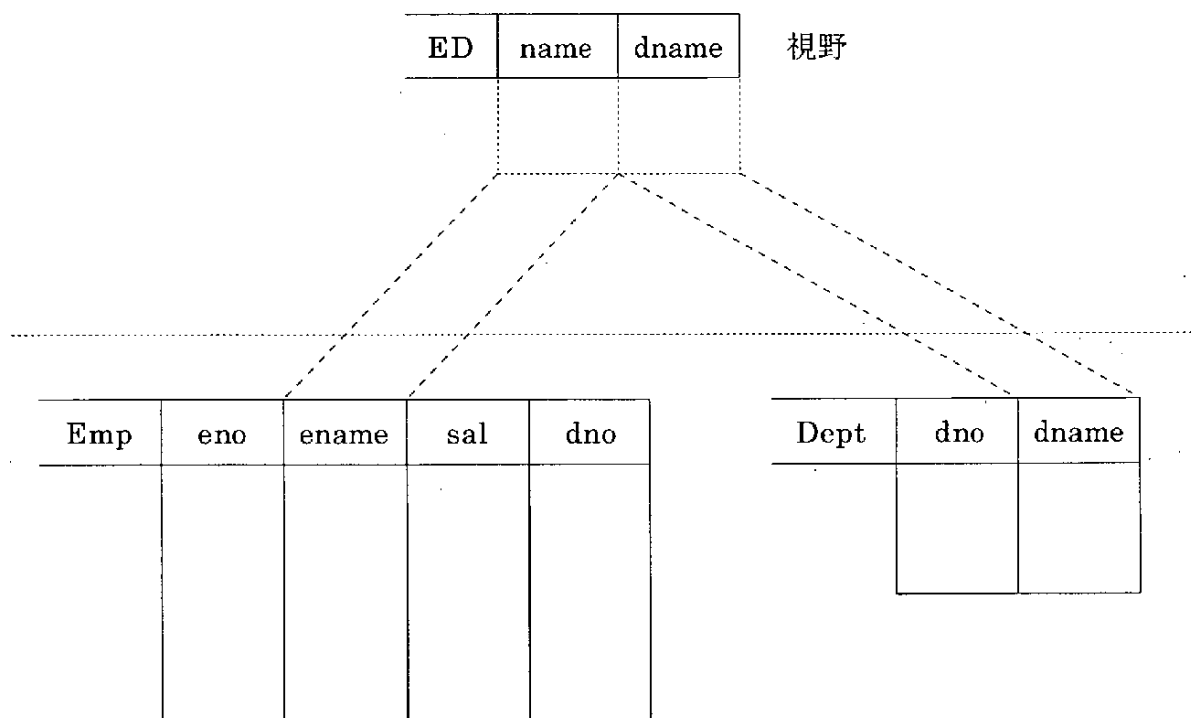


図2.2 視野

視野を通して検索を行えるが、一般に更新を行うことはできない。

2.2 クライアントサーバモデル

現在のデータベースシステムは、データベース管理システム (DBMS) の機能のなかの利用者インタフェースの部分をクライアントに、またデータベースに対する基本操作部分をサーバに機能分散された形態をとっている。データベースシステムは、複数のクライアントと複数のサーバがローカルエリアネットワーク (LAN) 等の通信ネットワークにより相互接続されている [図2.3]。

データベースサーバは、データベースに対する基本的な操作を行うものである。即ち、SQL をデータベースに対して実行するとともに、データベースのインテグリティ制約を保ちながら複数の利用者の SQL を同時実行し、障害から復旧することを行う。これに対してクライアントは、ユーザとのインタフェースの機能を持つ。ユーザの応用プログラムは、クライアントで実行され、この中のデータベースに対する操作演算 (SQL) は、サー

バに送信される。サーバは、データベースの操作演算を受信すると、これを実行し、結果をクライアントに返す。以上のように、クライアントの応用プログラムから見ると、サーバを関数呼び出しすることにより、データベースを利用しているように見える。これを、遠隔手続き呼び出し (RPC: remote procedure call) という [図2.4]。

これまでのデータベースシステムでは、複数のクライアントとサーバが、通信ネットワークにより接続され、各クライアントから、通信ネットワークを通じて、一つ以上のサーバを操作できる。Oracle, Sybase 等の現在のデータベースシステムは、クライアントとサーバから構成されている。ここでは、各データベースシステムは、リレーショナルモデルに基づいたものであるとする。

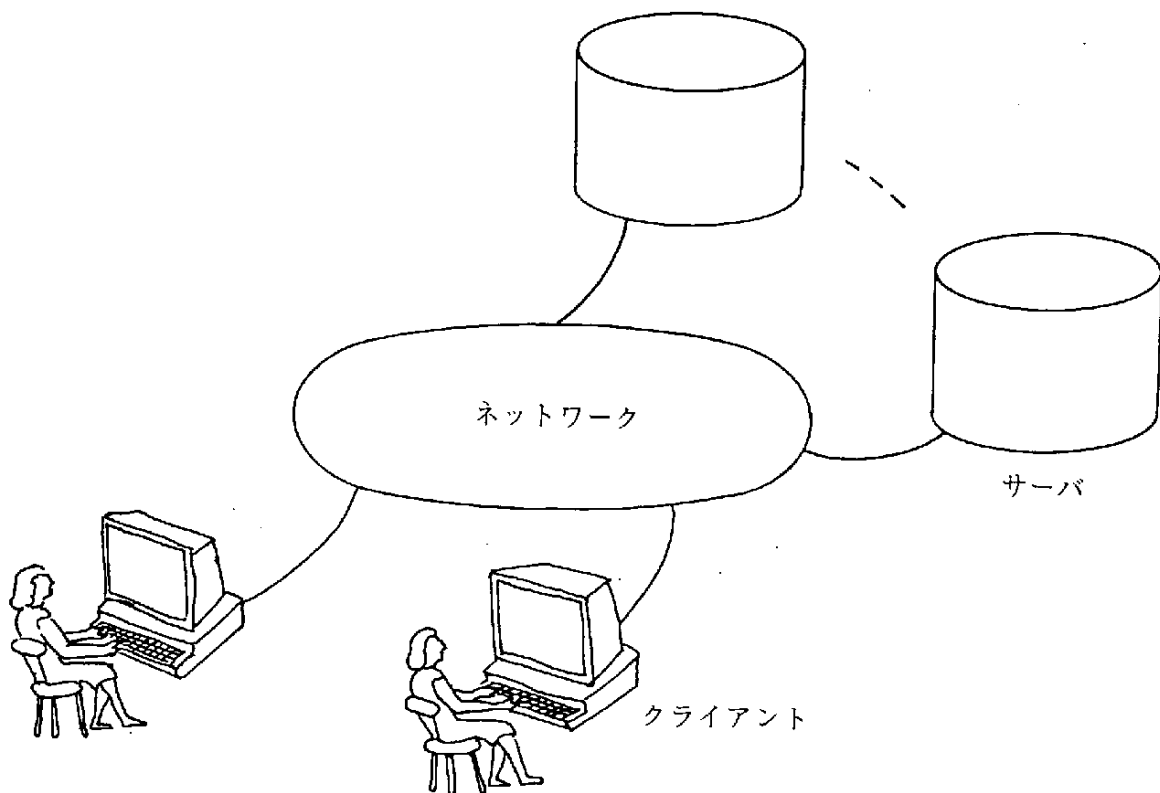


図2.3 クライアントサーバモデル

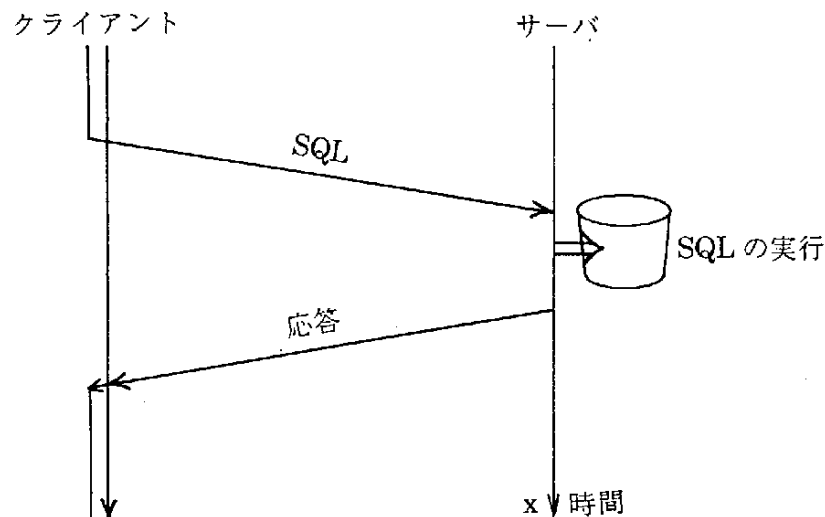


図2.4 遠隔手続き呼び出し (RPC)

2.3 モーバイル通信

これまでの通信ネットワークは、有線のネットワークであり、サーバもクライアントも、ネットワーク上の固定の位置に接続されていた。例えば、出張先から事務所にあるコンピュータを利用するためには事務所に電話線等を用いて接続してから利用する必要がある。これに対して、現在、携帯電話、自動車電話、PHS等の無線を用いたモーバイル通信技術が発展し、普及してきている。これらの通信では、任意の地点から任意の時に通信を行える。また、コンピュータも小型化と高性能化、高機能化が進み、手帳サイズでも現在のワークステーション並みの機能を有した超小型コンピュータが登場しようとしている。これらとモーバイル通信の融合されたシステムが、これまでの有線のコンピュータシステムに代わりつつある[図2.5]。ここでは、コンピュータをステーションと呼ぶことにする。

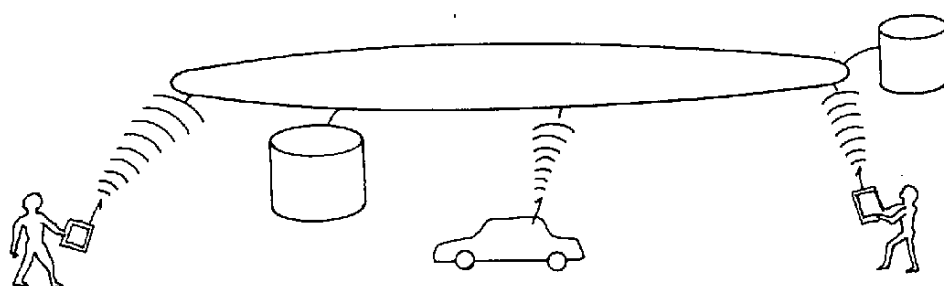


図2.5 モーバイ通信

ネットワークは、セルという一定の大きさの領域から構成されている。各セル内のステーションは、モバイルサポートステーション (MSS) と呼ばれるシステム要素と無線により通信を行う。セル内で、ステーションは、自由に移動を行うことができる。MSS は、光ネットワーク等の高速通信ネットワーク、インターネット等で接続されている。ステーションが、あるセル A から他のセル B に移動したとする。このとき、ステーションは、A の MSS にかわり、B の MSS と通信を行うことにより通信を継続できる [図2.6]。通信を行っている二つのステーションが移動を行っても、二つのステーション間の通信コネクションは、MSS 間の通信により保持される。このための種々の通信プロトコルが示されている。従って、通信しあう二つのステーションが互いに通信ネットワーク内を移動しても、二つのステーション間の通信コネクションは、保持されている。

利用者が利用する携帯型のステーションは、小型で軽量であることから、容量の小さいバッテリーが搭載されている。無線の通信を行うと電力を消費することから、長時間の通信を行うことができない。これまでの有線により接続されたステーションでは、電力の消費を考慮する必要がなかったが、無線通信を携帯型ステーションで行う場合には、ステーションの無線通信のための電力消費を考慮する必要がある。消費電力を低減するための手法として、携帯型のステーションとデータベースサーバ間でのコネクションを切断する方法がある。携帯型ステーションとサーバ間での応用レベルでの処理は行われているが、通信コネクションを切断するものである。携帯型ステーション内に、データベースサーバ内のデータをダウンロードし、サーバと通信を行わずに、携帯型ステーション内でローカルに処理を行わせるものである。

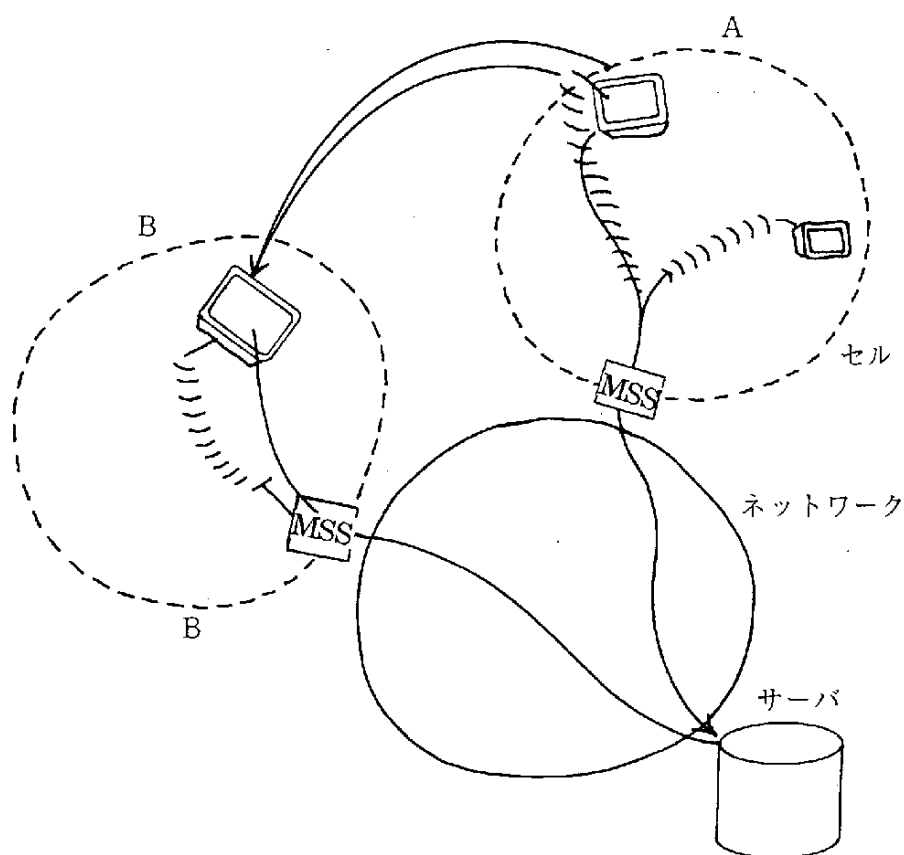


図2.6 モーバイ通信ネットワーク

2.4 モーバイクライアントとモーバイサーバ

モーバイステーションとは、ネットワーク上の位置が変化するステーションである。これに対して、固定ステーションとは、ネットワーク上の位置が変化しないステーションである。データベースシステムがクライアントとサーバから構成されることから、クライアントとサーバに、以下の種類がある。

- (1) モーバイクライアント
- (2) 固定クライアント
- (3) モーバイサーバ
- (4) 固定サーバ

モーバイクライアントとは、携帯型端末に実装されているクライアントである。利用

者の移動にともないクライアントも移動する[図2.7]。一般に、モバイルクライアントは、小型、軽量であることから、容量の十分大きいバッテリーを装備しておらず、長時間の通信を無線を用いて行えない。

一方、固定クライアントとは、従来のようにワークステーション内のクライアントである。ワークステーションのネットワーク内での位置は固定であることから、クライアントは移動しない。

固定サーバも、ワークステーション内で位置固定のサーバである。これまでのデータベースシステムサーバは、この固定型サーバである。これに対して、モバイルサーバとは、ネットワーク内を移動するサーバである。これまでのサーバが、固定型であるのに対して、利用者の携帯型の端末内に構成されてきた個人用のデータベースシステム、監視ロボット内に蓄積される監視データ、自動車、飛行機等の移動体内のデータベースシステム等が、モバイルデータベースシステムの例である。これまでの固定型データベースサーバに対して、移動を行いながらデータベースサービスを提供するサーバが、モバイルサーバである。今後、これまでの固定型データベースサーバに加えて、モバイルデータベースサーバを考えていく必要がある。

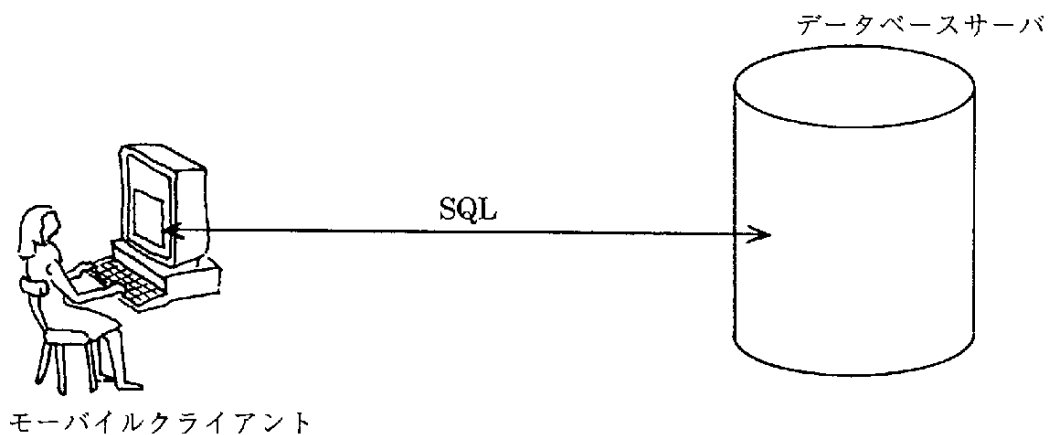


図2.7 モバイルクライアントとサーバ

3 章 検索方式

どのようにモバイルデータベースシステムで、検索を行うかについて考える。

3.1 モーバイロbject

クライアントとサーバが移動していても、クライアントから特定のサーバにコネクションを開設し、通信を行うことは、現在の通信技術で可能である。クライアントが移動しようとしないうちに、クライアントからデータベースサーバにコネクションを開設し、データベースを利用できる。

モバイルデータベースシステムでは、以下の問題を考える。

- (1) 同じまたは同様なデータを持つ複数のサーバが存在する。
- (2) このとき、一番有効なサーバを特定し、利用する。

ここで、データベースシステム内のデータの単位をオブジェクトと呼ぶことにする。オブジェクトは、リレーショナルデータベースシステムでは、テーブル、行、行内の属性値である。何をオブジェクトとするかは、データベースシステムの設計の問題となる。また、オブジェクト単位にデータがデータベースサーバに分散されたとする。ここで、あるオブジェクト o が、複数のサーバ $s_1, \dots, s_m$ 内に記憶されているとき、各サーバ s_i に記憶されているものを o のレプリカ (replica) o_i とする。オブジェクト o のレプリカの集合を、 $r(o) = \{ o_1, \dots, o_m \}$ とする。レプリカは、異なったサーバに記憶されていたとしても、互いに同一の値を示さねばならない。図3.1には、テーブルAが三つのサーバ s_1, s_2, s_3 に冗長に存在している例を示す。

[定義] 各レプリカ o_i と o_j が同一の値であるとき、レプリカは、互いに無矛盾 (consistent) である、または一致しているという。

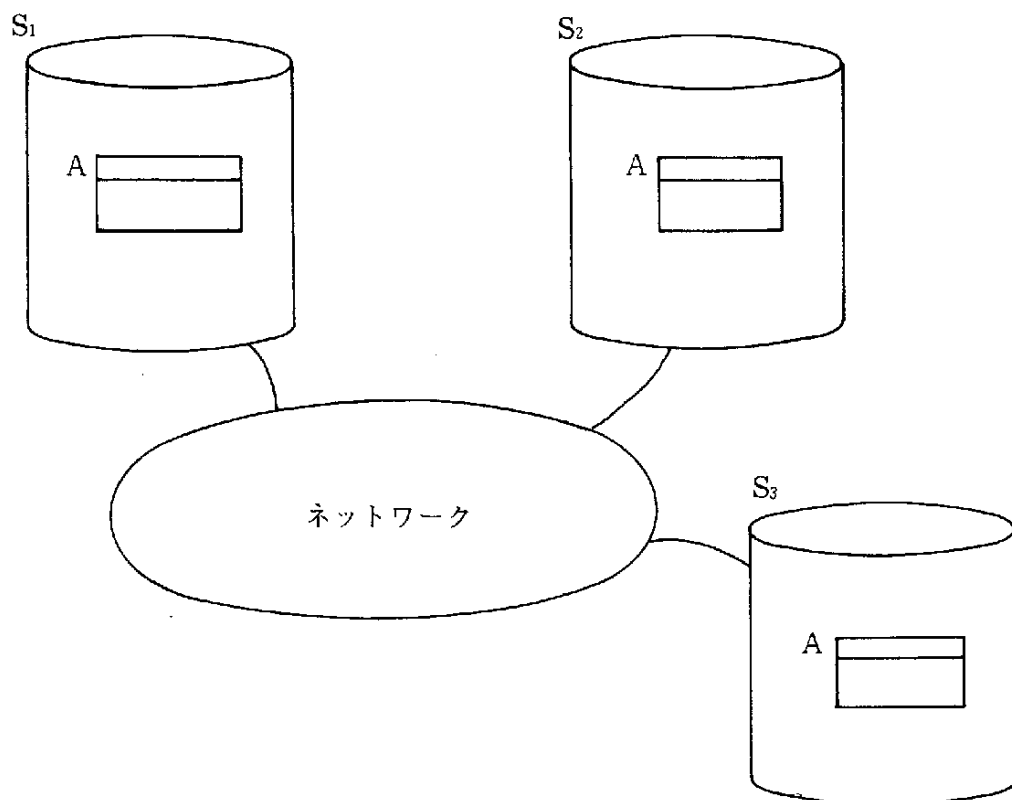


図3.1 レプリカ

また、 o_i と o_j が一致していないとき、 o_i と o_j は矛盾 (inconsistent) しているという。

各レプリカ o_i は、移動型ステーション内に存在する場合を、モバイルレプリカとする。モバイルレプリカ o_i は、ステーションの移動にともない、ネットワーク内の位置が変化する。また、レプリカ o_i がサーバ s_i から他のサーバに移動するときも、 o_i の位置が変化する。このように、レプリカ (オブジェクト) のネットワーク内での位置 (アドレス) が変化するものを、モバイルレプリカ (オブジェクト) とする。こうしたオブジェクトの移動とは何かについて考える。各オブジェクトは、あるサービスを提供するものである。具体的には、リレーショナルモデルを考えていることから、オブジェクトに対する SQL を提供している。オブジェクトの移動にともない、クライアントとオブジェク

