コンピュータ・ネットワーク16−3
（1978. 7. 19）

DONAのハイレベルプロトコル

松下　温　　山崎　晴明　　西垣　秀樹　　吉田　勇
（沖電気工業株式会社）

1. はじめに　近年のコンピュータネットワーク技術の発展に伴なって、様々なアプリケーション技術、およびそれをネットワーク上で実現するためのハイレベルプロトコルの議論が盛んになってきている。広く分散処理と呼ばれるこれらの技術のうち、特に各方面で研究、開発が進められているものに、分散型データベースシステム（Distributed Data Base System）がある。このアプローチは、地理的に分散しているデータベースを、ユーザが論理的に単一のデータベースとしてとらえることを可能とするもので、現行のコンピュータネットワークのアプリケーションのうち、解決を要する種々の複雑な問題を含んではいるが、広汎な適用が予想されるものとして注目されている。

本稿では、ハイレベルプロトコルのうち、特にこの分散型データベースサービスプロトコルをとりあげ、当社のネットワーク体系であるDONA（Decentralized Open Network Architecture）上で、分散型データベースシステムを構築する際の問題点、サービスプロトコルの満すべき機能、そのレイヤ構成上の位置、データベースマネジメントシステムの果すべき役割等の考察を行なう。

2. データベースサービスプロトコルの位置　本節では、DONAのレイヤ構成等の一般概念を述べ、DBMSおよびデータベースサービスプロトコルがDONAの全体構成の中でどの位置を占めるのかについて述べる。

DONAは下位レベルから順にData Link control Layer(DLL)、Switching Control Layer(SCL)、Network Facility Layer(NFL)、User/Application Layer(UAL)の4つのレイヤから構成される。DLLは隣接ノード間のデータ転送を保障する機能をもち、SCLはルーチング機能をもつ。NFLはエンド-トゥ-エンドのプロセス間のデータ転送を保障する機能をもつ。DONAの全体的な概要およびNFL以下のレイヤの詳細な議論については他の文献を参照せられたい[1],[2],[3]。

UALはネットワークの標準的なアプリケーションやユーザの業務処理が行われるレイヤであるが、本レイヤは図2.1に示すようにさらにMessage Transfer Sub-Layer(MTSL)、Standard Application Sub-Layer(SASL)、User Application Sub-Layer(UASL)の3つのサブレイヤから構成される。MTSLは上位レイヤの各プロトコルに共通に表われるホスト内のプロセス同士間、端末同士間およびプロセス－端末間のメッセージ（コマンドやテキスト）通信のためのプロトコルであるMessage Transfer Protocol(MTP)を実行するサブレイヤである。SASLはネットワークの標準アプリケーションのためのプロトコルを実行するサブレイヤであり、当面最も汎用的と思われるRemote Job Entry Protocol(RJEP)、Data Base Service Protocol(DBSP)、Time Sharing Service Protocol(TSSP)、Data Access Protocol(DAP)を考慮している。UASLはユーザ毎の業務処理を実行するサブレイヤである。

UALにおいて情報授受の対象となる実体を"サブスクライバ"と呼ぶ。このサブスクライバには2種類あり、ネットワークアプリケーションの基礎となるもので、あらかじめ用意されている汎用的なサブスクライバを標準サブスクライバ（SSB）と呼び、アプリケーション毎にその都度ユーザによって用意されるサブスクライバをユーザサブスクライバ（USB）と呼ぶ。例えば、データベース

サービスの場合には、DBMSがSSBにあたる。

NFLがサブスクライバにみせる情報授受窓口をネットワークポートと呼び、SSBがそのユーザにみせる情報授受窓口をユーザポートと呼ぶ。また、ネットワークポート間の論理パスをネットワークパス、ユーザポート間の論理パスをユーザパスと呼ぶ。分散データベースサービスの場合には、UASL上のユーザプロセスからのアクセス要求に対して、SSBであるDBMSは、その要求が自ノード内のデータベースアクセスで済めばローカルに処理するが、他ノード内のデータベースアクセスが必要なときはネットワークポートを通して他ノードのDBMSとの間にネットワークパスをはり、そのパス上でDBSPに従ってデータのやりとりを行うことになる。