トの存在するステーション間のネットワークが変化することにより、クライアントとステーション間の伝送速度、伝送品質等が変化する。オブジェクトが提供するサーバの品質を、サービスの品質 (QoS: Quality of Service) とする。オブジェクト A が、図3.2に示すように、ネットワーク N から M に移動したとする。ネットワーク M は 1Mbps であり、N は 64Kbps の伝送速度があるとする、M は N よりも伝送速度が早いとすると、利用者に対するサービス品質としての伝送速度が向上したことになる。このように、利用者からみて、オブジェクトの移動にともない、サービス品質 (QoS) が変化することが重要な点となる。

以上から、オブジェクトの移動を以下のように定義する。

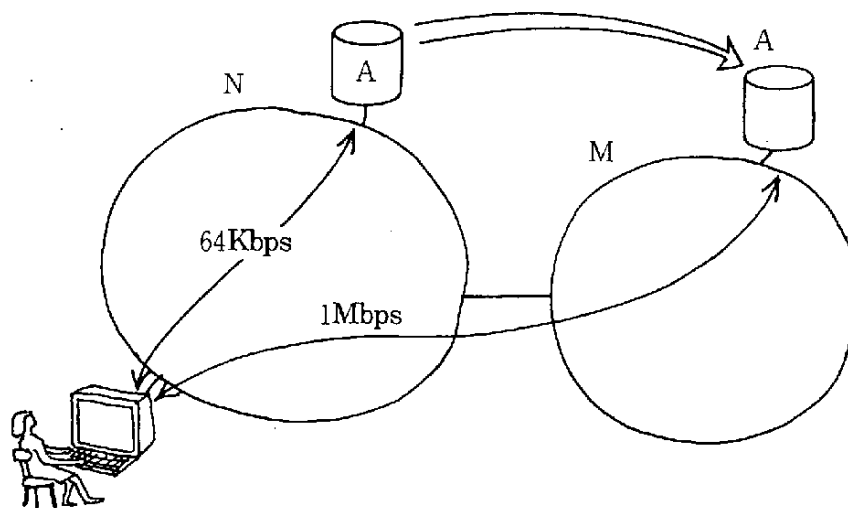


図3.2 レプリカの移動

〔定義〕 オブジェクトのサービス品質 (QoS) が変化したとき、オブジェクトは移動いたとする。

クライアントは、オブジェクト o のあるレプリカ o_i を利用しているとする。レプリカ o_i はデータベースサーバ s_i 内にある。クライアントの移動またはレプリカを提供するサーバ s_i の移動に伴い、クライアントが o_i から受けるサービスの品質 (QoS) が変化する。ここで、以下を定義する。

$QoS(c, o_i)$ = クライアント c が、オブジェクトのレプリカ o_i から受けるサービス品質 (QoS)。

例えば、サービス品質の例として、クライアント c とレプリカ o_i 間の以下のものがある。

- (1) 伝送速度、
- (2) 応答時間、
- (3) 通信遅延時間、
- (4) 通信品質 (通信誤り率)。

これらの中で、何をサービス品質として用いるかは、クライアントの応用により決まる。一方、クライアントは、オブジェクト o に対して、あるサービス品質を要求している。これを以下のように書く。

$ReQ(c, o)$ = クライアント c が、オブジェクト o に対して要求しているサービス品質。

ここで、 q_1 と q_2 を、二つのサービス品質とする。 q_1 が q_2 よりも、サービス品質が高いとき、 $q_1 \geq q_2$ と書くことにする。

クライアント c が、オブジェクト o を利用することを考える。オブジェクト o の複数のレプリカが通信ネットワーク内に存在している。従って、クライアント c は、レプリカ集合 $r(o) = \{o_1, \dots, o_m\}$ の中から、あるレプリカ o_i を選択して、 o_i を利用する。このとき、 $r(o)$ 内のレプリカの中で、以下の条件を満足するものを選択する。

[レプリカ利用条件] $ReQ(c, o) \leq QoS(c, o_i)$ 。

即ち、クライアント c がオブジェクト o に対して要求しているサービス品質 $ReQ(c, o)$ を提供しているレプリカ o_i を $r(o)$ から見つける必要がある。

クライアント c は、レプリカ利用条件を満足しているレプリカの中で、最大のサービス品質を提供しているレプリカを見つけ、利用する。

3.2 検索手順

クライアント c が、オブジェクト o のレプリカ集合 $r(o) = \{o_1, \dots, o_m\}$ の中から検索対象のレプリカ o_i を求める手順を以下に示す。

[レプリカの決定手順]

- (1) クライアント c は、レプリカ集合 $r(o)$ 内の各レプリカ o_j に、サービス品質 $QoS(c, o_j)$ を問い合わせる。
- (2) 各レプリカ o_j は、自分のサービス品質 $QoS(c, o_j)$ を、クライアント c に通知する。
- (3) クライアント c は、 $r(o)$ 内で、利用条件を満たすレプリカの中で、最大のサービス品質を提供するレプリカ o_i を求める (即ち、 $QoS(c, o_i) \geq ReQc, o$) かつ、任意のレプリカ o_j に対して、 $QoS(c, o_i) \geq QoS(c, o_j)$)。

以上の手順で決定されたレプリカ o_i に対して、検索要求を行う。クライアント c が、オブジェクト o の全てのレプリカに問い合わせを行うことが、通信時間等のサービス品質の点から難しい場合がある。この場合には、以下のようなヒューリスティクスを用いる。

- (1) クライアント c と同一のセル内にレプリカがあるかを調べる。同一のセル内にレプリカがあれば、これらに問い合わせを行う。この中で最も高いサービス品質を提供し、かつレプリカ利用条件を満足しているレプリカ o_i を選ぶ。
- (2) (1)でレプリカを選択できなかった場合には、クライアント c の近傍のセル内のレプリカについて、問い合わせを行う。
- (3) (2)でも、レプリカを選択できない場合には、オブジェクトの全レプリカに問い合わせを行う。

3.3 レプリカ変更手順

クライアント c は、オブジェクト o のレプリカ o_i を以上の手順により選択し、利用しているとする。クライアント c またはデータベースサーバの移動により、レプリカ o_i のサーバ品質 $QoS(c, o_i)$ も変化する。クライアント c は、レプリカ o_i を利用している間に、 o_i のサービス品質について以下が起こり得る。

[レプリカ変更条件]

- (1) レプリカ o_i のサービス品質 $QoS(c, o_i)$ が低下して、利用条件を満足しなくなる。即ち、 $QoS(c, o_i) < ReQ(c, o)$ 。
- (2) レプリカ o_i のサービス品質よりも高いサービス品質を提供するレプリカ o_j がある。即ち、 $QoS(c, o_i) < QoS(c, o_j)$ 。

以上のレプリカ変更条件が満足されるときには、クライアント c は、利用するレプリカを o_i から、 o_j に変更する。レプリカ変更条件(1)は、クライアント c が、現在利用しているレプリカ o_i のサービス品質 $QoS(c, o_i)$ と要求サービス品質 $ReQ(c, o)$ を比較することにより調べられる。一方、レプリカ変更条件(2)は、以下の方法により調べることができる。

- (1) クライアント c は、定期的に、オブジェクト o のレプリカ集合 $r(o)$ に、サービス品質についての問い合わせを行う。
- (2) $r(o)$ 内の各レプリカ o_k は、定期的に、自分のサービス品質 $QoS(c, o_k)$ をクライアント c に通知する。
- (3) レプリカ集合 $r(o)$ は、一つのマスターレプリカ o_s を持つ。マスターレプリカ o_s が、 $r(o)$ 内のレプリカのサービス品質をモニタしている。クライアント c は、 o_s に問い合わせることにより、 $r(o)$ 内のレプリカのサービス品質を得れる。

以上の手順により得られたサービス品質により、 $QoS(c, o_i) < QoS(c, o_j)$ なるレプリカ o_j を見つける。

[レプリカ変更手順]

- (1) クライアント c は、レプリカ o_i を利用しているとする。レプリカ o_i の変更条件が満足されるとする。
- (2) クライアント c は、オブジェクト o のレプリカのサービス品質情報を獲得して、レプリカ o_i にかわるレプリカ o_j を選択する。 o_j が見つければ、クライアント c は、 o_j を利用する。

4章 トランザクション管理方式

複数のモバイルオブジェクトを更新するトランザクションの管理方式について考える。

4.1 概要

モバイルデータベースシステムで、データをどのように更新するかについて考える。モバイルデータベースシステムでは、各ユーザが持つ移動型ステーションが長時間の通信を行うために十分な容量のバッテリーを持てない。このために、移動型ステーションでは、サーバとの通信をなるべく行わないようにする必要がある。移動型ステーション内のクライアントは、サーバとの通信コネクションを切断しても、サーバ内のデータを利用した応用プログラムを動作させる必要がある。

[定義] クライアントとサーバ間のコネクションが切断されていて、クライアントの応用から発行された演算を、切断演算 (disconnected operation) という。

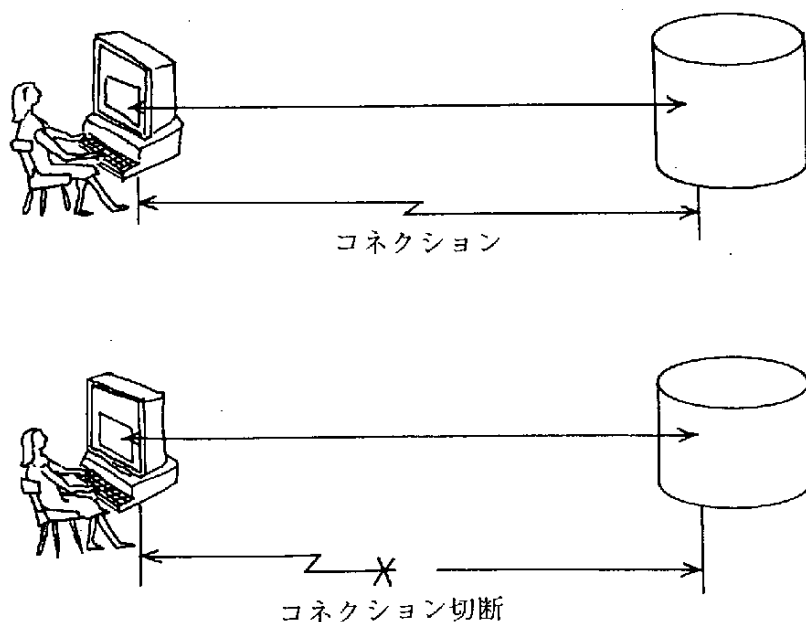


図4.1 切断演算

モバイルデータベースシステムでは、切断演算を、コネクションが切断されていても動作させることが必要となる。

切断演算を動作させる方法として、以下のキャッシング (casing) 方法がある。

[キャッシング] 切断演算 op が操作するサーバ内のデータを、クライアントにコピーする。

クライアントは、このようにキャッシングされたデータを操作することにより、コネクションの切断を意識せずにデータを利用できる。図4.2では、クライアント C のアプリケーション A が、サーバ S 内のテーブル Emp を利用している場合を示している。クライアント C とサーバ S 間の通信コネクションが切断するとする。このとき、サーバ S 内のテーブル Emp をクライアント C に転送する。コネクションが切断されても、アプリケーション A は、クライアント C 内に転送されたテーブル Emp を操作できる。これが、キャッシングである。

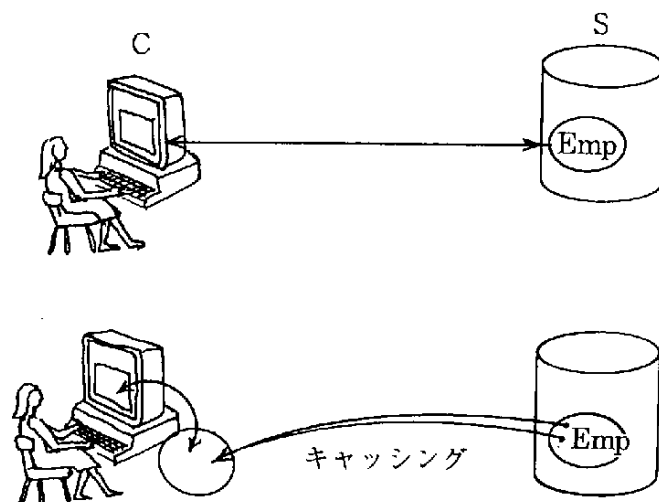


図4.2 キャッシング

以上のキャッシングにより、データをクライアントに転送することにより、クライアントは、ローカルにデータを利用できる。しかし、データが更新される場合には、サーバ内のデータと、クライアントにキャッシングされたデータ間での一貫性 (consistency) を保つことが必要となる。

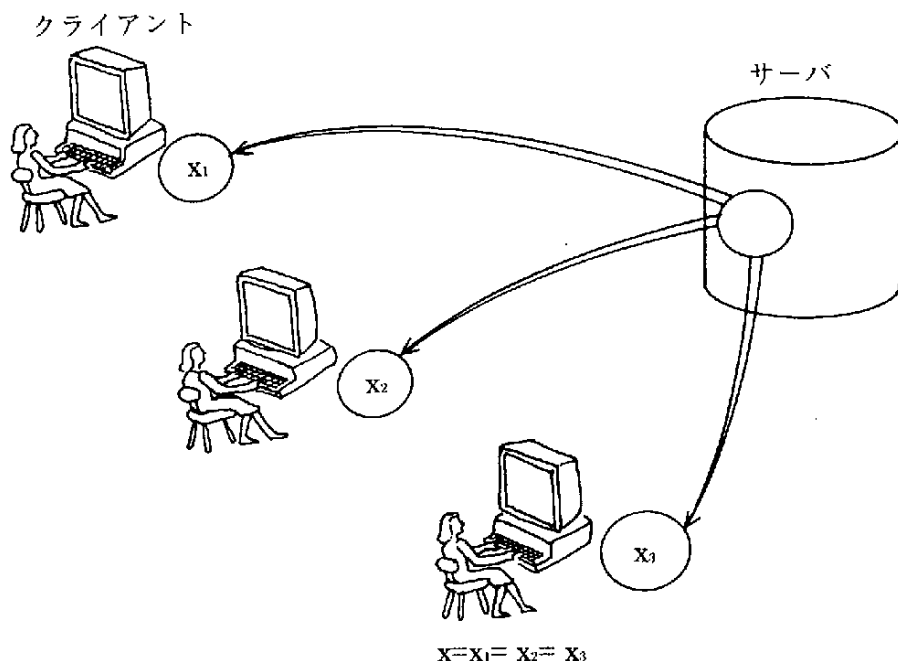


図4.3 レプリカ間の一貫性

〔キャッシングの問題点〕サーバ内のデータと、キャッシングされたデータ間の一貫性の保持を行う必要がある。

ここで、クライアントにキャッシングされたデータを、サーバ内のデータのレプリカ (replica) と呼ぶことにする。サーバ S 内のデータ x を複数のクライアント $C_1, \dots, C_m$ ($m \geq 1$) が利用する。各クライアント C_i 内の x のレプリカを x_i とする。データ x とレプリカ $x_1, \dots, x_m$ 間の一貫性をどのように保つかが問題となる。レプリカ間の一貫性を保つ方法として、これまでに、いくつかの方法が提案されてきている。

- (1) 集中方式。
- (2) 分散方式。

集中方式では、検索は、各クライアント C_i のレプリカ x_i に対して行われる。しかし、更新はすべてのレプリカをロックしてから、全レプリカの更新を行う。

モバイルデータベースシステムでは、各クライアント C_i のコネクションは切断されているために、他のクライアントとサーバ内のレプリカをロックできない。また、サーバ内のデータ x をロックしてから、コネクションを切断すると、コネクションが再開されるまで、どのクライアントもデータ x を利用できなくなる。従って、以下の条件が満足される場合を考える。

[仮定] 複数のクライアントが、サーバ内の同一データを更新することは稀である。

以上の仮定が成り立たない場合には、従来のレプリカの更新方式を行う必要がある。以上の仮定が成り立つもとで、以下の楽観的な方法を考える。

[楽観的方法]

- (1) 各クライアント C_i で、コネクションが切断されている場合には、レプリカ x_i の更新を行う。
- (2) コネクションが再開されたとき、サーバ S 内のデータ x がどのクライアントによっても更新されていない場合には、サーバ S 内の x も更新する。
- (3) サーバ S は、 x が更新されたとき、コネクションが開設されているクライアント C_j に x の更新を通知する。または、 C_j が x の更新状況をサーバ S に問い合わせる。

4.2 トランザクションの定義

モバイルデータベースシステムでの処理の単位をトランザクション (transaction) という。ここでは、トランザクションとは何かについて考える。利用者が、複数のデータベースサーバ $DBS_1, \dots, DBS_n$ 内のデータを操作する問題を考える。ここで、各 DBS_i 内

のデータベース DB_i をオブジェクトの集合とする。オブジェクトとは、各 DB_i の基本的な操作単位である。リレーショナルデータベースシステムでは、テーブル、断片、行がオブジェクトである。何をオブジェクトとするかは、各サーバにより決まる。こうしたオブジェクトの単位を粒度 (granularity) という。

4.2.1 データベースの演算

利用者のプログラムは、クライアントで実行される。プログラム内のデータ操作演算が実行されると、クライアントはデータベースサーバにデータ操作演算 (SQL) を送信する。ここでは、 DB_i の演算を抽象化して考える。 DB_i 内のオブジェクト x に対する基本操作としては、以下の演算がある。

[基本操作演算]

- (1) $r_i[x]$ オブジェクト x の値の読み出す (read) 演算。
- (2) $w_i[x]$ オブジェクト x に値を書き込む (write) 演算。

$r_i[x]$ は x に対する SQL の select 文に対応し、 $w_i[x]$ は update、insert、delete 文に対応する。クライアントは、これらの操作演算をデータベースサーバに対して送信する。

4.2.2 トランザクション

次に、銀行 A から B に送金することを考える [図4.4]。送金を行う場合には、A の口座と B の口座を示す二つのオブジェクトの更新を行う必要がある。両方の更新を正しく行えたときのみ、送金処理を完了できる。いずれか一方の更新を行えないときには、送金を行えない。このような複数の操作演算の実行系列をトランザクションという。以上の基本操作演算に加えて、トランザクションの終了を行うための演算として以下のものがある。

- (1) $c(\text{commit})$ トランザクションを正常終了 (コミット) させる演算である。
- (2) $a(\text{abort})$ トランザクションを異常終了 (アボート) させる演算である。

アボート a は、これまでに実行された演算を無効とする演算である。これに対して、コミット c は、これまでに実行された演算を完了する演算である。

例えば、A の口座の更新後に、B の口座の更新を行えないことがわかったとき、アボート a を実行することにより、A の口座の更新を無効とできる。一方、A と B の口座の更新を行えたときに、コミットメント c を行くと、データベース内の A と B の口座のオブジェクトが変更され、完了する。c と a は、おのこの SQL の `commit_work` と `abort_work` 文の実行を示す。

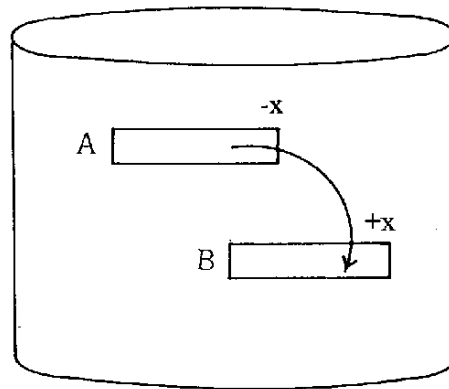


図4.4 トランザクション

以上をまとめると、トランザクション T とは、複数のデータベースサーバ内のオブジェクトに対する一つ以上の基本演算の系列の実行状態である。但し、以下の性質を充足するものである。

[トランザクションの性質 (ACID)]

- (1) 原子性 (atomicity) : T 内の全操作演算が実行されるか、全くされないかのいずれかである。
- (2) 一貫性 (consistency) : T を単独で実行させたときに、データベースのインテグリティ制約を保つ。
- (3) 独立性 (isolation) : T の中間結果を他のトランザクションは見れない。
- (4) 耐久性 (durability) : T が正常終了した時、更新結果はデータベースから消失しな

い。

[例] トランザクションの例を考える。3つのデータベースサーバ DBS_1 、 DBS_2 、 DBS_3 があるとする。各 DBS_i は、各個人の名前 (name) とその人の口座の残金 (account) を示すテーブル $Bank(name, account)$ を持つ。ここで、ある個人 A が、 DBS_1 と DBS_2 内の自分の口座から、おのおの x 円と y 円を引き出して、その合計額 ($= x + y$) を DBS_3 内の B 氏の口座に振り込むとする。これは、SQL により以下のように書ける。以下で、 $DBS_i.Bank$ は、 DBS_i 内のテーブル $Bank$ を示す。

```
select account into t1 from DBS1.Bank
    where name = "A";
if t1.account - x < 0, then abort_work;          (1)
update DBS1.Bank set account = account - x
    where name = "A";
select account into t2 from DBS2.Bank
    where name = "A";
if t2.account - y < 0, then abort_work;          (2)
update DBS2.Bank set account = account - y
    where name = "A";
update DBS3.Bank set account = account + x + y
    where name = "B";
commit_work;                                     (3)
```

ここで、`abort_work` はトランザクションのアボート (異常終了) を行わせる演算であり、これまでの更新結果を無効とする。例えば、この例では、(2) で、 DBS_1 の $Bank$ を更新した後に、 DBS_2 の $Bank$ が残高不足のために更新できなかった場合に、 DBS_1 の更新を無効としている。`commit_work` はトランザクションのコミット (正常終了) を示す。正常終了すると更新が完了する。このプログラムはクライアント C で実行され、データベースに対する演算がデータベースサーバに送信される。ここで、データベースサーバに

送信される演算の系列がトランザクションTである。正常終了した場合のトランザクションTを以下に示す。

T: $r_1[a_1]; w_1[a_1]; r_2[a_2]; w_2[a_2]; r_3[b]; w_3[b]; c;$

[原子性]

ここで、 a_1 と a_2 は、おのおの DBS_1 と DBS_2 内のAについてのオブジェクト（行）を示す。また、 b は DBS_3 内のBについてのオブジェクトを示す。すべての操作演算を正しく行えるときにだけ、Tは正常終了する。もし、どれかの演算を実行できない場合には、ここまでに実行された全演算が無効となる。これが原子性である。

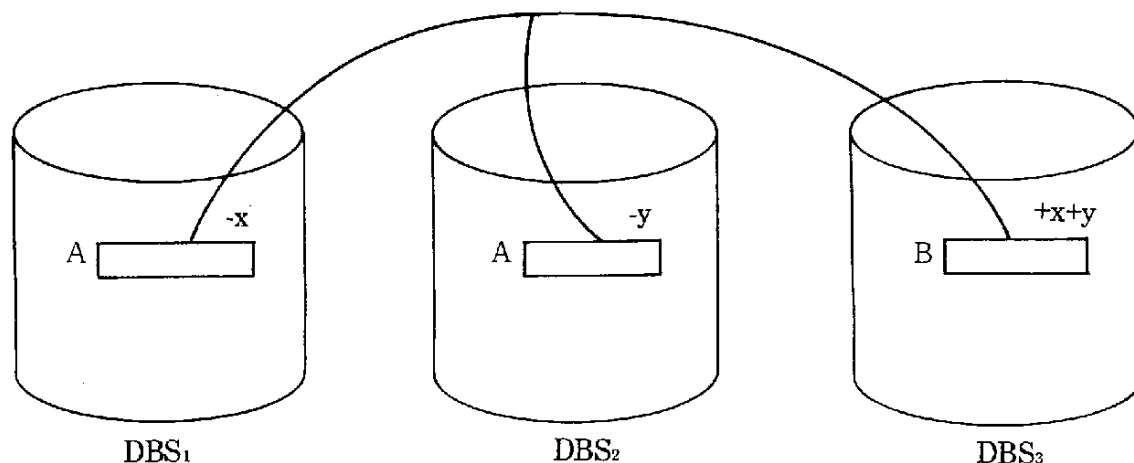


図4.5 トランザクション

[耐久性]

Tが正常終了したときには、 DBS_1 、 DBS_2 、 DBS_3 のBank内の値は更新される。ここで、データベースサーバが故障を起こした後に復旧したとする。このとき、Bank内のaccountの値は更新された値でなければならない。この値を変更するためには、変更を行うためのトランザクションを実行する必要がある。これが耐久性である。

[独立性]

さらに、 DBS_1 と DBS_2 内のオブジェクトの更新後で、 B の更新を行う前に、他のトランザクションが A の口座を操作することができないことが、独立性である。即ち、トランザクションの中間結果を、他のトランザクションが見れないことである。

[一貫性]

T を単独で実行した場合、 A の口座の値を B に移しているので、データベース内の残金の合計は変わらない、即ちインテグリティ制約が保たれている。これが、一貫性である。

[副トランザクション]

トランザクション T は、クライアント C で実行される。 T 内の各演算は、 C からサーバ DBS_1 、 DBS_2 、 DBS_3 に送信され、各サーバで実行される。各 DBS_i で実行される T の演算の系列を副トランザクション T_i とする。例えば、図5.19で、 T の副トランザクション $T_1 = r_1[a1]; w_1[a1]$ 、 $T_2 = r_2[a2]; w_2[a2]$ 、 $T_3 = r_3[b]; w_3[b]$ である。

[プロセスとトランザクション]

クライアントでのプログラムの実行状態はプロセスである。トランザクションはプロセスの開始からコミット (c) またはアボート (a) までか、 c または a が実行されてから次の c または a までがトランザクションである。図4.6にプロセスを示す。ここで、演算 $op_i \dots op_h$ が実行され、コミットしている。ここでトランザクションが終了し、次のトランザクション $op_{h+1} \dots op_i$ が実行されるが、アボートする。 $op_{h+1} \dots op_i$ の実行は無効とされる。最後にトランザクション $op_{i+1} \dots op_j c$ が実行される。このように、一般に、プロセスは、一つ以上のトランザクションから構成される。

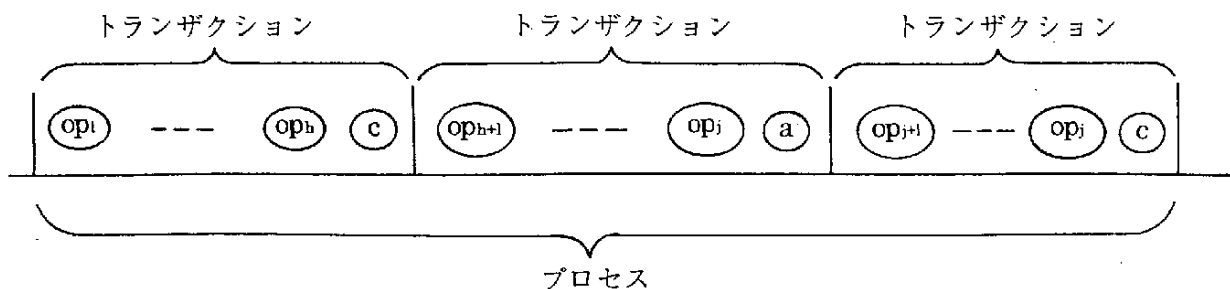


図4.6 プロセスとトランザクション

4.3 実行履歴

$m(\geq 1)$ 個のトランザクション $T_1, \dots, T_m$ が、 $n(\geq 1)$ 個のデータベースサーバ $DBS_1, \dots, DBS_n$ 内のオブジェクトを操作する場合について考える。ここで、 $op_i[x]$ は、 T_i の演算で、 DBS_i 内のオブジェクト x に対する演算を示すとする。

4.3.1 履歴

各 DBS_i は、複数のトランザクションからの演算を一つずつ直列に実行する。この演算実行の履歴を、 DBS_i のローカル履歴 (local history) L_i とする。ローカル履歴 $L_1, \dots, L_m$ の集合を、 $T_1, \dots, T_m$ の履歴 L とする。

[例] DBS_1 がオブジェクト x と y を持ち、 DBS_2 が z を持つとする [図4.7]。次の二つのトランザクション T_1 と T_2 が DBS_1 と DBS_2 に対して実行されるとする。

$T_1: r_{11}[x]; w_{12}[z]; r_{11}[y]; c$

$T_2: r_{22}[z]; w_{21}[x]; r_{21}[y]; c$

DBS_1 と DBS_2 に対するローカル履歴 L_1 と L_2 の例を以下に示す。

$L_1: r_{11}[x]; w_{21}[x]; r_{11}[y]; r_{21}[y];$

$L_2: r_{22}[z]; w_{12}[z];$

これらは、 DBS_1 と DBS_2 で、 T_1 と T_2 をインタリーブ実行した履歴である。

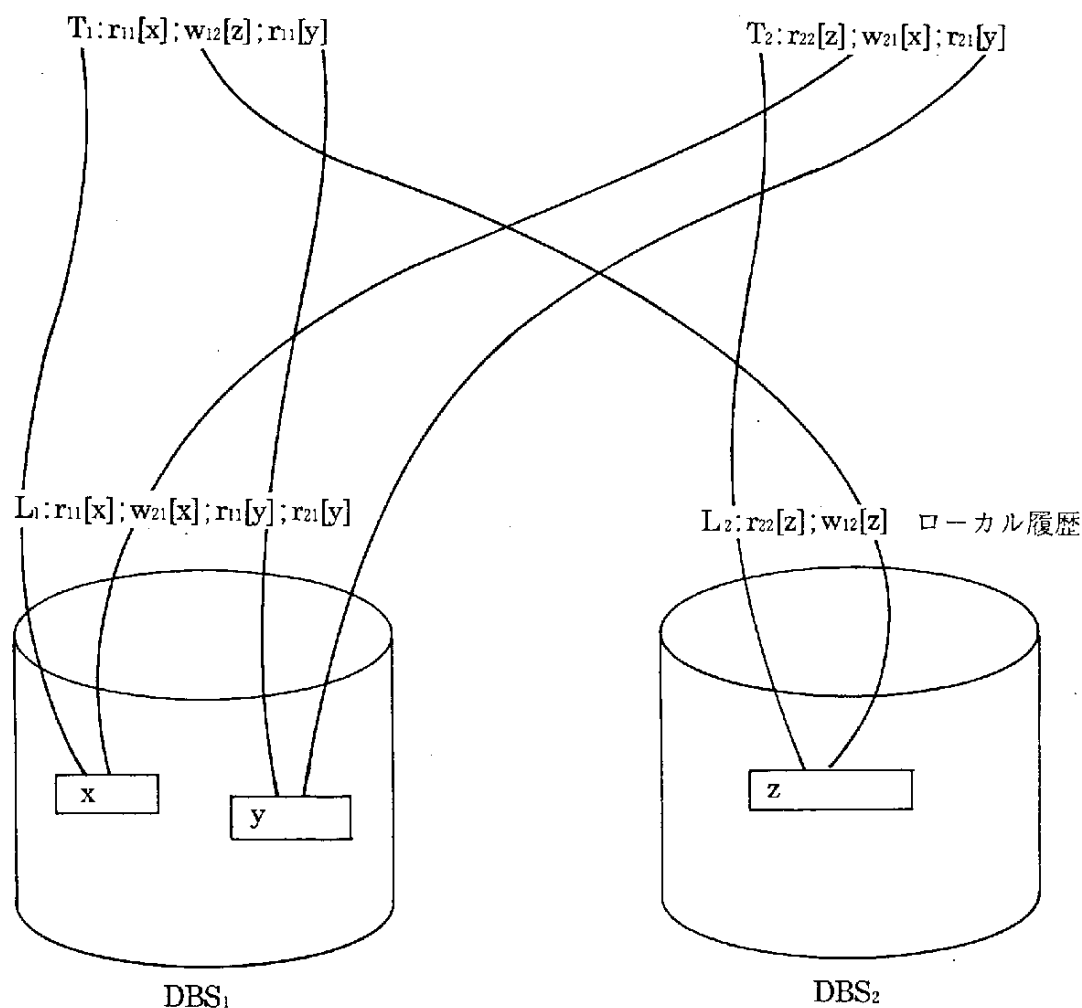


図4.7 ローカル履歴

複数のトランザクション $T_1, \dots, T_m$ の履歴 L の性質について考える。複数のデータベースシステム内の各データベースサーバのデータベースの状態が正しいとき、分散データベースは正しいとする。

4.3.2 正しい履歴

正しい分散データベースに対して、履歴 L を実行させた結果、正しい分散データベースをもたらすならば、履歴 L は正しいとする。

4.4 独立性

トランザクションが独立性を満足することから、トランザクション T の実行中に、 T が `write` を行った結果を他のトランザクションが見ることはできない。独立性について考える。

4.4.1 読出し関係

ここで、 x をデータベースサーバ DBS_i 内のオブジェクトとする。また、 $r_j[x]$ 、 $w_j[x]$ 、 a_j 、 c_j を各々トランザクション T_j の x に対する `read`、`write`、`abort`、`commit` 演算とする。

[トランザクション間の読みだし関係]

L をトランザクション $T_1, \dots, T_m$ の履歴とする。 x を DBS_h 内のオブジェクトとする。 T_j が T_k からオブジェクト x を読むとは、以下の条件を満足することである[図4.8]。

- (1) DBS_h で、 $w_k[x]$ が $r_j[x]$ に先行して実行されている。
- (2) $r_j[x]$ の前にアボート a_k は実行されていない。
- (3) $w_k[x]$ より後で $r_j[x]$ より前に $w_k[x]$ が実行されているが、 $j[x]$ の実行前に T_k はアボートしている。

即ち、 T_k が書いた x の値を、 T_j が読んでいるとき、 T_j は T_k から x を読むという。

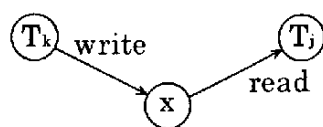


図4.8 読みだし関係

4.4.2 復旧可能履歴

次に履歴 L の性質について考える。履歴 L 内で、トランザクション T_j が T_k からオブジェクト x を読んでいる場合を考える。ここで、 T_k がコミットしている、即ち、 c_k を L が含んでいるとする。このとき、 T_k は T_j より先にコミットしているとき、 L は復旧可能

(recoverable) という。そうでないとき、 L は復旧不可能であるという。

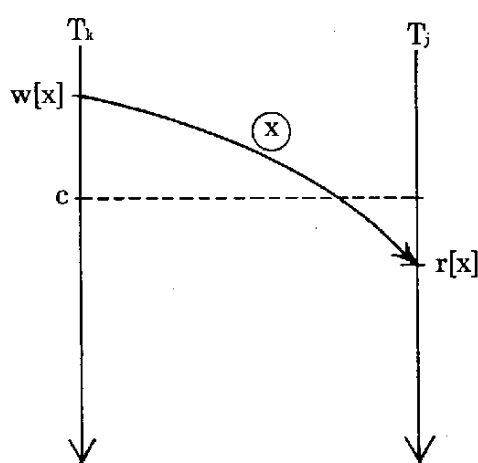
以下の3つのログについて、例を示す。

L_1 : ... $w_k[x]$... c_k ... $r_j[x]$... c_j ...

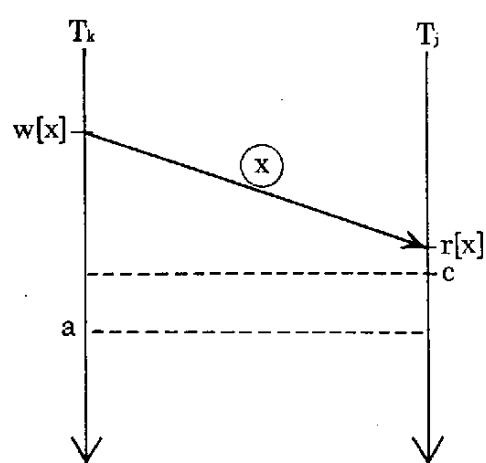
L_2 : ... $w_k[x]$... $r_j[x]$... c_j ... a_k ...

L_3 : ... $w_k[x]$... $r_j[x]$... c_k ... c_j ...

L_1 では、トランザクション T_j は、 T_k が書いた値を、コミット後に読んでいますので、復旧可能な履歴である [図4.9(1)]。これに対して、 L_2 と L_3 では、 T_j は、 T_k がコミットする前に T_k が書いた値を読んでいて、 L_2 では、 T_j がコミットした後に、 T_k がアボートしている。 T_j はすでにコミットしているので、アボートできない。結果として、 T_j は存在しない値を読んだことになってしまう。従って、このような履歴では、トランザクションのアボートに対して、復旧を行えず、復旧不可能という [図4.9(2)]。



(1) 復旧可能



(2) 復旧不可能

図4.9 復旧可能性

[連鎖的アボート]

L_3 は復旧可能であるが、 T_j は T_k の書いた値を読んでいて、したがって、 T_k がアボートすると、 T_j もアボートする必要がある。一つのトランザクションのアボートにより、

他のトランザクションもアボートすることを連鎖的アボートという。

[連鎖的なアボートを起こさない履歴]

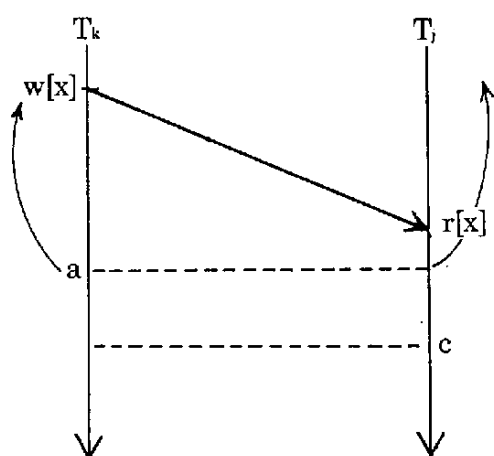
履歴 L で、トランザクション T_i が T_k から x を読むとする。このとき、 T_k が $r_j[x]$ 前にコミットしているとき、 L は連鎖的なアボート (cascading abort) を起こさないという。

以下の例について考える。

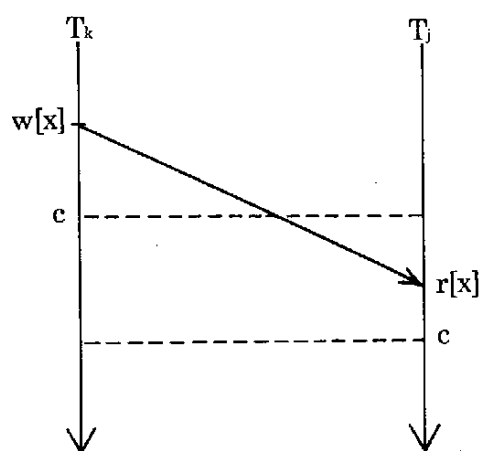
L_4 : ... $w_k[x]$... $r_j[x]$... a_k ... c_j ...

L_5 : ... $w_k[x]$... c_k ... $r_j[x]$... c_j ...

履歴 L_4 では、 $r_j[x]$ の後に T_k がアボートしたならば、 T_k もアボートしなければならない。したがって、連鎖的なアボートを起こす履歴である。 L_5 は、連鎖的なアボートを起こさない。また、 L_5 は、復旧可能である。



(1) 連鎖的アボート



(2) 非連鎖的アボート

図4.10 連鎖的アボート

[厳格性]

ここまでは、トランザクション間での w と r の関係を考えてきた。次に、トランザクション間での w 演算の間の関係について考える。

[厳格な履歴]

$op_j[x]$ を、トランザクション T_j の $r_j[x]$ 又は $w_j[x]$ とする。 $w_k[x]$ が $op_j[x]$ に先行し、 T_k が $op_j[x]$ の前に終了(コミットまたはアボート)するとならば、履歴 L は厳格 (strict) であるという。

以下の例で考える [図4.11]。

L_6 : ... $w_j[x]$... $w_k[x]$... c_k ... $r_j[x]$... c_j ...

L_7 : ... $w_k[x]$... c_k ... $w_j[x]$... $r_j[x]$... c_j ...

L_6 は、 T_j がコミットする前に x を書いているので、厳格な履歴ではない。 L_7 は厳格な履歴である。また、 L_7 は連鎖的アボートを起こさないこともわかる。

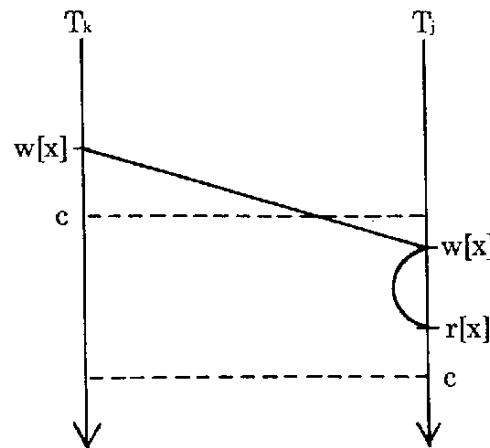


図4.11 厳格履歴

以上から、厳格な履歴は連鎖的アボートを起こさず、連鎖的アボートを起こさない履歴はない履歴は復旧可能であることがわかる。しかし、この逆は、成り立たない。厳格な履歴は連鎖的アボートを起こさない履歴よりも制約が厳しく、連鎖的アボートを起こさない履歴は復旧可能な履歴よりも制約が厳しいが、トランザクションの並行度は減少し、分散処理システムの性能が低下してしまう。

4.5 競合演算

$op_i[x]$ をトランザクション T_i の演算で、 DBS_j 内のオブジェクト x に対する演算とする。 op は r (read) または w (write) である。 x に対する $op_i[x]$ と $op_k[x]$ を考える。

[競合演算] 二つの演算の実行順序により演算結果が異なる時、 $op_i[x]$ と $op_k[x]$ は競合するという。

例えば、 $r_i[x]$ の次に $w_k[x]$ を実行した場合と、 $w_k[x]$ の後に $r_i[x]$ を実行した場合では、 x の値は同じでも、 $r_i[x]$ する値が異なる。したがって、 $r_i[x]$ と $w_k[x]$ は競合する。同様に、 $w_k[x]$ と $w_h[x]$ も競合する。競合する演算の実行順序を、各サーバでどのような順序とするかが重要な問題となる。



図4.12 トランザクション

[汚い読出し]

各データベースサーバで、複数のトランザクションからの演算を任意の順序で実行すると、分散データベースを正しくない状態とする場合がある。まず、例で考える。

[例] 二つのデータベースサーバ DBS_1 と DBS_2 が、おのこのオブジェクト a と b を持つとする。インテグリティ制約として、" $a = b$ " という関係があるとする。 T_1 は a と b の値に 1 加え、 T_2 は a と b の値を 2 倍にするトランザクションとする。 T_1 を T_2 を以下に示す[図4.13]。

T_1 : $r_{11}[a]; /* a=a+1 */ w_{11}[a]; r_{12}[b]; /* b=b+1 */ w_{12}[b]; c;$

T_2 : $r_{21}[a]; /* a=a*2 */ w_{21}[a]; r_{22}[b]; /* b=b*2 */ w_{22}[b]; c;$

明らかに、 T_1 と T_2 を、おのこの単独で実行した場合には、 $a=b$ である。しかし、 DBS_1 と DBS_2 のローカル履歴 L_1 と L_2 [図4.13] を考えると、 $a \neq b$ となる。これは、 T_2 が T_1 の中間結果を読むことによりおきるので、汚い読みだし (dirty read) という。従って、 L_1 と L_2 から成る履歴 L は、正しくない。

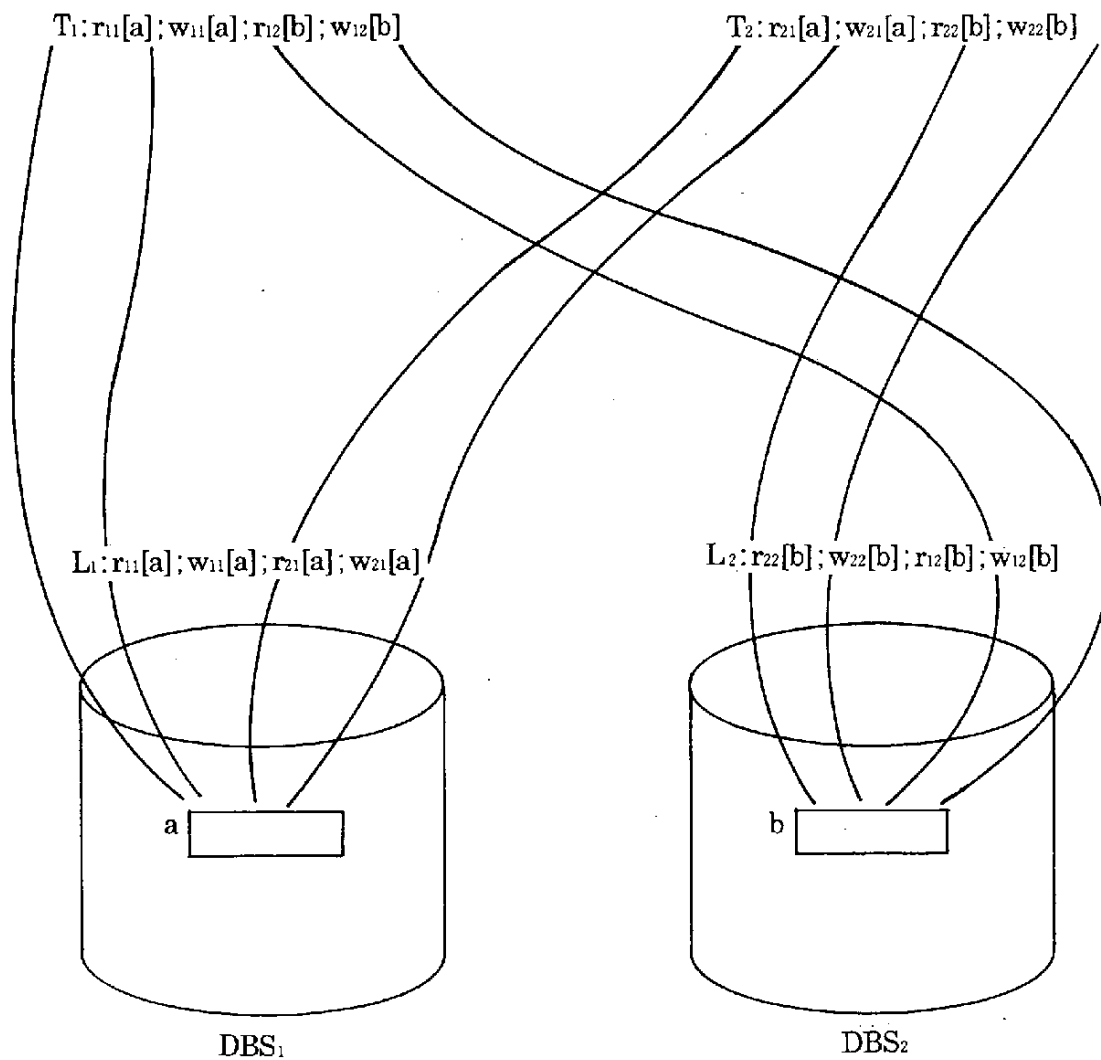


図4.13 汚い読出し

[紛失更新]