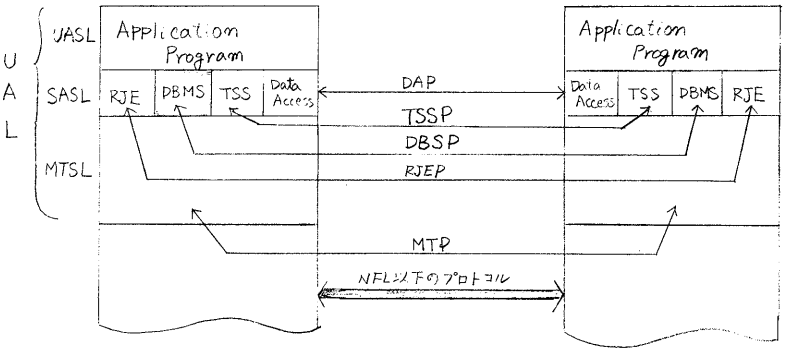

図2.1 DBMS および DBSP の位置

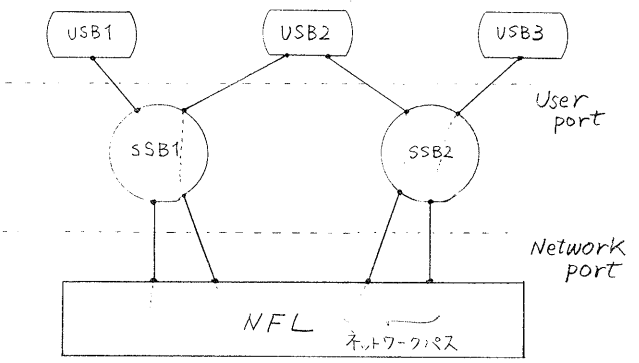

図2.2 ネットワークポートとユーザポート

3. 分散データベース管理システム　ここでは分散データベースの特長および問題点を述べ、DBMSが標準サブスクライバ(SSB)としてどの様な機能を持つべきか、分散管理に焦点をあてて述べる。

3.1 分散データベースの特長　集中型データベースに比べ、分散データベースは以下に示す特長を持っている。

(1) システムの信頼性向上　データベースシステムは集中型とは異り、ネットワーク上に散在する複数のノードとこれを結合するリンクから成っているため1つのノードまたはリンクの障害に対しても全体のシステムダウンとはならず、サービスが続行できる。

(2) 通信コストの低減　データベースアクセスのトラヒックの状況に応じてデータの配置が自由に行える。従ってアクセスの多いデータについては、そのアクセスがよく発生するノードに配置すれば良い。またトラヒックの状況に応じて同一のデータを複数のノードに配置してトラヒックの負荷を分散し、通信コストを低減できる。

(3) 拡張の容易性　従来の集中型データベースでは、容量の拡張はシステム

にとって重大な変更であった。これに対し分散データベースでは、ノードは通信リンクを介して結合しており、各ノードは独立性が強い。このため容量の拡張はノードの追加で対処でき集中型に比べ、このような変更に対し、よりフレキシブルである。

3.2 分散データベースの問題点　分散データベースは集中型では考えられなかった長所を持つものの、それを実現するためには以下に示す様々な問題を解決せねばならない[4]。

(1) 処理の同期　複数のサイトで同一データのアップデートが発生した場合、各サイトはお互い同期をとってデータを更新せねばならない。さもないと冗長データのconsistencyが維持できなくなってしまう。

(2) Queryの分散処理　Queryの実行に当り、その実行順序は実行時間に大きな影響を与える。すなわち、分散したサイトに対して処理をどの順序で行うかによって通信リンク上を流れるデータ量が大きく異り、それに従ってQueryの実行時間は大中に異る。データが各サイト間にいかに分散されているかを正確に把握することは処理効率を向上させるため重要な要因となる。

(3) Invisibilityの実現　各サイトのアプリケーションプログラム(AP)からネットワーク内の分散データベースの物理構造は見せる必要がない。すなわちAPは求めるデータがネットワーク内のどのサイトにあるか意識せずアクセスすることができる。これは分散データベースのディレクトリ管理にかかわる問題であり、また(1)処理の同期、(2)Queryの分散処理と大きな関係がある。

(4) サイトの障害対策　ネットワーク内のサイトの障害をいかに早期に発見するか、および障害修復後のサイトを全体のデータベースといかに矛盾なく組み込むかが問題となる。