[例] 次に、図4.13に示す二つのオブジェクトaとbに対する以下の二つのトランザクション T_1 と T_2 を考える。

$T_1: w_{11}[a]; /* a = 20 */ w_{12}[b]; /* b = 20 */ c;$

$T_2: w_{21}[a]; /* a = 30 */ w_{22}[b]; /* b = 30 */ c;$

このローカル履歴の例 L_1 と L_2 を図4.14に示す。この結果、 $a \neq b$ となってしまふ。ここでは、競合する write 演算の実行順序が問題である。これを、紛失更新 (lost update) という。

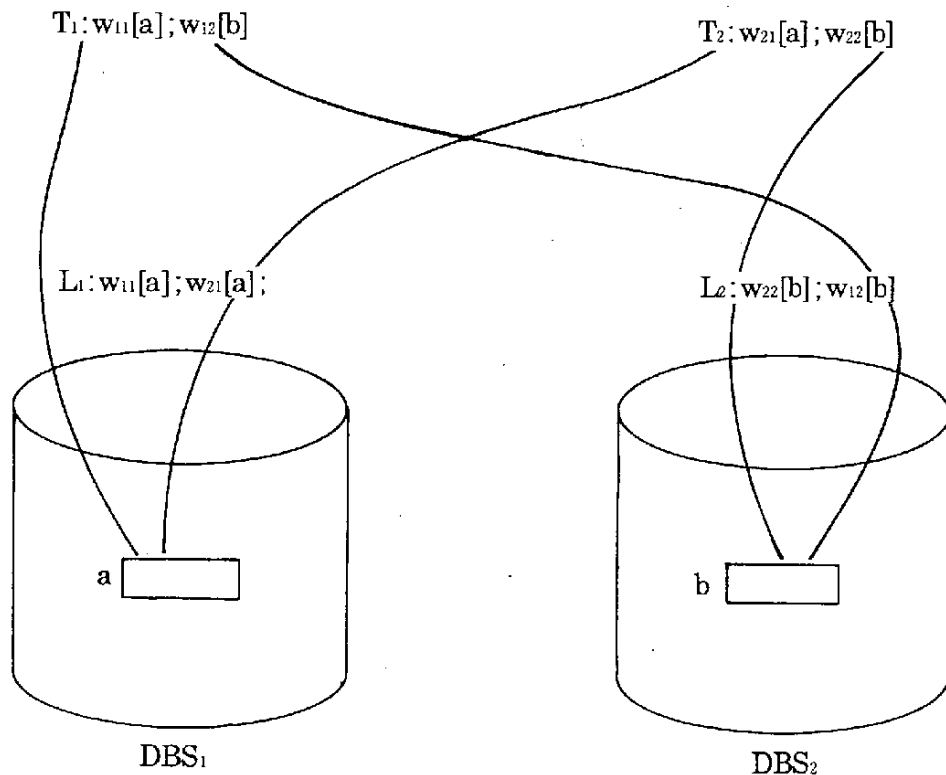


図4.14 紛失更新

以上の例からわかるように、分散データベースを正しい状態に保つためには、複数のトランザクションの競合する演算の実行順序を制御する必要がある。このための制御を、同時実行制御という。

4.6 同時実行制御 (concurrency control)

複数のトランザクションが、複数のデータベースサーバ内のオブジェクトを操作する問題について考える。このとき、各データベースサーバ内のデータベースのインテグリティ制約を保つとともに、システム全体のスループットを向上させる必要がある。このための制御が、同時実行制御 [BERN87] である。

4.6.1 直列可能性

[並列実行の正さ]

トランザクション $T_1, \dots, T_m$ が、データベースサーバ $DBS_1, \dots, DBS_n$ 内のオブジェクトを更新する場合を考える。複数のトランザクションが複数のサーバにより並列実行されることの正しさの根拠として直列可能性 (serializability) [BERN87, LYNC93] がある。あるトランザクションが完了してから次のトランザクションが一つずつ実行されることを直列実行とし、その履歴を直列とする。トランザクションは一貫性を満足していることから、トランザクションを単独で実行したときには、分散データベースシステム内のデータベースを正しい状態に保つ。このことから、複数のトランザクションが直列に実行された場合も、データベースを正しい状態に保つ。複数のデータベースサーバで、並列実行された結果が、ある直列実行した結果と同じであることをもって、この並列実行の正しさとしている。

L をトランザクション $T_1, \dots, T_m$ の履歴とする。このとき、 L に対する $T_1, \dots, T_m$ 間の順序関係 $<_{<L}$ を以下に定義する。

[トランザクション間の順序関係]

二つのトランザクション T_j と T_k に対して、以下が成り立つとき、 $T_j <_{<L} T_k$ とする。即ち、あるローカル履歴 L_h で、

- (1) T_j の演算 op_{jh} と T_k の演算 op_{kh} を含み、 op_{jh} と op_{kh} が競合し、
- (2) op_{jh} が op_{kh} が DBS_h で先行して実行される。

例えば、図4.13で、DBS₁内のオブジェクトaについては、T₁がT₂よりも先に操作しているので、T₁ <<_L T₂である。一方、DBS₂では、T₂がT₁よりも先にbを操作しているので、T₂ <<_L T₁である。

順序関係 <<_L を用いて、T₁, ..., T_n の履歴 L が直列可能であることを定義する。

[直列可能性]

履歴 L について、<<_L が全順序関係ならば、L は直列可能である。

[例] 二つのサーバ DBS₁ と DBS₂ を考える。ここで、DBS₁ がオブジェクト x と y を持ち、DBS₂ が z を持つとする [図4.15]。DBS₁ と DBS₂ に対する次の二つのトランザクション T₁ と T₂ を考える。

T₁: r₁₁[x]; w₁₂[z]; r₁₁[y]; c

T₂: r₂₂[z]; w₂₁[x]; r₂₁[y]; c

DBS₁ と DBS₂ に対するローカル履歴 L₁ と L₂ の例を以下に示す。

L₁: r₁₁[x]; w₂₁[x]; r₁₁[y]; r₂₁[y];

L₂: r₂₂[z]; w₁₂[z];

L₁ と L₂ から構成される履歴 L は、直列可能ではない。何故ならば、DBS₁ で、x の競合演算は、T₁ の r₁₁[x] が T₂ の w₂₁[x] より前に実行されているが、DBS₂ では逆に T₂ の r₂₂[z] が T₁ の w₁₂[z] に先行しているからである。即ち、T₁ <<_L T₂ で T₂ <<_L T₁ であり、<<_L は全順序関係でない。

しかし、以下に示すローカル履歴 M₁ と M₂ の履歴 M は直列可能である。DBS₁ の x については、T₁ の r₁₁[x] が T₂ の w₂₁[x] に先行している。DBS₂ の z については、T₁ の w₁₂[z] が T₂ の r₂₂[z] に先行している。y については、T₂ の r₂₁[y] が T₁ の r₁₁[y] に先行しているが、これらは競合しない。従って、DBS₁ と DBS₂ における競合演算は、T₁ が T₂ よりも先行している (T₁ <<_M T₂)。このことは、T₁ を実行した後に、T₂ を直列に実行し

た場合と同じ結果が得られる。

$M_1: r_{11}[x]; w_{21}[x]; r_{21}[y]; r_{11}[y];$

$M_2: w_{12}[z]; r_{22}[z];$

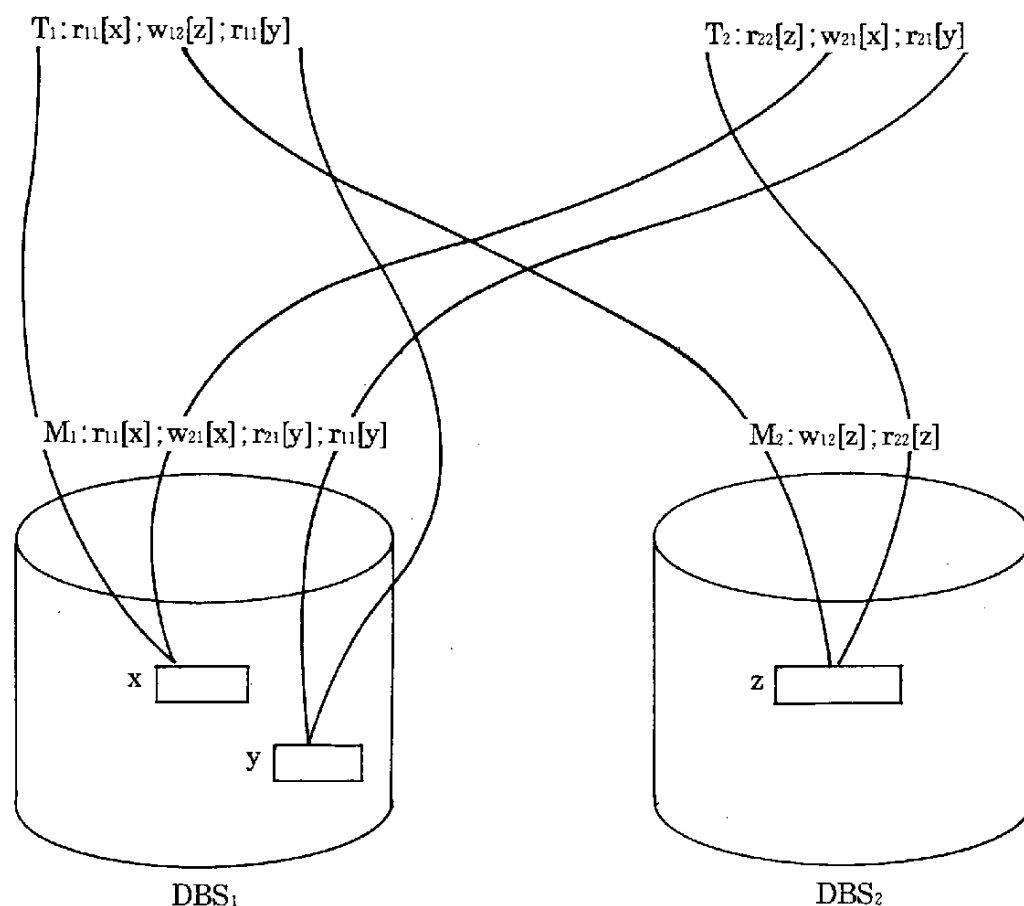


図4.15 直列可能性

直列可能性の判定のために、履歴 L に対する以下の順序グラフを用いる。

[順序グラフ]

履歴 L に対して、以下の手順により得られる有向グラフ G を L の順序グラフ (precedence graph) とする。

(1) L 内の各トランザクション T_i に対して、節点 T_i を設ける。

(2) 任意のトランザクション T_i と T_j に対して、 T_i の演算 op_i と T_j の演算 op_j が競合し、 L 内で op_i が op_j より先行して実行されているとき、節点 T_i から T_j に有向辺 $T_i \rightarrow T_j$ を設ける。□

例えば、図4.15で考えた履歴 L と M の順序グラフを図4.16に示す。



図4.16 順序グラフ

順序グラフを用いると、二つの履歴が同値であることを、以下のように定義できる。

[履歴の同値性]

二つの履歴 L_1 と L_2 の順序グラフが同じとき、 L_1 と L_2 は同値である。例として、図4.15で、 T_1 の後に T_2 が実行された直列ログ S は、以下のようになる。

$S_1: r_{11}[x]; r_{11}[y]; w_{21}[x]; r_{21}[y];$

$S_2: w_{12}[z]; r_{22}[z];$

オブジェクト x について $T_1 \ll_s T_2$ であり、 z についても $T_1 \ll_s T_2$ である。よって、この順序グラフは、図4.16(2)と同じとなる。すなわち、履歴 M は、直列履歴 S と同値である。また、二つの履歴が同値であるとき、分散データベースを同じ状態とすることがわかる。

以上から、以下がいえる。

[直列可能性]

履歴 L の順序グラフ G が巡回閉路を含まないとき、 L は直列可能 (serializable) である。履歴が直列可能であれば、履歴は正しい。

4.6.2 二相ロック

複数のトランザクション $T_1, \dots, T_m$ の履歴を直列可能とする方法について考える。このためには、競合する演算の実行順序を、各データベースサーバで一定の順序とする必要がある。

A. 二相ロックプロトコル

履歴を直列可能とする代表的方法としては、二相ロック [GRAY78] がある。オブジェクト x に対して、新しくロック (lock) 演算 $l[x]$ とロックの解放 (unlock) 演算 $u[x]$ を導入する。

[ロックプロトコル]

- (1) 各トランザクション T_i は、オブジェクト x を操作する前に、 $l[x]$ により x のロックを行う。他の T_j が x をロックしている場合は、 T_i は T_j のロックの解放 $u[x]$ を待つ。
- (2) T_i は、終了前に、 T_i が獲得した全ロックの解放を行う。

トランザクションが、ロックプロトコルに従って、オブジェクトをロックしても、得られる履歴は、直列可能とはならない。

[例] 図4.13で考えたトランザクション T_1 と T_2 を考える。ここで、各トランザクションは、以下のように a と b をロックするとする。

T_1 : $l_{11}[a]; r_{11}[a]; w_{11}[a]; u_{11}[a]; l_{12}[b]; r_{12}[b]; w_{12}[b]; u_{12}[b]; c;$

T_2 : $l_{21}[a]; r_{21}[a]; l_{21}[a]; w_{21}[a]; r_{22}[b]; w_{22}[b]; u_{21}[a]; u_{22}[b]; c;$

この履歴 K を図4.17に示す。図からわかるように、ロックを行っても、図4.13と同じ履歴となり、汚い読出しが起き、直列可能ではない。

$K_1: l_{11}[a]; r_{11}[a]; w_{11}[a]; u_{11}[a]; l_{21}[a];$
 $r_{21}[a]; w_{21}[a]; u_{21}[a]$

$K_2: l_{21}[b]; r_{22}[b]; w_{22}[b]; u_{21}[b]; l_{12}[b];$
 $r_{12}[b]; w_{12}[b]; u_{12}[b]$

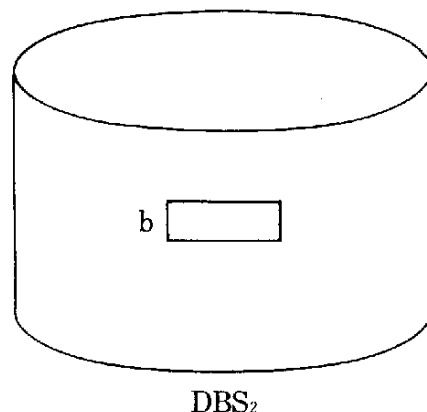
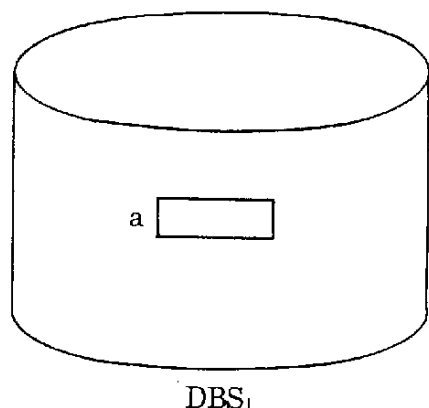


図4.17 ロック演算

履歴を直列可能するためのロック方法として、二相ロック (two-phase locking) 方式がある。これを以下に示す。まず、冗長なオブジェクトが存在しない場合を考える。T_iをトランザクションとする。

[二相ロックプロトコル]

- (1) T_i内の各演算 op_i[x] に対して、DBS_j内の x をロック (l_j[x]) する。
- (2) T_iがロックの解放演算を行ったならば、T_iはこれ以後ロックを行わない。

二相ロックトランザクションの例を以下に示す。

[例] 図4.15に示すデータベースサーバ DBS₁ と DBS₂ に対する以下の二つのトランザクション T₁ と T₂ を考える。

T₁: l₁₁[x]; r₁₁[x]; l₁₂[z]; w₁₂[z]; l₁₁[y]; u₁₂[z]; r₁₁[y]; u₁₁[y]; u₁₁[x]; c;

T₂: l₂₂[z]; r₂₂[z]; l₂₁[x]; w₂₁[x]; l₂₁[y]; r₂₁[y]; u₂₁[y]; u₂₁[x]; u₂₂[z]; c;

T₁ と T₂ では、ロックの解放演算が実行された後にロック演算が実行されていない。し

たがって、 T_1 と T_2 は、二相ロックトランザクションである。しかし、次のトランザクション T_3 は二相ロック形式ではない。 x のロックを $u_{11}[x]$ により解放した後に、 z と y のロックが行われているからである。

T_3 : $l_{11}[x]; r_{11}[x]; l_{12}[z]; u_{11}[x]; w_{12}[z]; l_{11}[y]; u_{12}[y]; r_{11}[y]; u_{11}[y]; c;$

DBS_j での T_i の操作演算の系列 T_{ij} は、 T_i の副トランザクションである。 T_1 と T_2 の副トランザクションは、以下のようになる。

T_{11} : $l_{11}[x]; r_{11}[x]; l_{11}[y]; r_{11}[y]; u_{11}[x]; u_{11}[y]; c;$

T_{12} : $l_{12}[z]; w_{12}[z]; u_{12}[z]; c;$

T_{21} : $l_{21}[x]; w_{21}[x]; l_{21}[y]; r_{21}[y]; u_{21}[y]; u_{21}[x]; c;$

T_{22} : $l_{22}[z]; r_{22}[z]; u_{22}[z]; c;$

各 T_{ij} は、二相ロック形式であることがわかる。副トランザクション T_{11} と T_{21} の DBS_1 でのインタリーブ実行を示すローカル履歴 N_1 と、 T_{12} と T_{22} の DBS_2 でのインタリーブ実行を示すローカル履歴 N_2 を以下に示す。

N_1 : $l_{11}[x]; r_{11}[x]; l_{11}[y]; u_{11}[x]; l_{21}[x]; u_{11}[y]; w_{21}[x]; l_{21}[y]; r_{21}[y]; u_{21}[x]; u_{21}[y];$

N_2 : $l_{12}[z]; w_{12}[z]; u_{12}[z]; l_{22}[z]; r_{22}[z]; u_{22}[z];$

副履歴 N_1 と N_2 から成る履歴 N では、どの競合演算も、 T_1 のものが T_2 よりも先に実行されている。即ち、 $T_1 \ll_N T_2$ である。従って、 T_1 の実行後に、 T_2 を実行した場合を示す以下の直列履歴と同値である。

S_1 : $l_{11}[x]; r_{11}[x]; l_{11}[y]; r_{11}[y]; u_{11}[x]; u_{11}[y]; l_{21}[x]; w_{21}[x]; l_{21}[y]; r_{21}[y]; u_{21}[y]; u_{21}[x];$

S_2 : $l_{12}[z]; w_{12}[z]; u_{12}[z]; l_{22}[z]; r_{22}[z]; u_{22}[z];$

以上から、二相ロックトランザクションについて、以下の結果が得られている。

[二相ロックトランザクションの性質] どのトランザクションも二相ロック形式ならば、得られる履歴は直列可能である。

即ち、どのトランザクションも二相ロック形式ならば、これらのトランザクションを複数のデータベースサーバで実行した結果は正しい。

[ロックの解放と独立性]

二相ロック方式では、ロックの解放をいつ行うかが問題となる。オブジェクト x のロックの解放をトランザクション T_i の終了前に行うと、他のトランザクション T_j が、 x を T_i の終了前に操作できる。 T_i が、アボートした場合には、 T_j もアボートされる必要がある。こうした連鎖的アボートを防ぐための方法として、トランザクションの終了時に、獲得した全てのロックを一括して解放する方法がある。この方法を行うと、履歴を厳格なものとする。

[厳格な二相ロック]

- (1) T_i 内の各演算 $op_i[x]$ に対して、 DBS_i 内の x をロック ($l_i[x]$) する。
- (2) T_i がコミットまたはアボートする時に、 T_i のロックをすべて解放 ($u_i[x]$) する。

現在のデータベースシステムでのトランザクションは、厳格な二相ロック形式が取られている。

B. ロックのモード

オブジェクト x を、あるトランザクション T_i がreadをする場合を考える。他のトランザクション T_j も x をreadするとき、 T_i が x をロックしているために、 T_j が x を利用できない。しかし、readはどのような順序で実行されても結果は同じである。このために、各演算 op に対して、ロックモード $mode(op)$ を考える。

[ロックのモード] x に対する二つの演算 op_1 と op_2 をどのように実行しても結果が同じであるとき、 op_1 と op_2 は、交換可能とする。交換可能な演算 op_1 と op_2 のロックモード

mode(op₁) と mode(op₂) は伴立である (compatible) とする。

演算 op₁ と伴立なモードでオブジェクトが他の演算 op₂ でロックされているときには、op₁ は x を操作できる。そうでないときには、op₂ によるロックの解放を待たねばならない。例えば、read 演算のロックモード Rlock は互いに伴立であるが、write 演算のロックモード Wlock は、Wlock と Rlock とともに伴立ではない。これらのロックモード間の伴立関係を図4.18に示す。

	R	W
R	○	×
W	×	×

○：伴立 ×：非伴立

図4.18 ロックモードの伴立性

4.6.3 レプリカオブジェクトの二相ロック

次に、オブジェクト x が複数のレプリカ x₁, ..., x_m を持つ場合を考える。例えば、テーブル Emp の断片 Emp_i=Emp[eno,ename] が、二つのデータベースサーバ DBS₁ と DBS₂ に存在する場合[図4.19]である。

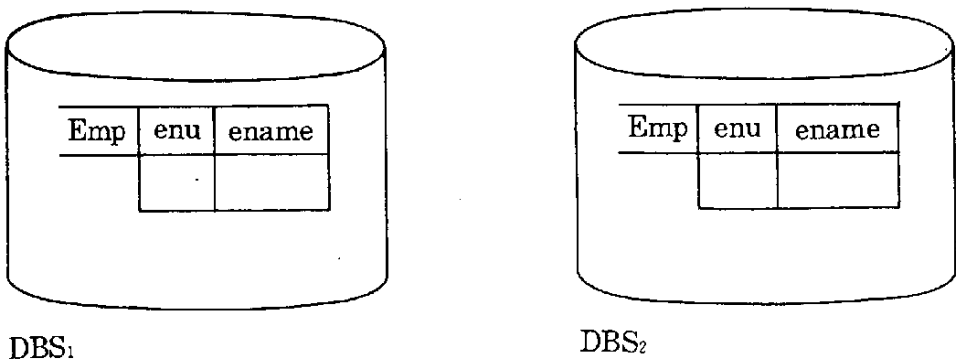


図4.19 レプリカ

A. 基本ロック方式

一つの方法は、オブジェクト x のレプリカ $x_1, \dots, x_m$ を全てロックしてから、 $x_1, \dots, x_m$ を read と write することである。しかし、この方法では、全レプリカをロックするために、通信と処理の負荷が大きくなる。このために以下の方法がある。

[レプリカのロック方式]

- (1) オブジェクト x の read に対しては、あるレプリカ x_i を Rlock モードでロックしてから、 x_i を read する。
- (2) x の write に対しては、全てのレプリカ $x_1, \dots, x_m$ を Wlock モードでロックし、 $x_1, \dots, x_m$ を write する。

この方法では、自分のまたは近くの D_i 内の x_i を read できる。このために、検索中心の応用に適した方法である[図4.20]。

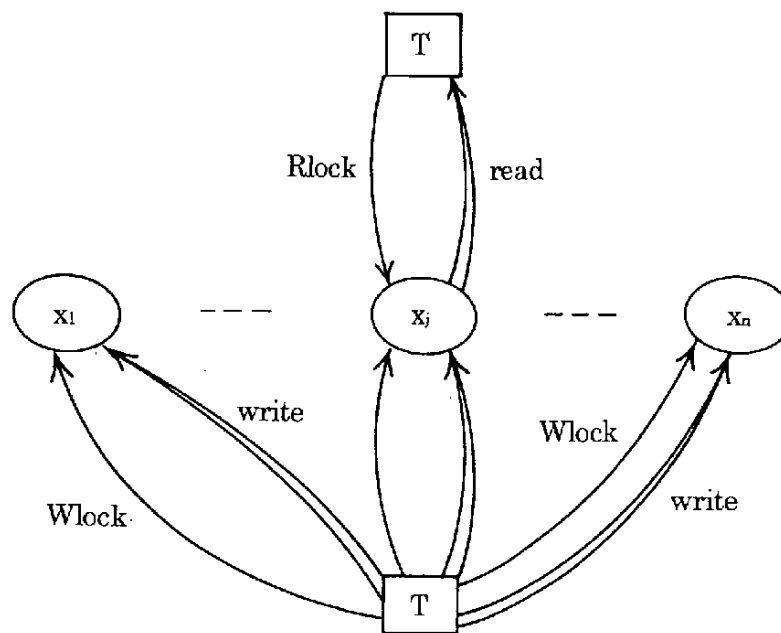


図4.20 レプリカのロック方式

B. 主レプリカロック方式

もう一つの方法は、レプリカの中の一つ、ここでは x_1 をオブジェクト x の主レプリカとして、ロックは主レプリカに対して行う方法である[図4.21]。

[主レプリカロック方式]

- (1) オブジェクト x の read に対しては、 x_1 を Rlock モードでロックしてから、ある x_i を $r_0[x_i]$ する。
- (2) x の write に対しては、主レプリカ x_1 を Wlock でロックし、すべての $x_1, \dots, x_m$ を write する。

この方式では、ロックを主レプリカ x_1 に対してのみ行えばよいので、write 演算のロックの負荷を減少できる。しかし、 x_1 に負荷が集中する。

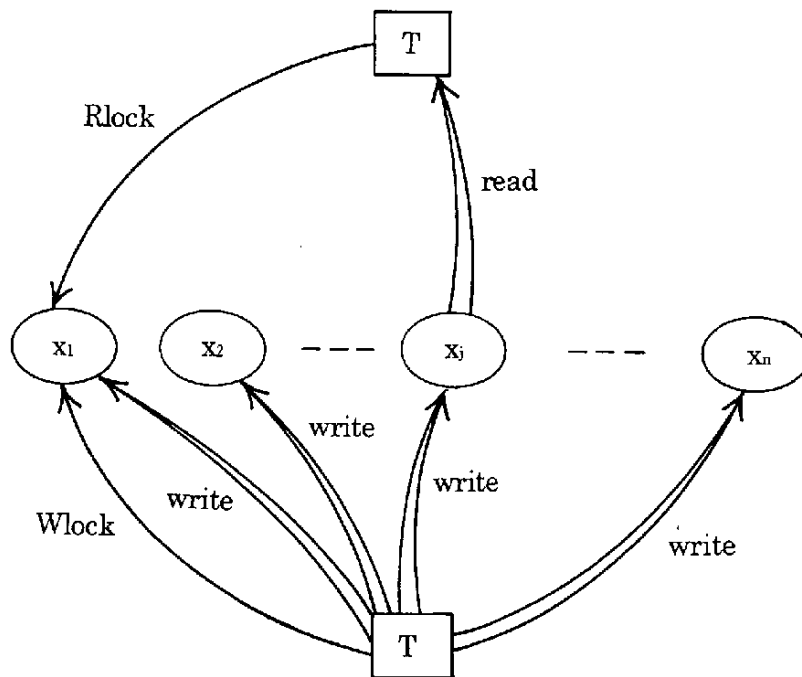


図4.21 主レプリカのロック方式

C. コーラム方式

他に、コーラムに基づいた方法がある。オブジェクト x は、 m 個のレプリカ $x_1, \dots, x_m$ を持つとする。各レプリカ x_i は、版番号 $\text{version}(x_i)$ を持つ。 x は、read コーラム $r\text{-quorum}$ と write コーラム $w\text{-quorum}$ という二つの整数値を持つ。ここで、両値は、以下の制約を満足している。

[コーラムの制約] $r\text{-quorum} + w\text{-quorum} > m$ 。

まず、トランザクションが、 x を read する場合を考える。

[read 方式]

(1) オブジェクト x のレプリカ集合 X の中から、 $r\text{-quorum}$ 個のレプリカを検索する。

これらのレプリカの集合 R を、読出しについてのコーラムとする。

(2) 読出しコーラム R の中で、版番号が一番大きいレプリカ x_i の値を読む。

次に、オブジェクト x を write する場合を考える。

[write 方式]

(1) オブジェクト x のレプリカ集合 X から $w\text{-quorum}$ 個のレプリカを検索する。これらのレプリカの集合を、書込みについてのコーラム W とする。 W の中で、版番号が一番大きなレプリカ x_i を見つける。

(2) $\text{version}(x_i)$ を増加させる。

(3) 書込みについてのコーラム W 内の全レプリカに、書き込まれる値 t と新しい版番号 v とを送る。

(4) W 内のレプリカ x_j が、新しい値 t と版番号 v を受信したとする。 $\text{version}(x_j) < v$ ならば、 x_j の値を t として、 $\text{version}(x_j)$ を v とする。

$r\text{-quorum}$ と $w\text{-quorum}$ の条件は、read を行う物理オブジェクトと、write を行うレプリカの集合は、少なくとも一つのレプリカが共通に含まれている。このために、read

は、最新のオブジェクトの値を読める。write に対しても、全レプリカを write する必要がなく、書込みコーラム内のレプリカを write すればよい。

4.6.4 デッドロック

デッドロックとは、複数のトランザクションが他のトランザクションのロックの解放を待ちあっている状態である。

A. 検出方法

トランザクション間のデッドロックは、資源割当てグラフを用いることにより検出できる。トランザクションをプロセス節点、オブジェクトを資源節点として、トランザクション T_i がオブジェクト x のロックしているとき、 x から T_i に有向辺を設ける。また、 T_i が x をロックしようとして待っているときには、 T_j から x に有向辺を設けるものである。特に、複数のデータベースサーバからなるシステムでは、資源割当てグラフから、オブジェクトを除き、 T_i が T_j によりロックされているオブジェクトを待っているときに、 T_i から T_j に有向辺を設ける待ちグラフ (wait-for graph) がよく用いられる。

[待ちグラフ]

- (1) 各トランザクション T_i に対して、節点 T_i を設ける。
- (2) T_i が T_j によりロックされているオブジェクトの解放を待っているならば、 T_i から T_j に有向辺を設ける ($T_i \rightarrow T_j$)。

三つのトランザクション T_1 、 T_2 、 T_3 があり、おののがオブジェクト x 、 y 、 z をロックしているとする。このとき、 T_1 が y を、 T_2 が z を、 T_3 が x のロックを行おうとしているとする。ここで、 T_1 は T_2 の y の解放を待ち、 T_2 は T_3 の z の解放を待ち、 T_3 は T_1 の x の解放を待つ。この関係を示す待ちグラフを、図4.22に示す。

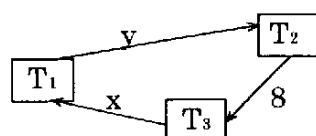


図4.22 待ちグラフ

図4.22に示す状態は、 T_1 、 T_2 、 T_3 が互いに待ちあっていることからデッドロック状態である。図4.22をみると、巡回有向閉路が存在していることがわかる。

[デッドロック] 待ちグラフが巡回閉路を含むならば、かつこのときに限りシステムはデッドロックしている。

次に、図4.23に示すように、 DBS_1 がオブジェクト x と y を持ち、 DBS_2 が z を持つ場合を考える。図4.22に示したように、三つのトランザクション T_1 、 T_2 、 T_3 がおのの x 、 y 、 z をロックしているとする。このとき、 T_1 が y を、 T_2 が z を、 T_3 が x のロックを行おうとしているとする。 DBS_1 を見てみると、 x と y を利用している T_1 、 T_2 、 T_3 間には、巡回有向閉路は存在しない。 DBS_2 も同様である。しかし、分散データベースシステムとしては、デッドロックとなっている。このように、分散データベースシステムでは、各データベースサーバ内でデッドロックとなっていなくても、システム全体としてみたときに、デッドロックとなっていることがある。分散データベースシステムでデッドロックを検出するためには、システム全体の待ち関係を把握するために、データベースサーバ間、トランザクション間での通信が必要となる。

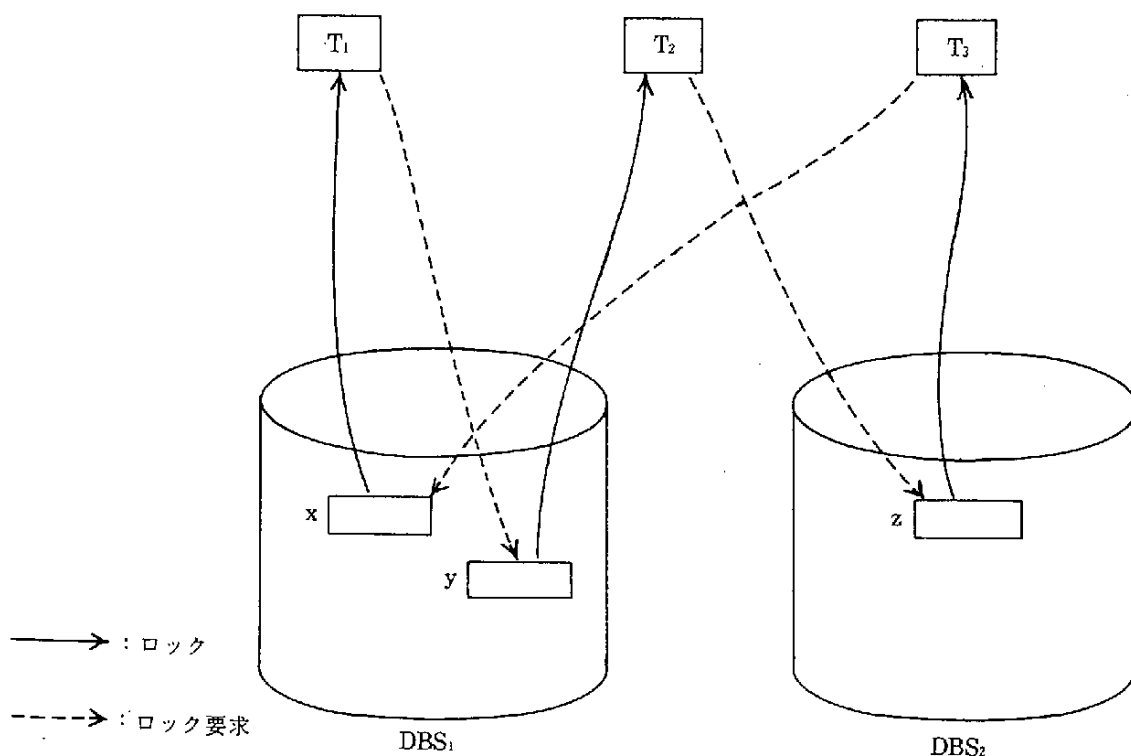


図4.23 複数データベースサーバ間のデッドロック

C. 検出方法

デッドロックを解決する方法として、デッドロックを検出して、デッドロック状態を解消する方法がある。この方法は、商用のシステムで広く用いられているものである。

[デッドロック検出解除方法]

- (1) 現在の状態に対する待ちグラフ G を作り、巡回閉路を探す。
- (2) 見付ければ、この閉路内のあるトランザクション T_i を選択し、 T_i をアボートさせる。

T_i のアボートにより、 T_i がロックしていたオブジェクトのロックが解放される。これにより、デッドロックを解消できる。

[擬デッドロック] デッドロックが生じていなくても、デッドロック検出手順がデッドロックが生じていると見なしてしまう状態が、偽デッドロックである。

トランザクションが二相ロック形式である場合には、以下の性質が示されている [BERN87]。

[性質] 各トランザクションが二相ロック形式であり、頻繁にアボートしないならば、偽デッドロックは生じない。

[デッドロック検出方法の問題点]

デッドロックの検出・解除方法では、以下が問題となる。

(1) 検出の間隔

周期的にデッドロック検出を行う方法が用いられる場合が多い。このとき、周期をどの位にするかが問題となる。周期を短くすると、デッドロック検出のための負荷が増大してしまう。一方、周期を長くすると、デッドロックが長時間生じていることになる。

(2) アボートするトランザクションの選択

デッドロックが検出されたとき、どのトランザクションを強制終了させるかが問題となる。実行されている時間の一番短いもの、長いもの、ロックしているオブジェクト数

が最少のもの等を選択する方法がある。

D. 防止方法

防止方法では、デッドロックが生じないように、トランザクションにオブジェクトをロックさせる方法である。このための方法として以下がある。

- (1) オブジェクトを順序付けて、トランザクションに、この順にロックさせる。
- (2) トランザクションを全順序付ける。

[オブジェクトの順序化]

- (1) は、オブジェクトをロックする順番をあらかじめ決めておく方法である。例えば、オブジェクト a 、 b 、 c に対して、各トランザクションは、この順にオブジェクトのロックを行わせる方法である。この方法では、順位の高いオブジェクトにロック要求が集中することになり、トランザクション実行の同時実行性を減少させてしまう点が問題となる。

[トランザクションの順序化]

次に、(2)の方法について述べる。各トランザクション T_i に対して、 T_i が起動された時刻を T_i の時刻印 (time stamp) $ts(T_i)$ として与える。ここで、 x をオブジェクト、 T_1 と T_2 をトランザクションとする。いま、 x は T_2 によりロックされているとし、新たに、 T_1 が x をロックしようとしている場合を考える [図4.24]。



図4.24 待ち関係

以下の二つの方法がある。

[横取り (preemption) 方法]

- (1) $ts(T_1) < ts(T_2)$ ならば (T_1 が T_2 より古い)、 T_2 をアボートさせ、再実行させる。

(2) $ts(T_1) > ts(T_2)$ ならば、 T_1 を待たせる。

[非横取り (nonpreemption) 方法]

(1) $ts(T_1) < ts(T_2)$ ならば (T_1 が T_2 より古い)、 T_1 をアボートさせ、 T_1 に新しい時刻印を与えて再実行させる。

(2) $ts(T_1) > ts(T_2)$ ならば、 T_1 を待たせる。

以上のデッドロック防止方法では、デッドロックは生じない。しかし、あるトランザクションがロックしているオブジェクトを、他のトランザクションがロックしようとしたときに、トランザクションのアボートと再実行が生じる場合がある。トランザクションがどの位再実行されるかが問題となる。横取り方法では、トランザクションは時刻印の小さい(即ち、古い)ものが優先して実行される。これに対して、非横取り方法では、新しいトランザクションが優先して実行される。従って、オブジェクト x を操作しようとする古いトランザクションがあるとき、これらより新しいトランザクションが x を獲得してしまうと、古いものはアボートされてしまう。このために、横取り方法の方が、一般的に、非横取り方法よりも、アボートされるトランザクションは少ない。

4.7 楽観的二相ロック方式

クライアント c でトランザクション T が実行されているとする。 T は、オブジェクト o を操作するとする。オブジェクト o は、レプリカ集合 $r(o) = \{o_1, \dots, o_m\}$ を持つとする。楽観的方式は、オブジェクト x に対しては、read 演算が中心であり、write 演算はあまり行われないう仮定に基づいている。楽観的方法では、write は、ロックを行わずに実行され、トランザクションのコミット時に、他のトランザクションが x を read/write したかどうか調べられる。もし他のトランザクションが x を操作していれば、トランザクションはアボートされる。

[楽観的二相ロック方式]

- (1) トランザクション T がオブジェクト o を read を行うとする。このとき、すべてのレプリカ $o_1, \dots, o_m$ を Read ロックする。
- (2) トランザクション T がオブジェクト o を write するとする。このとき、どのレプリカもロックしない。
- (3) トランザクション T がコミットするときに、 T がオブジェクト o を write しているならば、すべてのレプリカを Write ロックする。Write ロックできるならば、コミットする。Write ロックできないときには、トランザクション T をアボートする。トランザクションがコミットまたはアボートするときには、 T のロックを全て解放する。

モバイルデータベースでは、データベースサーバのデータがクライアントにキャッシングされ、クライアントはこのデータをローカルに操作を行う。従って、データベースサーバ内のオブジェクト o のレプリカ $o_1, \dots, o_m$ がクライアント $c_1, \dots, c_m$ に存在することになる。このとき、データベースサーバ内のオブジェクト o とクライアント c_i 内のレプリカ o_i の間の一致性と、レプリカ間の一致性を保つ必要がある。このとき、以上の楽観的な二相ロック方式を用いる。

5 章 データの配置方法

5.1 テーブル分散方法

複数のデータベースサーバから構成される分散データベースシステムの目的の一つは、利用者にデータの分散の独立性を提供することである。利用者から見える分散データベースシステムのデータベースは、テーブルの集合である。これの全体層を示し、ここでは論理テーブルとし、その集まりを論理データベース LDB とする。論理テーブル内の値は、実際には、あるデータベースサーバ DBS_i 内の物理的テーブル内に格納される[図5.1]。物理テーブルは、ローカル層のデータである。ここでは、論理テーブルを、どのように物理テーブルとして格納するかについて考える。

[分散方法]

- (1) 断片化
- (2) 冗長化。

論理的テーブルを、射影演算と制限演算により分割した各要素を断片 (fragment) とする。各断片を一つの物理的テーブルとして、あるデータベースサーバに格納する。断片化には、大別して、垂直断片化と水平断片化とがある。

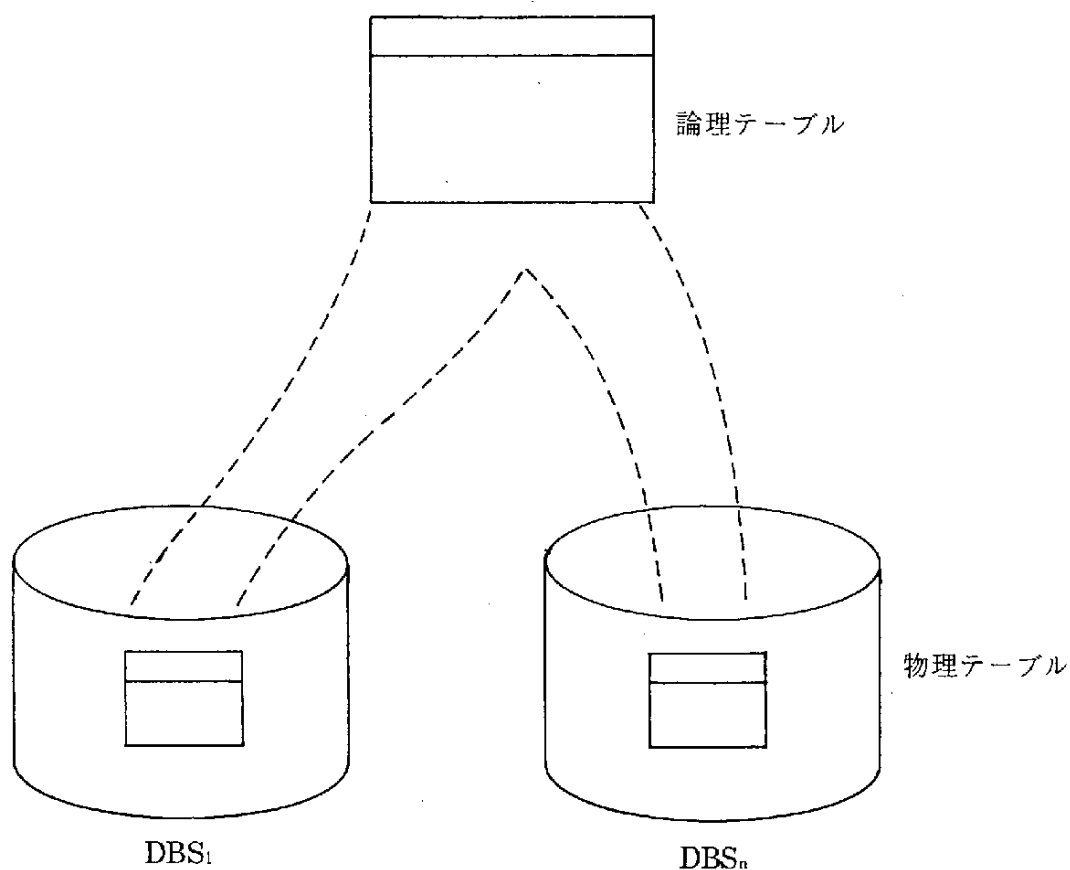


図5.1 論理テーブルと物理テーブル

5.2 垂直断片化

[垂直断片化]