(5) ファイルの割当　分散データベースを実現するに当り、実際発生するトラヒックが効率よく処理されるためには、どのようにデータをサイトに割り当てるかが問題となる。すなわち、処理上の通信量が最小となる様に、トラヒックのローカリティを考慮してデータをサイトに割り当てるという最適値問題である。

3.3 SSBとしてのDBMSの機能　分散データベースでは、集中型データベースとは異った点で問題が発生する。これはデータベースを分散化し、ネットワーク化したために発生するものであり、必ずしもデータベース独自の問題とは言えない。そこで分散データベースと集中型データベースは分離して考えることができる。すなわち、分散データベースで問題となる機能はDBMSの別モジュールとし、集中型データベース管理システムはDBMSのnucleusとしてインプリメント時には独立させることができる。分散管理モジュールの機能は3.2項で述べた内容であり、これを以下に示す。ただし、ファイルの割当はファイル設計時の問題であるためDBMSの機能とはならない。

- 処理の同期
- Queryの分散処理
- Invisibilityの実現
- サイトの障害管理

本稿ではこれらのうち、データベースモデルやアプリケーションとは比較的独立に論ずることができるものとして、処理の同期法、Invisibilityの実現法、を選び、主にこの2点について述べる。

4. Invisibility の実現方法

図4.1に分散型データベースシステムの概念図を示す。各サイトは、通信ネットワークを介して結合されており、ユーザもしくはアプリケーションプログラムは、自身の属するサイトのDBMSを通して、分散データベース全体にアクセスすることが可能である。

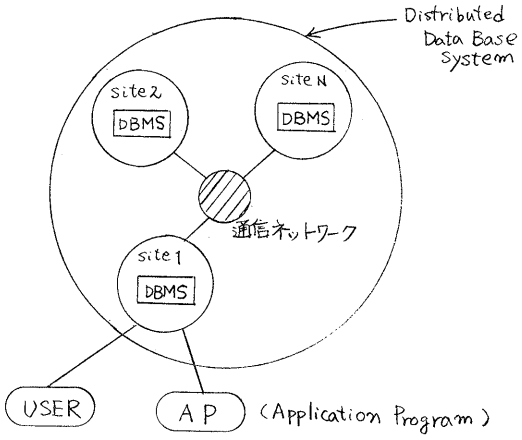

図4.1 分散データベースシステム概念図

このデータベース全体へのアクセス方法に関して、分散データベースシステムを特徴づける重要な概念が存在する。すなわち、Invisibility もしくはLocation transparency と呼ばれるものがそれであり、ユーザもしくはアプリケーションプログラムは、自身の必要とするデータが、どのサイトに格納されているかを知らなくともアクセス可能とする概念である。

このInvisibility を実現するためには、実データとそれを格納しているサイト名との関係を記述したディレクトリと呼ぶデータを保持する必要がある。このディレクトリの収容形態のちがいにより、種々のInvisibility実現の方式が考えられる。表4.1に代表的な方式を示す[4]。

分散データベースにおいて、データの冗長性は基本的条件である。すなわち、データの冗長がないというのは冗長型の特殊型として対処すべきである。この観点からディレクトリの管理方式も冗長型を基本とする。集中制御方式は分散データベースの諸問題解決には容易であるが、制御の集中化による信頼性の低下、トラヒックが集中した場合の処理能力上のボトルネック等の問題が発生する。すなわち、集中型は小規模低トラヒックには向くが一般的な分散データベースには向

表4.1 Directory 管理方式

方式		処理概要
冗長型	集中制御	自身のデータに関するDirectoryは自身で持つが，他のデータについては，特定センタで一括管理する。
	完全分散制御	各サイト毎に，全データベースのDirectoryを管理する。
非冗長型	組合せ制御	ネットワークをいくつかのゾーンに分割し，各ゾーン内では集中制御，ゾーン間はbroad castで制御する。
	集中制御	特定のセンタを設け，すべてのDirectoryは，そこで一括管理する。
	分散制御	自身のデータに関するDirectoryを，自身でのみ持つ。必要データの問合せは，broad Casting，又はChainmingにより処理する。