図5.2に垂直断片化の例を示す。ここで、論理的テーブル Emp を、射影断片 $Emp1 = Emp[eno, ename]$ と $Emp2 = Emp[eno, dno]$ に分割し、各々をデータベースサーバ DBS_1 と DBS_2 に記憶した例である。

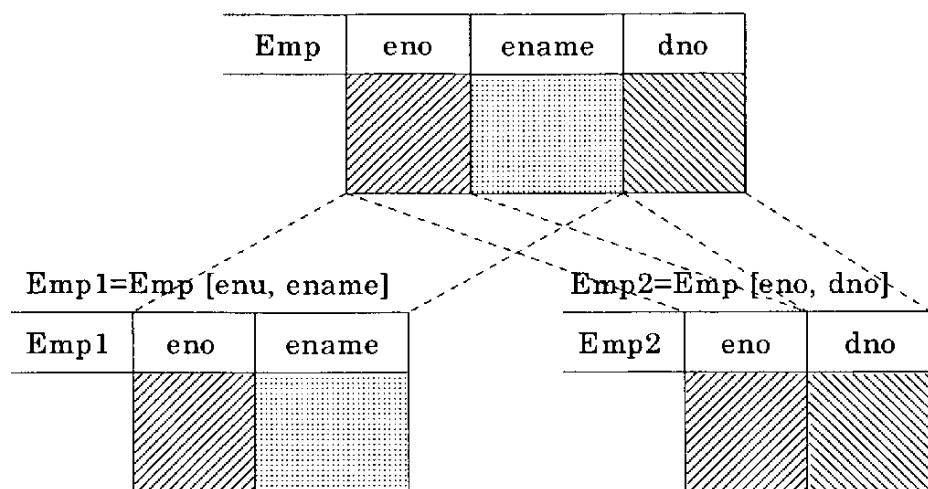


図5.2 垂直断片化

ここで、Emp1と Emp2を自然結合した結果は Emp とならねばならない (即ち、 $Emp = Emp1 * Emp2$)。このために、各断片は、Emp の主キー eno を含む必要がある。例えば、 $Emp3 = Emp[ename, dno]$ を考え、Emp1 と Emp3 の二つの断片に分割するとする。 $Emp1 * Emp3$ は Emp と異なったテーブルになってしまう [図5.3]。

[垂直断片化の条件]

テーブル R を射影断片 $R_1, \dots, R_m$ に分割したとする。このとき、 $R = R_1 * \dots * R_m$ でなければならない。

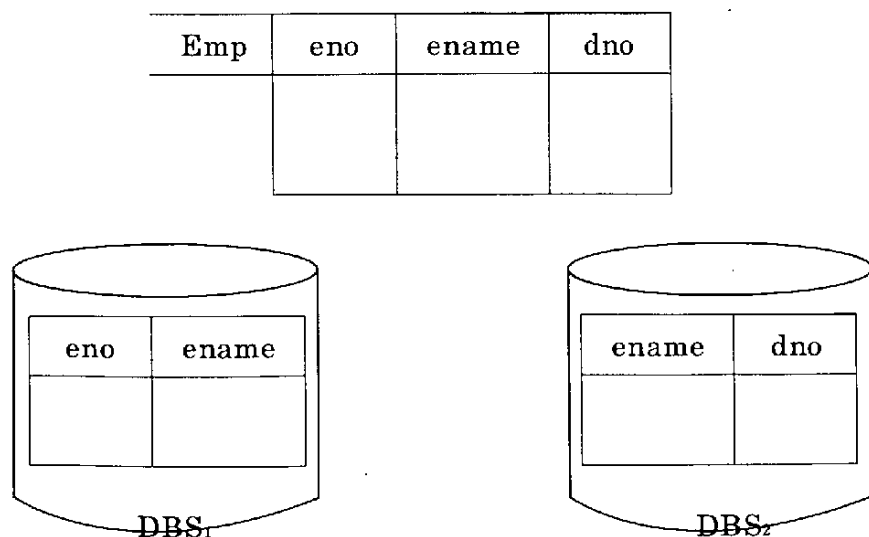


図5.3 正しくない垂直断片化

[断片化方法]

ここで、どのような属性ごとに断片とするかが問題となる。例えば、図5.2で、enameのみを検索する応用は、DBS₁内の断片 Emp1を検索すればよい。dnoのみを検索する応用は、DBS₂内の Emp2を検索する。このように、負荷を二つのデータベースサーバに分散できる利点がある。しかし、enameとdnoを利用する検索に対しては、Emp1とEmp2の両方を結合することが必要である。例えば、DBS₂からEmp2を、DBS₁に送信する必要がある。Emp2の転送には通信時間がかかることから、断片化せずにEmpを検索した方が検索時間が短くなる。このように、垂直断片化するためには、各応用が複数の断片を利用しなくてよいように断片を定める必要がある。

5.3 水平断片

垂直断片化では論理的テーブルの射影を断片としていたが、行の集合である制約を断片とすることもできる。これを水平断片化とする。例えば、図5.2に示すテーブル Emp を、dno = 1 の行の集合 Emp1 = Emp[dno = 1] と dno = 2 の Emp2 = Emp[dno = 2] に分割し、各々を異なったデータベースサーバ DBS₁ と DBS₂ に記憶する[図5.4]。dno = 1 の行についての検索は DBS₁ に対して行えて、dno = 2 についての検索は DBS₂ に対して行える。このように、負荷を二つのサーバに分散できる。さらに、Emp 全体を探索する場合に、Emp1 と Emp2 を並列に探索できる。

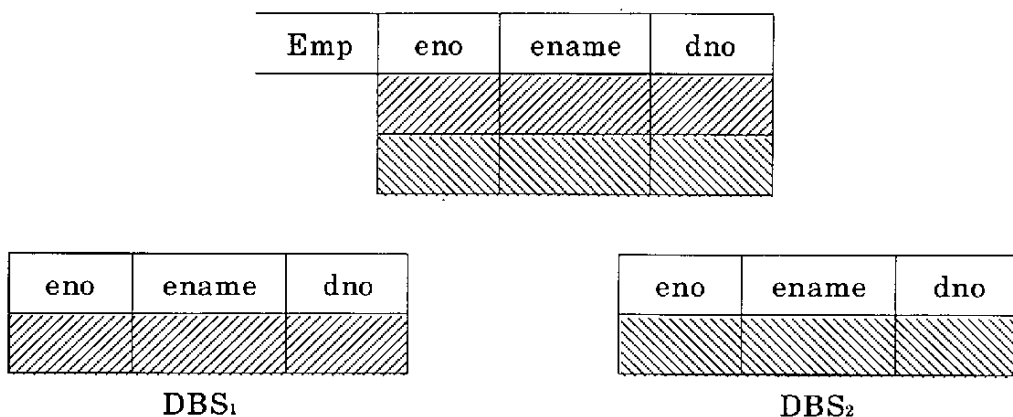


図5.4 水平断片化

実際には、垂直と水平の断片化の組み合わせを断片として用いることができる。

5.4 冗長化

断片を複数のデータベースサーバに記憶することを、断片の冗長化とする。このように、断片を冗長化したものを、断片のレプリカ (replica) とする。図5.2に示した射影断片 **Emp**[eno, ename]が二つのレプリカ **Emp1**と **Emp3**を持ち、おのおの異なったデータベースサーバ **DBS₁**と **DBS₃**に記憶されているとする。**DBS₁**が故障しても、レプリカ **Emp3**が **DBS₃**に存在するので、**Emp**に対する操作を行える。

レプリカを持つことの利点として、以下がある。

[利点]

- (1) 利用者の必要とする断片が、自分の近くのデータベースサーバに存在するので、通信コストを低減できる。
- (2) データベースサーバの故障に対して、断片の他のレプリカがデータベースサーバに存在する信頼性と可用性を向上できる。

[短所] 更新演算に対して、レプリカ間の一致性を保つための制御が必要となる。

5.5 視野

断片化と冗長化は、論理テーブルを、どのように複数のデータベースサーバに格納する方法であった。即ち、トップダウンの設計である。これに対して、各データベースサーバ内にあるテーブルから、どのように論理テーブルを与えるかを考える。このとき、物理テーブルに対する視野として、論理テーブルを与える。

例えば、データベースサーバ **DBS₁**がテーブル **Emp**を持ち、**DBS₂**がテーブル **Dept**を持つ場合を考える。これに対して、図2.2に示すような視野 **ED**(name, dname)を定義

する。利用者は、Emp と Dept の所在と独立に、視野 DE に対して検索を行える。Emp と Dept に対する検索をどのように実行するか、例えば、Dept を DBS₂ から DBS₁ に送信して DBS₂ で Emp と結合を行うかは、システムが行う。

DE を用いて、更新を行うことには、問題がある。たとえば、DE に対して、name = "A" で dname = "DDD" である行の更新を行うとする。このとき、Emp から ename = "A" の行を消去し、Dept から dname = "DDD" である行を消去すると、DE から dname = "DDD" のすべての行が消えてしまう。同様に、DE に ename = "A" で、dname = "HHH" の行を追加すると、Dept に新しく dname = "HHH" なる行を追加せねばならない。しかし、Dept の主キー dno の値が未定であるために、実体インテグリティ制約を乱すことから、Dept に追加を行えない。

6章 モーバイルステーションによるデータベースサービス

高速通信ネットワークの発展により、データベースシステムも変化してきている。今後のデータベースシステムとして、以下のものを考える必要がある。

- (1) 統合データベースシステム
- (2) 個人データベースシステム。

これまでのデータベースシステムは、固定的なコンピュータ内に構成され、データベース管理者のもとに集中的に管理されてきていた。主に、企業等の組織体で管理され、利用されてきたものである。これらのデータベースシステムを、統合データベースシステムとする。今後も、情報提供データベースシステムとして、大規模なシステムとして構築されていくものと思われる。

これに対して、VLSI等のハードウェア技術、無線等の通信技術の発展により、高機能、高性能な携帯型の端末が利用されてきている。これらは、無線通信機能を提供、ワイヤレスで通信を行う機能を有している。こうした携帯型のステーションは、単にネットワークの端末または表計算等の処理を行うものとして利用されるだけでなく、各個人のデータを蓄積する個人データベースシステムとしても利用されてきている。各個人は、いつでも、どこでもコンピューティングと通信を行うことができる。例えば、これまで紙等で保管されていた情報が、個人の携帯型ステーション内に、データベースシステムとして構築されてきている。今後のデータベースシステムは、こうした個人用のデータベースシステムが中心となっていくものと思われる。さらにこれらの種々のデータベースシステムがインターネットにより相互接続される。極論すれば、世界中の各個人が個人データベースシステムを持ち、それらの膨大な数のデータベースシステムが相互接続された超分散システム[図6.1]となっていく。このとき、以下が問題となる。

- (1) 各データベースシステムの関連付け。
- (2) 必要なデータベースシステムの検索。
- (3) モーバイルデータベースシステム。

超分散システムでは、膨大な数のデータベースシステムの中から、必要なデータベースシステムをどのように見つけ、相互に関連つけていくかが問題となる。

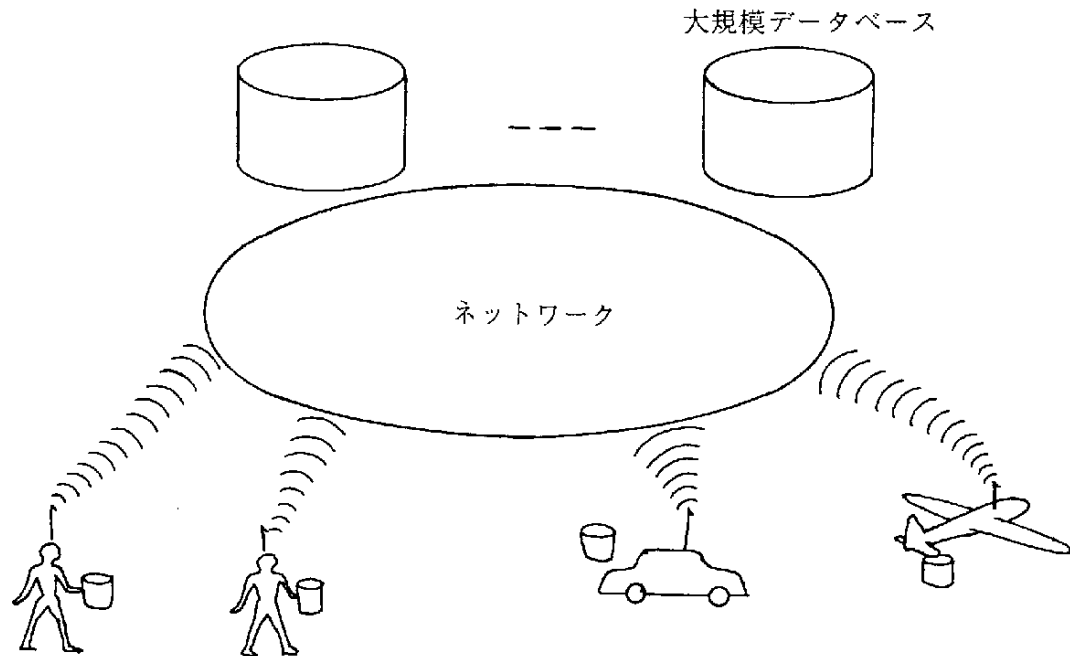


図6.1 超分散システム

さらに、こうした環境では、データベースシステムも移動するもの、即ち、モバイルデータベースシステムと考えることが必要となる。通信技術により、相互に移動するステーション間での通信コネクションを維持することができている。各個人も、携帯ステーションとともに、自宅により高機能、高性能なステーションを持ち、この中に携帯ステーション内のデータベースシステムのコピーを持つ。例えば、バックアップのためである。また、統合データベースシステムから、自分のステーションにデータをダウンロードすることもある。このように、モバイルデータベースシステムでは、データのレプリカが複数存在してくることが問題となる。情報は、これまで、固定的なデータベースシステム内に記憶され、これを複数の利用者が共有するものであった。データベースシステム内のデータを参照し、これを更新するものであった。これに対して、個人データベースシステムが移動する環境では、情報は、一箇所に存在するのではなく、ネットワーク内を移動し、その複製がレプリカとして存在することになる。モバイルデータベースシステムの問題の一つは、こうしたレプリカをいかに利用していくかである。

本報告書では、まず、複数のレプリカと利用者のステーション内のクライアントが移動する環境下で、どのように検索を行うかについて検討した。レプリカの移動を、そのレプ

リカが提供するサービス品質 (QoS) の変化として、モデル化を行った。サービス品質としては、通信速度、遅延時間、通信コスト、通信誤り等の品質がある。検索では、最良のサービス品質を提供するレプリカを見つけることが必要となる。さらに、検索処理を行っている間に、レプリカまたはクライアントの移動にともない、利用していたレプリカのサービス品質が変化する。従って、利用している間も、より良いサービス品質を提供するレプリカがあれば、これを見つけていくことが必要となる。本報告では、このための処理方式を示した。

さらに、クライアントが一つ以上のオブジェクトを更新するトランザクション管理を考える必要がある。このために、本報告書では、これまでのトランザクション管理方式を概観し、モバイルデータベースシステム用のトランザクション管理方式として、楽観的な二相ロック方式を示した。クライアント内のトランザクションが、オブジェクトを更新するとオブジェクトのレプリカも更新する必要がある。レプリカとしては、データベースサーバからクライアントにキャッシングされたデータがある。レプリカが多数存在することから、これらを全てロックすることは、通信時間、通信量の観点から効率的ではない。このために、レプリカの操作を行った後に、他のトランザクションがこのオブジェクトを操作していないならば、更新を行う楽観的な方法を示した。

今後のデータベースサービスも、これまでの固定的なデータベースシステムによるサービスだけでなく、個人データベースシステムのような個人またはグループにより構築されてきた情報のサービスも重要となってくる。インターネットでも、WWWサーバを用いて、世界中に個人、研究室といった単位での情報発信が行われている。さらに、モバイル通信技術の利用により、移動する個人が情報発信を行っていくことが考えられる。データベースシステムも、これまでの枠組みから脱却した、新しい発想が求められていると考えられる。例えば、今データベースシステムというと多くの利用者は、インターネットで利用されている WWW サーバを思い描くだろう。これらの WWW のデータベースシステムは、リレーショナルデータベースシステム、オブジェクト指向データベースシステムといった伝統的なデータベース管理システムを利用していない。むしろ、Unix 等のファイルを利用した簡単なデータ管理を行っているに過ぎない。データベースシステムも、その一般的な仕組みの研究開発も必要であるが、インターネット、モバイル通信等の環境の変化に対して、どのようにデータを提供できるかについての研究が求められていると考えられ

る。このために、本調査研究では、モバイルデータベースシステムとして、オブジェクトの移動をモデル化し、その利用方法についての方式を示した。

参考文献

- [BARB93] Barborak, M. and Malek, M., "The Consensus Problem in Fault-Tolerant Computing," ACM Computing Surveys, Vol.25, No.2, 1993, pp.171-220.
- [BERN87] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems," Addison Wesley, 1987.
- [BIRM91] Birman, K., Schiper, A., and Stephenson, P., "Lightweight Causal and Stomic Group Multicasr," ACM Trans. Computer Systems, Vol.9, No.3, 1991, pp.272-314.
- [BLOU89] Blough, D. M., Sullivan, G. F., Masson, G. M., "Fault Diagnosis for Sparsely Interconnected Multiprocessor Systems," Proc. 19th IEEE FTCS-19, 1989, pp.62-69.
- [BUDH94] Budhiraja, N., Marzullo, K., Schneider, B. F., and Touerg, S., "The Primary-Backup Approach," Distributed Computing Systems, ACM Press, 1994, pp.199-221.
- [CARE91] Carey, J. M. and Livny, M., "Conflict Detection Tradeoffs for Replicated Data," ACM TODS, Vol. 16, No.4, 1991, pp.1-12.
- [CASA93] Casavant, T. L. and Singhal, M., "Distributed Computing Systems," IEEE Computer Society Press, 1993.
- [CERI84] Ceri, S. and Pelagati, G., "Distributed Databases: Principles and Systems," McGraw Hill, 1984.
- [CHAND83] Chandy, K. M., Misra, J., and Haas, L. M., "Distributed Deadlock Detection," ACM Trans. Computer Systems, Vol.1, 1983, pp.144-156.
- [CHAND85] Chandy, K. M. and Lamport, L., "Distributed Snapshots: Determining Global States of Distributed Systems," ACM TOCS, Vol.3, No.1, 1985, pp.63-75.
- [CHEN86] Chen, P.P.-S., "The Entity-Relationship Model: Toward a Unified View of Data," ACM Trans. on Database Systems, Vol.1, No.1, 1976, pp.9-36.
- [CODA73] CODASYL, "Data Description Language Journal of Development,"

Canadian Government Publishing Centre, 1973.

[Codd70] Codd, E. F., "A Relational Model for Large Data Banks," CACM, Vol.13, No.6, 1970.

[DENN82] Denning, D. E. R. and Denning, ***, "Cryptography and Data security," Addison-Wesley, 1982.

[DU89] Du, W. and Elmagarmid, "Quasi Serializability: A Correctness Criterion for Global Concurrency Control in Heterogeneous Database Systems," Proc. of the VLDB, 1989, pp.347-355.

[ELLI91] Ellis, C. A., Gibbs, S. J., and Rein, G. L., "Groupware," Comm. ACM, Vol.34, No.1, 1992, pp.38-58.

[ELMA85] Elmasri, R., Weeldreyer, J., and Hevner, A., "The Category Concept: An Extension to Entity-Relationship Model," Data and Knowledge Engineering, North-Holland, 1985, pp.75-116.

[EPPI91] Eppinger, J. L., Mummert, L. B., and Spector, A. Z., "Camelot and Avalon." Morgan Kaufmann, 1991.

[FISC85] Fischer, M. J., Lynch, N. A., and Paterson, M. S., "Impossibility of Distributed Consensus with One Faulty Process," JACM, Vol.32, No.2, 1985, pp.374-382.

[GARC82] Garcia-Molina, H., "Elections in a Distributed Computing System," IEEE Trans. on Computers, Vol.31, No.1, 1982, pp.48-59.

[GARC85] Garcia-Molina, H. and Barbara, D., "How to Assign Votes in a Distributed System, JACM, Vol.32, No.4, 1985, pp.841-860.

[GARC87] Garcia-Molina, H. and Salem, K., "Sagas," Proc. of the ACM SIGMOD Conf., 1987, pp.249-259.

[GRAY78] Gray, J., "Notes on Database Operating Systems," Lecture Notes in Computer Science, No.60, 1978, pp.393-481.

[GRUB94] Bruber, R., Kaashoek, F., Liskov, B., and Shira, L., "Disconnected Operations in Thor Object-Oriented Database Systems," Proc. of IEEE Workshop on Mobile Computing Systems and Applications, 1994, pp.13-24.

[HEIM85] Heimbigner, D. and McLeod, D., "A Federated Architecture for

Information Management," ACM Trans. on Office Information Systems, Vol.3, No.3, 1985, pp.253-278.

[HO82] Ho, G. S. and Ramamoorthy, "Protocols for Deadlock Detection in Distributed Database Systems," IEEE Trans. Software Eng., Vol.***, No.11, 1982, pp.554-557.

[HOLT72] Holt, R. C., "Some Deadlock Properties of Computer systems," ACM Computing Surveys, Vol.4, No.****, 1972, pp.179-196.

[HUAN94] Huang, Y., Sistla, P, and Wolfson, O., "Data Replication for Mobile Computers," Prof. of ACM SIGMOD, 1994, pp.13-24.

[IEEE87] Proc. of the IEEE, Special Issue on Distributed Database System, 1987.

[JING95] Jing, J., Bukhres, O., and Elmagarmid, A., "Distributed Lock Management for Mobile Transactions," Proc. of the ICDCS-15, 1995, pp.118-125.

[KIM85] Kim, W., Reiner, D., and Batory, D., "Query Processing in Database Systems," Springer-Verlag, 1985.

[KIM90] Kim. W., "Object-Oriented Databases: Definitions and Research Directions," IEEE Trans. of Knowledge and Data Engineering, Vol.2, No.3, 1990.

[KIST92] Kistler, J. J. and Satyanarayanan, H., "Disconnected Operations in the Coda File System," ACM TODS, Vol. 10, No.1, 1992, pp.2-25.

[KNAP87] Knapp, E., "Deadlock Detection in Distributed Databases," ACM Computing Surveys, Vol.19, No.4, 1987, pp.303-328.

[KUNG81] Kung, H. T. and Robinson, J. T., "On Optimistic Methods for Concurrency Control," ACM TODS, Vol.6, No.2, 1981, pp.213-226.

[LAMP78] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," Comm. ACM, Vol.21, No.7, 1978, pp.558-564.

[LAMP82] Lamport, R., Shostak, R., and Pease, M., "The Byzantine Generals Problems," ACM Trans. on Programming Language and Systems, Vol.4, No.3, 1982, pp.382-401.

[LAND82] Landers, T. and Rosenberg, R., "An Overview of Multibase," Distributed Databases (Schneider, H. -J. ed.), North-Holland, 1982, pp.153-184.

- [LISK88] Liskov, B., "Distributed Programming in Argus," *Comm. ACM*, Vol.31, No.3, 1988, pp.300-312.
- [LITW86] Litwin, W. and Abdellatif, A., "Multidatabase Interoperability," *IEEE Computer*, No.12, 1986, pp.10-18.
- [LU94] Lu, Q. and Satyanarayanan, M., "Isolation-Only Transactions for Mobile Computing," *ACM Operating Systems Review*, Vol.28, No.2, 1994, pp.81-87.
- [LYNC93] Lynch, N., Merritt, M., Weihl, W., and Fekete, A., "Atomic Transactions," Morgan Kaufmann, 1993.
- [MAEK85] Maekawa, M., "A n Algorithm for Mutual Exclusion in decentralized Systems," *ACM Trans. Computer Systems*, Vol.3, No.2, 1985, pp.145-159.
- [MAEK91] 前川、所、清水, "分散オペレーティングシステム." 共立, 1991.
- [MANA93] 真鍋義文, "分散チェックポイント・ロールバックアルゴリズム," *情報処理*, Vol.34, No.11, 1993, pp.1366-1374.
- [MATT89] Mattern, F., "Virtual Time and Global States of Distributed Systems," in *Parallel and Distributed Algorithms* (Cosnard, M. and Quinton, P. eds.), North-Holland, Amsterdam, 1989, pp.215-226.
- [MULL89] Mullender, S., "Distributed Systems," ACM Press, 1989.
- [OSZU91] Ozsü, M. T. and Valduriez, P., "Principles of Distributed Database Systems," Prentice-Hall, 1991.
- [PREP67] Preparata, F., Metze, G., and Chien, R., "On the Connection Assignment Problem of Diagnosable Systems," *IEEE Trans. Elect. Comput.*, Vol.EC-16, No.6, 1967, pp.848-854.
- [RAND75] Randell, B., "System Structure for Software Fault Tolerance," *IEEE Trans. Software Eng.*, Vol.29, No.2, 1975, pp.220-232.
- [RAYN88] Raynal, M., "Distributed Algorithms and Protocols," John Wiley & Sons, 1988.
- [RAYA92] Raynal, M., "About Logical Clocks for Distributed Systems," *ACM Operating Systems Review*, Vol.26, No.1, 1992, pp.41-48.
- [RICA81] Ricart, G. and Agrawala, R. K., "An Optimal Algorithm for Mutual

Exclusion in Computer Networks," Comm. ACM, Vol.24, No.1, 1981, pp.9-17.

[SCHA86] Schantz, R. E., Thomas, R. T., and Bono, G., "The Architecture of the Cronus Operating System," Proc. of the 6th Int'l Conf. on Distributed Computing System (ICDCS-6), 1986, pp.250-259.

[SCHN94] Schneider, B. F., "Replication Management Using the State-Machine Approach," Distributed Computing Systems, ACM Press, 1994, pp.169-197.

[SHET90] Sheth, A. and Larson, J.A., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," ACM Computing Surveys, Vol.22, No.3, 1990, pp.183-235.

[SKEE81] Skeen, D., "Nonblocking Commit Protocols," Proc. of the ACM SIGMOD, 1981.

[SKEE83] Skeen, D. and Stonebraker, M., "A Formal Model of Crash Recovery in Distributed System," IEEE Trans. Software Eng., Vol. SE-9, No.3, 1983.

[SMIT81] Smith, J. M., Bernstein, P. A., et al., "Multibase - Integrating Heterogeneous Distributed Database Systems," Proc. of AFIPS, 1981, pp.487-499.

[SOMA89] Somani, A. K. and Agrawa, V. K., "Distributed Syndrome Decoding for Regular Interconnected Structure," Prod. of 19th IEEE FTCS-19, 1989, pp.70-77.

[TAKI80] Takizawa, M. and Hamanaka, E., "Query Translation in Distributed Databases," Proc. of the IFIP'80, 1980, pp.451-456.

[TAKI83] Takizawa, M., "Distributed Database System - JDDBS," JARECT Vol.7, Computer Science and Technologies (Kitagawa, T., ed.), Ohmsha and North-Holland, 1983, pp.262-283.

[TAKI93] 滝沢 誠, 中村章人, "放送型通信アルゴリズム," 情報処理, Vol.34, No.11, 1993, pp.1341-1349.

[TANE92] Tanenbaum, A. S., "Modern OPERating Systems," Prentice-Hall, 1992.

[THOM79] Thomas, R. H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases, ACM Trans. Database systems, Vol.4, No.2, 1979, pp.180-209.

[THOMA90] Thomas, L. C. and Mukesh, S., "The Delta-4 Distributed Fault-Tolerant

Architecture," Distributed Computing Systems, IEEE CS Press, 1990, pp.223-247.

[TSIC78] Tsichritzis, D. and Klug, A., "The ANSI/X3/SPARC DBMS Framework," Information Systems, Vol.3, No.4, 1978.

[YAMA93] 山下雅史, "分散相互排除問題とコータリ," 情報処理, Vol.34, No.11, 1993, pp.1350-1357.

Type-based Optimisitic Transaction Management on Distributed Mobile Objects

Takeaki Yoshida and Makoto Takizawa
Dept. of Computers and Systems Engineering
Tokyo Denki University

Ishizaka, Hatoyama, Hiki-gun, Saitama 350-03, JAPAN
e-mail {take, taki}@takilab.k.dendai.ac.jp

Abstract

According to the advances of communication technologies, kinds of mobile wireless stations like personal handy systems and intelligent robots are available. Objects support abstract operations and are distributed in not only fixed stations but also mobile ones. Transactions manipulate multiple, possibly replicated objects in mobile and fixed stations. While the objects are moving from one location to others in the system, the quality of service (QoS) supported by the objects are changed. The connection is tentatively closed by the mobile station in order to reduce the power consumption while the operations issued by the mobile station are being computed, i.e. disconnected operations. We discuss the migration and replication methods to treat the disconnected operations. In addition, we present an optimistic concurrency control to maintain the mutual consistency among the replicas by taking into account more abstract types of operations on the objects than read and write on files.

1 Introduction

According to the advances of communication and computer technologies, kinds of mobile wireless stations like *personal handy systems* are available. The distributed systems are composed of mobile and fixed stations interconnected by the communication networks. The fixed stations are connected at the fixed location in the communication network. The mobile stations in a *cell* communicate with the *mobile support station (MSS)* in the cell by using the wireless communication. The mobile support station maintains the *connection* of the mobile station in the cell with another station. If the mobile station moves to another cell, it can continue to communicate with the station through the mobile support station in the cell. Tanaka [17] and Teraoka [18] discuss protocols for supporting connections with the mobile stations.

Users access objects in the fixed server stations through the mobile stations. The mobile stations are not equipped with enough capacity of battery to have long-time communication. In order to reduce the power consumption, the connections among the mobile stations and fixed stations are disconnected while the operations issued by the mobile stations are being computed, i.e. *disconnected* operations [12]. One technique to compute the disconnected operations is to *cache* data in the fixed station like server to the mobile station. Without communicating with the fixed station, users can manipulate the data cached into the mobile station. Barbara [3] and Huang [9] present how to cache the data in the fixed stations to the mobile stations and how to maintain the mutual consistency among the caches and the fixed stations. Jing [11] discusses the locking scheme based on the optimistic two-phase locking [4] on the replicas and a way to reduce the communication overhead to release the locks.

In this paper, the distributed system is assumed to be composed of objects distributed in multiple stations. Each object supports abstract data and operations for manipulating the data, while only *read* and *write* operations are considered in the other papers [3,9,11]. On receipt of the operations, the objects start to compute the operations, which furthermore may issue operations to other objects. On completion of the operations, the objects send back the responses. The computation of each operation on an object is viewed to be *atomic*, i.e. the operation is completely computed or nothing [7,2]. The computation of an operation issued by the operation is also atomic, i.e. *nested* [16,19].

The objects may be replicated into multiple replicas in order to increase the reliability, availability,

and performance. In this paper, we assume that the object is fully replicated; i.e. the replicas have the same data and operations as the object. We discuss an optimistic concurrency control to maintain the mutual consistency among replicas of objects supporting more abstract nested operations than read and write.

According to the movement of the mobile stations, the objects in the mobile stations are viewed to move from one location to different locations. *Mobile* objects are objects which can move from one location to others in the system. *Fixed* objects are in the fixed stations. Each object is considered to support some *quality of service* (QoS) like *response time* and *bandwidth*. Thus, according to the movement of the object o , the QoS of o is *changed*. The movement of o is modeled as the *change* of the QoS supported by o in this paper. Problem is how to support users with the service required by the users under situations where the objects are moving in the system. In this paper, we would like to discuss how to manage transactions which manipulate mobile and replicated objects, which support nested, abstract operations.

In section 2, we present the system model. In section 3, we discuss how to compute disconnect operations. In section 4, we present how to compute operations on mobile objects. In section 5, we discuss how to maintain the mutual consistency among the replicas.

2 System Model

The distributed system is composed of multiple stations interconnected by communication networks [Figure 1]. There are two kinds of stations, i.e. *fixed* and *mobile* ones. The fixed stations are connected at the fixed location of the network. The mobile stations communicate with the mobile support station by using the wireless channel. If the mobile station moves to another cell, it communicates with the mobile support station in the cell. By the current network technologies [18,17], the connection among the stations can be maintained while the stations are moving.

A unit of resource in the system is referred to as *object*, which is composed of abstract data and operations for manipulating the data. Each object o can be manipulated only by the operations supported by o . We assume that each object is stored in one station.

There are two kinds of objects, i.e. *class* and *instance*. The class includes the scheme of the data and

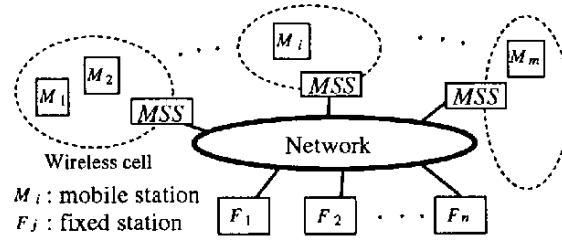


Figure 1: System model

the operations for manipulating the data. The instance is composed of the data instance of the scheme and the operations inherited from the class.

The objects may be replicated into multiple *replicas* which are in different stations. Here, suppose that an object o is replicated into multiple replicas $o^1, \dots, o^l$ ($l \geq 2$) where each o^i is in a station s_i ($i = 1, \dots, l$). If the replicas have the same data and operations as o , o is referred to as *fully replicated* to $o^1, \dots, o^l$. If not, they are *partially replicated*. First, suppose that o is a class. s_i has all operations supported by o if o is fully replicated. s_i has some operations of o if partially replicated. Next, suppose that o is an instance. Each s_i has the data instance and the operations. If o is partially replicated, s_i has a part of the data of o .

If an object o is in a mobile station, the location of o is changed according to the movement of the station. We would like to discuss how the movement of the object o is viewed. For example, the response time to manipulate o may be increased due to the increased latency to o . Thus, the movement of o is modeled to be the change of the quality of service (QoS) supported by o .

[Definition] An object o is *mobile* iff the QoS supported by o is time-variant. $\square$

The computation of an operation op in an object o may invoke operations in other objects. The computation of op is considered to be *atomic*. That is, all the operations invoked by op complete successfully or none of them. If some operation invoked by op fails, all the operations invoked by op have to be aborted. The computation of each operation invoked by op is also atomic. Hence, the computation of the operation is considered to be a *nested transaction* [16, 19].

3 Operations on Disconnected Objects

We would like to discuss how to compute operations on mobile and replicated objects.

3.1 Disconnected operations

Suppose that there are three objects o_i , o_{ij} , and o_{ijk} with the data d_i , d_{ij} , and d_{ijk} , respectively. Suppose that an operation op_i in o_i invokes an operation op_{ij} in o_{ij} and op_{ij} further invokes op_{ijk} in o_{ijk} as shown in Figure 2. op_{ij} manipulates d_{ij} in o_{ij} and op_{ijk} manipulates d_{ijk} in o_{ijk} . Since the

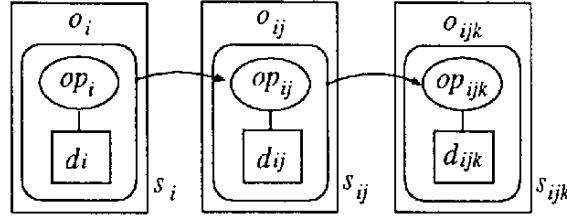


Figure 2: Invocation

mobile station is not equipped with such powerful battery that it can have long-time communication, the mobile station often has to close the connection with other stations to reduce the power consumption. The mobile station also may be disconnected due to the jamming and noise. Thus, the operations may be *disconnected* [11] while the operations are being computed. If the object has no connection with the other objects, the object is a *disconnected*. Objects which are not disconnected are *connected*.

There are ways to continue the distributed computation on mobile and fixed objects in the presence of the disconnected objects:

- (1) migration of objects, and
- (2) replication of objects.

In the migration way, the operations and data of the disconnected object are transferred to another station. On behalf of the disconnected operations, the operations migrated are continued to be computed. The data caching is a kind of migration where only data in the object is copied to another station. In the replication way, the object is replicated into multiple replicas $o_{ij}^1 \dots o_{ij}^n$. If a replica o_{ij}^k used by an object o_i is disconnected, o_i manipulates another replica o_{ij}^k on behalf of o_{ij}^k .

3.2 Migration of objects

First, we would like to discuss how to migrate objects from one station to others. Here, suppose that o_{ij} in Figure 2 is to be disconnected due to the close of the connections. There are two ways for migrating the object:

- (1) to migrate the disconnected object o_{ij} in s_{ij} to another station, and
- (2) to migrate the connected object o_{ijk} in s_{ijk} to the disconnected station s_{ij} .

One way is to migrate the disconnected object o_{ij} to another station. For example, op_{ij} and d_{ij} of o_{ij} are migrated from s_{ij} to another station s_{ijk} as shown in Figure 3. If s_{ijk} has the class of o_{ij} , only d_{ij} can be migrated to s_{ijk} since s_{ijk} has the operation op_{ij} . After migrating o_{ij} to s_{ijk} , op_i can still invoke op_{ij} of o_{ij} in s_{ijk} . If o_{ij} in s_{ij} is reconnected, o_{ij} waits until op_{ij} in s_{ijk} completes. Then, d_{ij} in s_{ijk} is sent to s_{ij} if d_{ij} is changed by op_{ij} . On receipt of d_{ij} , d_{ij} is restored to the data in o_{ij} . In stead of migrating o_{ij} to s_{ijk} , o_{ij} may be migrated to s_i or the other station.

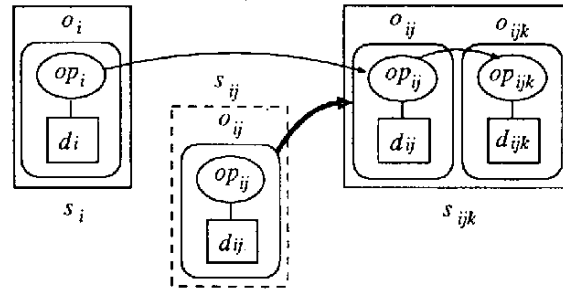


Figure 3: Migration of o_{ij} to s_{ijk}

Another way is to move the connected objects to the station s_{ij} to be disconnected. For example, suppose that o_{ijk} is migrated to s_{ij} as shown in Figure 4. o_{ijk} is migrated to s_{ij} from s_{ijk} . Since o_{ijk} is still connected, o_{ijk} is manipulated by other objects while d_{ijk} is being manipulated in s_{ij} . In the caching method, only d_{ij} is sent to s_{ij} assuming that s_{ij} has the class of o_{ij} , i.e. operations for manipulating d_{ij} . It is problem how to maintain the mutual consistency of d_{ijk} among s_{ij} and s_{ijk} . The problem is discussed already by many papers [3, 9].

As stated now, if o_{ij} is to be disconnected, there are two migration ways, i.e. (1) op_{ij} and d_{ij} of o_{ij} in s_{ij} are migrated to another station or (2) op_{ijk} and d_{ijk} invoked by op_{ij} are migrated from s_{ijk} . It depends on which object o_{ij} or o_{ijk} coordinates the distributed computation. For two objects o_{ij} and o_{ijk} , if o_{ij} coordinates the computation on o_{ij} and o_{ijk} , o_{ij} is referred to as *superior* to o_{ijk} . An object which is not superior is migrated to a superior object. For example, if o_{ij} is in the mobile handy station and a user interactively manipulates o_{ijk} through o_{ij} , o_{ij} is superior to o_{ijk} , i.e. o_{ijk} is migrated into s_{ij} . If neither o_{ij} nor o_{ijk} are superior, o_{ij} and o_{ijk} are referred to as *equivalent*.

Suppose that o_{ij} and o_{ijk} are equivalent. The following migration strategy is adopted to reduce the communication overhead:

[Selection of objects]

- (1) If either o_{ij} or o_{ijk} are updated, the object whose state is not changed is moved to the other.
 - (2) If a volume of operation and data to be sent to s_{ijk} is smaller than o_{ij} , o_{ijk} is migrated to s_{ij} .
- Otherwise, o_{ij} is migrated to s_{ijk} . $\square$

Suppose that an object o_{ij} is updated by the operation op_{ij} and o_{ijk} is not updated by op_{ijk} . If the object o_{ij} is migrated to another station s_{ijk} , o_{ij} in s_{ij} has to be synchronized with the object migrated in s_{ijk} when o_{ij} in s_{ij} is reconnected.

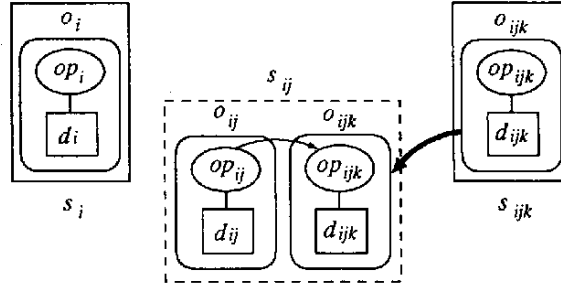


Figure 4: Migration of o_{ijk} to s_{ij}

3.3 Replication of objects

We would like to discuss a case that o_{ij} is replicated into multiple replicas. If one replica o_{ij}^h being manipulated is disconnected, another replica o_{ij}^k is used on behalf of o_{ij}^h . Suppose that o_{ij} is replicated into two replicas o_{ij}^1 and o_{ij}^2 as shown in Figure 5. In this paper, we assume that the objects are fully replicated, i.e. o_{ij}^1 and o_{ij}^2 are the same as o_{ij} . If o_{ij}^1 is to be disconnected, op_i can invoke op_{ij} in the replica o_{ij}^2 and op_{ij} in o_{ij}^2 can invoke op_{ijk} as shown in Figure 5. Here, the current state of o_{ij}^1 has to be sent to o_{ij}^2 . On receipt of the states of d_{ij} and op_{ij} , the states are restored to d_{ij} and op_{ij} in o_{ij}^2 and then o_{ij}^2 starts to compute op_{ij} for the current state received from o_{ij}^1 .

Another way is to abort op_i . By the abortion of op_i , op_{ij} and op_{ijk} are aborted. Then, op_i is restarted and op_i invokes op_{ij} in o_{ij}^2 again.

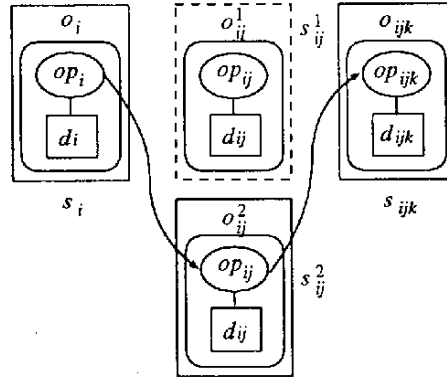


Figure 5: Replication of o_{ij}

4 Operations on Mobile Objects

We would like to discuss how to compute operations on mobile objects.

4.1 Less-qualified operations

Each object supports some service, i.e. operations for other objects. The quality of service means the performance, reliability, availability, and security aspects of the operations. According to the movement of the object, the *quality of service* (QoS) supported by the object is changed while the object supports the same service. For example, the bandwidth between o_{ij} and o_{ijk} in Figure 2 is changed to be lower if o_{ijk} is moved to a station s'_{ijk} which is connected with the lower bandwidth network b [Figure 6]. Thus, the QoS of o_{ijk} is defined for each object o_{ij} issuing operations to o_{ijk} . $QoS(o_{ijk}, o_{ij})$ denotes the QoS which o_{ij} supports for o_{ijk} .

Suppose that o_{ij} is replicated to two replicas o_{ij}^1 and o_{ij}^2 as shown in Figure 4. If the QoS of o_{ij}^1 for o_i , i.e. $QoS(o_{ij}, o_i)$ is degraded to be lower than op_i expects to take from o_{ij} . o_i can use another replica o_{ij}^2 in stead of o_{ij}^1 if o_{ij}^2 supports the better QoS than o_{ij}^1 . Here, let $ReQ(o_{ij}, o_i)$ denote the QoS which o_i requires o_{ij} to support. Let Q_1 and Q_2 denotes two QoS values. $Q_1 \preceq Q_2$ means that Q_1 is *better* than Q_2 . For example, if Q_1 and Q_2 represent the bandwidths 10Kbps and 1Mbps, respectively, $Q_1 \preceq Q_2$.

[Definition] Suppose that op_i invokes op_{ij} . op_{ij} is referred to *less-qualified* for op_i if $QoS(o_{ij}, o_i) \preceq ReQ(o_{ij}, o_i)$. $\square$

Suppose that an object o_{ij} is replicated to $l(\geq 2)$ replicas $o_{ij}^1, \dots, o_{ij}^l$, where each replica o_{ij}^k is stored in a station $s_{ij}^k (k = 1, \dots, l)$. Let $r(o_{ij})$ be a set of replicas $o_{ij}^1, \dots, o_{ij}^l$. o_i has to find a replica o_{ij}^k whose

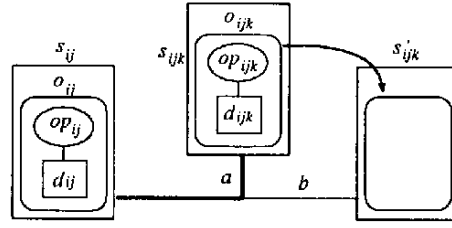


Figure 6: Movement of object

QoS is the best in $r(o_{ij})$. α_i selects one replica o_{ij}^k among the replicas in $r(o_{ij})$ as follows.

[Selection of the replica]

- (1) α_i sends *Rq-QoS* messages to all the replicas $o_{ij}^1, \dots, o_{ij}^l$.
- (2) On receipt of the *Rq-QoS* message from α_i , each replica o_{ij}^k sends back the *Rp-QoS* message with the QoS of o_{ij}^k to α_i ($k = 1, \dots, l$).
- (3) If α_i receives the *Rp-QoS* messages from the replicas, α_i selects one replica o_{ij}^k with the best QoS among them. $\square$

Then, op_i invokes the operation op_{ij} in o_{ij}^k . This method implies larger communication overhead to broadcast *Rq-QoS* messages to all the replicas. Hence, we adopt the following heuristics to select the replica.

[Selection of the replica]

- (1) If there is a replica o_{ij}^k in the same cell as α_i , the replica o_{ij}^k is selected. If there are multiple replicas in the cell, the replica o_{ij}^k which supports α_i with the best QoS among them is selected.
- (2) If there is no replica in the cell, α_i broadcasts the *Rq-QoS* message by the selection algorithm. $\square$

Another way is that there is one coordinator of the replicas, say o_{ij}^1 . o_{ij}^1 monitors the change of QoS of each replica o_{ij}^k . α_i first asks o_{ij}^1 to find the best replica for α_i . Then, o_{ij}^1 selects the best one, say o_{ij}^k .

While op_{ij} is computed in o_{ij}^k , the QoS of o_{ij}^k may be changed according to the movement of o_{ij}^k . If op_{ij} could not support the QoS required, i.e. op_{ij} is less-qualified, α_i can select another replica of α_i which supports the better QoS than o_{ij}^k .

[Resolution of the replicas]

- (1) If the QoS of o_{ij}^k is being degraded for some time units, α_i finds the best replica o_{ij}^h which is better

than o_{ij}^k by the selection procedure.

- (2) If o_{ij}^h is detected, o_i requires o_{ij}^k to send the states of d_{ij} and op_{ij} to o_{ij}^h . On receipt of them from o_{ij}^k , o_{ij}^h restores them to the state. o_i invokes op_{ij} in o_{ij}^h . $\square$

4.2 Faulty replicas

One problem on considering the disconnected operations is how to differentiate the disconnected objects from the faulty objects. Suppose that o_{ij} is faulty in Figure 2. If o_{ij} stops by failure, the connection with o_{ijk} is closed. o_{ijk} cannot know whether o_{ij} is faulty or not because the connection is closed and there is no way to communicate with o_{ij} . Here, we make the following assumptions:

[Assumptions]

- (1) The network is synchronous, i.e. the propagation delay is bounded.
- (2) The computations in the objects are synchronous [6]. $\square$

The assumptions mean that the faulty objects can be detected by the timeout mechanism.

We adopt the following strategy to detect the faulty objects:

[Detection of faulty objects]

- (1) The disconnected object o_{ij} sends periodically *Alive* message to o_i and o_{ijk} .
- (2) After the disconnection, if o_{ijk} or o_i does not receive any message from o_{ij} for some predetermined time units, o_{ijk} considers that o_{ij} is faulty. $\square$

The operational objects have to send *Alive* messages to inform other objects of their being operational.

The *Alive* message is sent by using the connectionless communication.

4.3 Computation of QoS

The QoS of o_i for o is given as $QoS(o_i, o)$. Since o manipulates o_i by an operation op_i , the QoS is given as $QoS(o_i:op_i, o)$. We would like to discuss how $QoS(o_i:op_i, o)$ is computed. Since operations in objects are nested, $QoS(o_i:op_i, o)$ depends on not only the computation of actions in o but also QoSs of operations invoked by op_i . Suppose that op_i invokes operations $op_{i1}, \dots, op_{im}$ of objects $o_{i1}, \dots, o_{im}$, respectively. $QoS(o_i:op_i, o)$ is computed as follows:

$$QoS(o_i : op_j, o) = f_i(QoS(o_{i1} : op_{i1}, o_i), \dots, QoS(o_{im} : op_{im}, o_i), qos(op_i, o)).$$

Here, $qos(op_i, o)$ denotes the QoS required for op_i to manipulate o_i ; f_i is a function which gives the QoS of op_i from the QoSs of $op_{i1}, \dots, op_{im}$. There are kinds of QoS. The computation time of op_i is obtained by adding the computation times of $op_{i1}, \dots, op_{im}$ and op_i , i.e. f_i is "+" if op_i is computed sequentially. If $op_{i1}, \dots, op_{im}$ in op_i are computed in parallel, the QoS is obtained by taking the maximum one among of $op_{i1}, \dots, op_{im}$.

In order to compute the QoS of op_i , o_i asks o_{ij} to send $QoS(o_{ij} : op_{ij}, o_i)$ periodically or each time op_i is invoked.

5 Type Based Optimistic Concurrency Control

We would like to discuss how to maintain the mutual consistency among the replicas.

5.1 Lock modes

Objects may be replicated. Here, for an object o_i , let $r(o_i)$ be a collection of replicas of o_i , i.e. $r(o_i) = \{o_i^1, \dots, o_i^{l_i}\} (l_i \geq 2)$, where each o_i^j is a replica of o_i ($j = 1, \dots, l_i$). Each replica o_i^j is stored in a station s_{ij} ($j = 1, \dots, l_i$). We would like to discuss how to maintain the mutual consistency among the replicas.

Before an operation op_i is applied to o_i , o_i is locked. If o_i is locked, op_i is computed in o_i . If not, op_i waits. Two operations op_i and op_j are referred to as *compatible* iff the states obtained by computing op_i and op_j in any order are the same. In order to increase the concurrency, kinds of lock modes are introduced, e.g. *read* and *write* modes. The objects support more kinds of operations than *read* and *write* of the file objects. An operation op_i of o_i is assigned a lock mode $m(op_i)$. The compatibility relation among the lock modes is defined as follows [13].

[Definition] For every pair of lock modes m_1 and m_2 supported by an object o_i , m_1 is *compatible* with m_2 iff an operation of m_1 is compatible with operations of m_2 . $\square$

If m_1 is not compatible with m_2 , m_1 *conflicts* with m_2 . That is, op_i of m_1 has to wait until the operations of m_2 complete in o_i . For example, a *Bank* object supports operations *deposit* and *withdrawal*. The modes of *deposit* and *withdrawal* are compatible.

Objects support various kinds of abstract operations like *deposit* and *withdrawal* while the database systems support only *read* and *write* operations. Hence, various kinds of lock modes are supported by the objects. The precedence relation among the lock modes is formally defined by Korth [13]. Here, let M_0 be a set of lock modes supported by an object o . For each mode m in M_0 , let $c(m) (\subseteq M_0)$ be a set of modes which m is compatible with.

[Definition] For every pair of modes m_1 and m_2 of an object o , $m_1 \prec m_2$ (m_2 is *stronger* than m_1) iff $c(m_1) \supseteq c(m_2)$. $\square$

Here, $m_1 \preceq m_2$ means that the operations of m_2 conflict with more kinds of operations than m_1 . If neither $m_1 \prec m_2$ nor $m_2 \prec m_1$, m_1 and m_2 are *equivalent* ($m_1 \parallel m_2$). Here, $m_1 \preceq m_2$ or $m_1 \parallel m_2$. Here, *read* $\prec$ *write* because $c(\text{read}) = \{\text{read}\} \supseteq c(\text{write}) = \emptyset$. If the mode of op_1 is stronger than op_2 , op_1 is referred to as *stronger than* op_2 .

5.2 Optimistic locking

The typical scheme to maintain the mutual consistency among multiple replicas is the *read-one* and *write-all* (ROWA) principle. That is, the read operation is issued to one replica while the write operation is issued to all the replicas. If one replica is locked in a read mode, the read operation can be computed in the replica. On the other hand, if all the replicas could be locked in the *write* mode, the write operation is computed in all the replicas. In order to reduce the communication overhead, the optimistic approach [14] is adopted. Carey [4] discusses the optimistic two-phase locking (O2PL) protocol. Jing [11] extends the O2PL so as to reduce the communication overhead by avoiding the release of the locks. In the O2PL, one replica is locked by *read* but the replicas are not locked by *write*. When the transaction commits, the replicas updated are locked by *write*. More abstract types of operations are considered in the objects than the *read* and *write* operations. The read-one and write-all principle can be extended by taking into account the various kind of lock modes.

The second point on the operations is concerned with whether the operations change the state of the object or not. For example, *deposit* and *withdrawal* change the state of *Bank* while they are compatible. If an operation op does not change the state of o , op can be computed in only one replica of o . Otherwise, op has to be computed in all the replicas to keep the mutual consistency among the replicas.

The third point is concerned with whether the operations invoke another operation or not. Suppose

that an operation op_i in o_i invokes op_{ij} in o_{ij} and o_i is replicated to two replicas o_i^1 and o_i^2 . If op_i is computed in o_i^1 and o_i^2 , op_{ij} is invoked twice, i.e. by op_i in o_i^1 and o_i^2 . It implies the inconsistency among o_i and o_{ij} . Hence, if an operation in an object invokes another operation, the operation can be computed in only one replica and the state obtained by computing the operation in the replica has to be transferred to the other replicas to make the states consistent.

[Optimistic locking] Suppose that an operation op of a mode m_1 is issued to o .

- (1) If $m_1 \prec m_2$ for every mode m_2 of o , one replica o^k in $r(o)$ is locked, and op is computed in o^k if op does not change the state of o , otherwise op is computed in all the replicas,
- (2) Otherwise, all the replicas in $r(o)$ are locked, and op is computed in all the replicas. $\square$

Problem is the communication overhead since all the replicas have to be locked by the operations whose modes are not minimal.

5.3 Optimistic type-based locking

We adopt the optimistic approach to reduce the communication overhead, named *optimistic type-based locking* (OTL). We make the following assumption.

[Assumption] The less restricted the operations are, the more often they are used. $\square$

Each operation op locks some number of replicas in $r(o)$ rather than locking all the replicas. The more restricted the operation mode is, the more replicas are locked. For each operation op_i in o_i , a number $q(op_i)$ is given as follows.

- $q(op_i) < q(op_j)$ if $m(op_i) \prec m(op_j)$.
- $1 \leq q(op_i) \leq l_i$.
- for every op_j , if $m(op_i) \preceq m(op_j)$, $q(op_i) = 1$.

op_i locks $q(op_i)$ replicas of o_i . For example, suppose that there are five replicas of an object o_i and o_i has three operations op_{i1} , op_{i2} , and op_{i3} . Suppose that $m(op_{i1}) \prec m(op_{i2}) \prec m(op_{i3})$. $q(op_{i1}) = 1$. $q(op_{i2})$ and $q(op_{i3})$ are, for example, given as 2 and 3, respectively. Before computing op_{i2} , two replicas in five ones are locked.

An operation op_i locks an object o_i by the following scheme.

[Locking scheme]

- (1) Before computing op_i , $q(op_i)$ replicas in $r(o_i)$ are locked in a mode $m(op_i)$. Here, let $s(op_i)$ be a subset of replicas in $r(o_i)$ which are to be locked here.
- (2) If all replicas in $s(op_i)$ are locked, op_i is computed.
 - (a) If op_i invokes operations in other objects, op_i is computed in one replica in $s(op_i)$.
 - (b) Otherwise, op_i is computed in all the replicas.
- (3) If some replica in $s(op_i)$ is not locked, op_i aborts. $\square$

Since a stronger operation op locks more replicas, the operation op is more often aborted if other stronger operations are manipulating the replicas. If a replica is in the same cell as an object invoking op_i , the replica is locked at step (1). The replicas with better QoS are selected to be locked as discussed in the preceding subsection.

We would like to discuss how an operation op invoking op_i commits. The commitment of op_i on multiple replicas is controlled by the two-phase commitment [2]. One replica o_i^k in $s(op_i)$ plays a role of the coordinator and the other replicas are the participants.

[Commitment]

- (1) o_i^k sends a *Prepare* message to all the replicas. The participant replica o_i^j which is not locked by op_i , i.e. o_i^j in $r(o_i) - s(op_i)$, is locked in the mode $m(op_i)$ on receipt of the *Prepare* message. If locked, the replica o_i^j sends back *Yes* message to o_i^k .
- (2) If some replica o_i^j in $r(o_i) - s(op_i)$ is not locked, o_i^j sends *No* to o_i^k .
- (3) If o_i^k receives *Yes* from all the participant replicas, o_i^k sends *Commit* to all the participants. If o_i^k receives *No* from some participant, o_i^k sends *Abort* to the participants sending *Yes*.
- (4) If the participant replica o_i^j receives *Abort*, o_i^j abort op_i if o_i^j had computed op_i .
- (5) If the participant replica o_i^j receives *Commit*, all the replicas in $r(o_i) - s(op_i)$ are locked.
 - (a) Unless op_i invokes operations in other objects, op_i is computed in all replicas in $r(o_i)$ if op_i changes the state, otherwise op_i commits.
 - (b) Otherwise, the state of the replica whose op_i is computed is sent to all the replicas. $\square$

If op_i invokes an operations op_{ij} in another object and op_i is computed in o_i , op_{ij} is computed more than once. In order to avoid the iterated computation, op_i is computed in only one replica, say o_i . In stead of

computing op_i in the other replica, the state of o_i is sent to all the other replicas of o_i . If op_i commits, all the locks on the replicas are released.

6 Evaluation

We would like to evaluate the optimistic type-based (OTL) locking scheme by comparing with the traditional read-one and write-all (ROWA) scheme in terms of the number of transactions aborted and the number of lock requests. Here, let o be an object supporting operations $op_1, \dots, op_h$, which is replicated into replicas $o^1, \dots, o^l$, i.e. $r(o) = \{o^1, \dots, o^l\}$. Suppose that $m(op_i) \prec m(op_j)$ ($i < j$). Let $f(op_i)$ be the probability that op_i is issued to o . Here, $f(op_1) + \dots + f(op_h) = 1$. $q(op_i)$ denotes the number of replicas to be locked by op_i in the OTL scheme. Here, $q(op_1) = 1$, $q(op_h) = l$, and $q(op_i) \leq q(op_j)$ if $i < j$, i.e. $m(op_i) \prec m(op_j)$.

Suppose that the operations $op_1, \dots, op_h$ are randomly issued to o . Let A_O be the probability that an operation is aborted in the OTL scheme. If at least one kind of operation sends the lock request to one replica, the operations are aborted. Hence, A_O is given as $1 - \prod_{i=1}^h (1 - f(op_i) \cdot q(op_i)/l) - \sum_{i=1}^h [f(op_i)q(op_i)/l \prod_{j=1(j \neq i)}^h (1 - f(op_j) \cdot q(op_j)/l)]$. In the OTL scheme, op_i locks $q(op_i)$ replicas. Let A_T be the probability that an operation is aborted in the traditional ROWA way. A_T is obtained by assigning $q(op_1)$ with 1 and $q(op_j)$ with l ($j \geq 2$) in A_O because only op_1 locks one replica and the other operations lock all the replicas in $r(o)$.

Next, let us consider how many lock requests are sent to the replicas. Let L_O and L_T denote the probabilities that each replica is locked in the OTL and traditional ROWA schemes, respectively. L_O is given by $\sum_{i=1}^h f(op_i)q(op_i)/l$. L_T is given by $f(op_1)/l + (f(op_2) + \dots + f(op_h)) = 1 - f(op_1)(l-1)/l$.

A_O , A_T , L_O , and L_T are computed for the number l of replicas where $h = 5$, i.e. o has five operations. Here, $q(op_i) = \lceil l/2^{h-i} \rceil$ ($i = 1, \dots, h$), $f(op_1) = 0.4$, $f(op_2) = 0.2$, $f(op_3) = 0.2$, $f(op_4) = 0.1$, $f(op_5) = 0.1$. Figure 7 and Figure 8 show A_O and A_T , and L_O and L_T for the number l of replicas, respectively. These Figures show that less number of transactions are aborted and less number of lock requests are issued in the OTL scheme than the traditional ROWA.

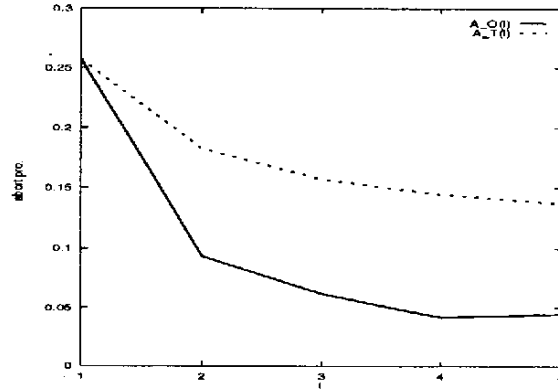


Figure 7: Probability of abort

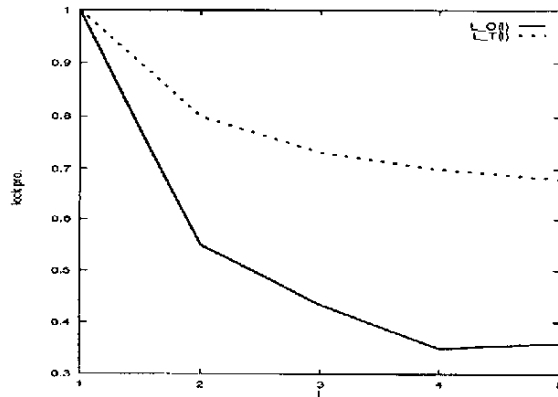


Figure 8: Probability of lock

7 Concluding Remarks

In this paper, we have discussed how to support nested transactions manipulating replicated and mobile objects in the distributed system. We have modeled the mobile objects to be ones whose QoS is changed according to the movement of the objects. We have discussed the optimistic two-phase locking to maintain the mutual consistency among the replicas. Here, the read-one and write-all principal is extended so that the objects can support more kinds of abstract operations than *read* and *write* and the operations are nested.

References

- [1] Badrinath, B. R., Acharya, A., and Imielinski, T., "Structuring Distributed Algorithms for Mobile Hosts," *Proc. of the 14th ICDCS*, 1994, pp. 21-28.
- [2] Bernstein, P. A., Hadzilacos, V., Goodman, N., "Concurrency Control and Recovery in Database

Systems," *Addison-Wesley*, 1987.

- [3] Barbara, D. and Imielinski, T., "Sleepers and Workaholics: Caching Strategies in Mobile Environments," *Proc. of the ACM SIGMOD*, 1994, pp. 1-12.
- [4] Carey, J. M. and Livny, M., "Conflict Detectin Tradeoffs for Replicated Data," *ACM TODS*, Vol. 16, No. 4, 1991, pp.703-746.
- [5] Clifton, C., Garcia-Molina, H., and Bloom, D., "HyperFile: A Data and Query Model for Documents," *VLDB Journal*, Vol. 4, No. 1, 1995, pp. 45-86.
- [6] Fischer, J. M., Nancy, A. L., and Michael, S. P., "Impossibility of Distributed Consensus with One Faulty Process," *Journal of ACM*, Vol.32, No.2, 1985, pp.374-382.
- [7] Gray, J., "The Transaction Concept : Virtues and Limitations," *Proc. of VLDB*, 1981, pp. 144-154.
- [8] Gruber, R., Kaashoek, F., Liskov, B., and Shrira, L., "Disconnected Operation in the Thor Object-Oriented Database System," *Proc. of IEEE Workshop on Mobile Computing Systems and Applications*, 1994, pp. 51-56.
- [9] Huang, Y., Sistla, P., and Wolfson, O., "Data Replication for Mobile Computers," *Proc. of the ACM SIGMOD*, 1994, pp. 13-24.
- [10] Imielinski, T., Viswanathan, S., and Badrinath, B. R., "Energy Efficient Indexing on Air," *Proc. of the ACM SIGMOD*, 1994, pp. 25-36.
- [11] Jing, J., Bukhres, O., and Elmagarmid, A., "Distributed Lock Management for Mobile Transactions," *Proc of the 15th ICDCS*, 1995, pp. 118-125.
- [12] Kistler, J. J. and Satyanarayanan, M., "Disconnected Operation in the Coda File System," *ACM Trans. on Database Systems*, Vol. 10, No. 1, 1992, pp. 2-25.
- [13] Korth, H. F., "Locking Primitives in a Database System," *JACM*, Vol. 30, No. 1, 1983, pp. 55-79.
- [14] Kung, H. T. and Robinson, J. T., "On Optimistic Methods for Concurrency Control," *ACM Trans. on Database Systems*, Vol. 6, No. 2, 1981, pp. 213-226.

- [15] Lu, Q. and Satyanarayanan, M., "Isolation-Only Transactions for Mobile Computing," *ACM Operating Systems Review*, Vol. 28, No. 2, 1994, pp. 81-87.
- [16] Moss, J. E., "Nested Transactions : An Approach to Reliable Distributed Computing," *The MIT Press Series in Information Systems*, 1985.
- [17] Tanaka, R. and Tsukamoto, M., "A CLNP-based Protocol for Mobile End Systems within an Area," *Proc. IEEE International Conf. on Network Protocols (ICNP)*, 1993, pp. 64-71.
- [18] Teraoka, F., Yokote, Y., and Tokoro, M., "A Network Architecture Providing Host Migration Transparency," *Proc. of ACM SIGCOM*, 1991, pp. 209-220.
- [19] Weikum, G. and Schek, H.-J., "Concepts and Applications of Multilevel Transaction and Open Nested Transactions," *Database Transaction Models for Advanced Applications*, 1992, pp. 516-553.
- [20] Yasuzawa, S. and Takizawa, M., "Uncompensatable Deadlock in Distributed Object-Oriented Systems," *Proc. of the International Conf. on Parallel and Distributed Systems (ICPADS)*, 1992, pp. 150-157.
- [21] Yeo, L. H. and Zaslavsky, A., "Submission of Transactions from Mobile Workstations in Cooperative Multidatabase Processing Environment," *Proc of the 14th ICDCS*, 1994, pp. 372-379.

————— 禁 無 断 転 載 —————

平成8年3月発行

発行 財団法人 データベース振興センター
東京都港区浜松町2丁目4番1号
世界貿易センタービル7階
TEL 03 (3459) 8581

委託先 株式会社 イフ・アドバタイジング
東京都新宿区四谷4丁目21番37号
TEL 03 (3355) 5371

印刷所 株式会社 現代工学社
東京都新宿区四谷2丁目13番
TEL 03 (3357) 7161