かないと思われる。一方完全分散制御方式は分散データベースの特長をすべて持っているものの、アップデートの割合が大きい場合およびノード数が多い場合はオーバヘッドが大きなものとなる。そこで本稿ではゾーン分割方式を採用し、かつゾーン間のディレクトリは冗長型とする案を提案する。すなわち、ゾーン内はセンタ集中型、ゾーン間は完全分散方式を採用する。ゾーン間のディレクトリとは例えばレコードのロケーション情報である。すなわち、レコードがどのゾーンにあるかを各ゾーンのセンタは保持している。この方式と、完全分散制御方式をディレクトリの通信量から評価する[5]。

パラメータを以下の様に定義する。

n　ノード数
m　ゾーン数
k　完全分散ノード間でアップデート時必要な通信回数
p　全トラヒック中のアップデートの比率
a　発生するトラヒックがゾーン内で処理できる割合

通信量の比較を表4.2に示す。*

表4.2　通信量の比較

比較項目	完全分散制御	ゾーン分割制御
Up-date の通信頻度（期待値）	$k(n-1)p$	$\{4a+(k(m-1)+4)(1-a)\}p$
Retrieve の通信頻度（期待値）	0	$(4-2a)(1-p)$
合計通信頻度（期待値）	T_d $= k(n-1)p$	T_i $=\{4a+(4(m-1)+4)(1-a)\}p+(4-2a)(1-p)$

ここで、ゾーンのローカリティを以下の様に定義する。

$$a = 1 - \frac{n - n/m}{n-1}(1-\alpha)$$

ただしαは1ノード当りのトラヒックのローカリティである。

図4.2に$n=10$, $k=5$ の場合の2案の比較を示す。曲線より上側がゾーン分割が有利な領域で下側が完全分散が有利な領域である。$n=10$ の場合はupdateが10%~20%以上あればゾーン分割方式が有利なことがわかる。またノード当りのローカリティが大きい程ゾーン分割方式が有利である。これよりゾーン分割方式は分散データベースのディレクトリ管理に有望な方法であると言える。

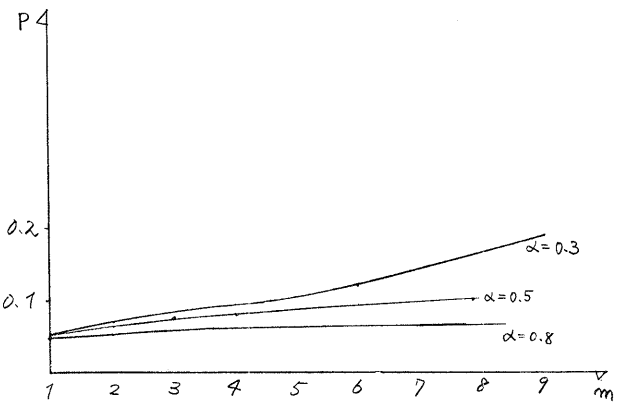

図4.2　2案の比較

*ただし、通信量の算出にあたり、集中型では右記のような制御が行われ、1件当りのupdateに要する通信量は4回として計算している。

update request ⟶
⟵ ACK
update data ⟶
⟵ ACK

5. 重複データのupdateおよびsynchronization

5.1 データベースのconsistency

前述のように、データアクセスのローカリティを利用して、効率のよい分散データベースシステムを構築した場合に、冗長データの存在は不可欠のものとなる。しかしながら、この冗長データはそのconsistencyを保つため、updateの同期を必要とする。

updateの同期を行うための最も一般的な方法はLockingである。そのうち、最も直接的かつ簡単な方法は、update sourceであるサイトが、broadcastによって全サイトをlockし、updateを行った後、lockを解除するものである[4]。従ってこのときにupdateのconflictが起っていれば、あるサイトはlock requestに対し、否定応答を返送することとなる。このとき、update, sourceが次のステップでとる方式の違いにより、いくつかのプロトコル形態が存在する。

その一の方法は、lock requestに対し、ひとつでも否定応答を返送したサイトがあれば、updateを行わないもので、本稿ではこれを完全一致方式と呼ぶ。updateのconflict生起時には、サイトに付加されたpriorityにより、どちらか一方のupdate requestは放棄されることになる。

その二の方法は、lock requestに対する過半数の肯定応答が得られればupdateを行うという方式で、多数決方式と呼ぶ[6],[7]。

一方、これらとは異なる方式として、各update transaction毎に、time stampを付加し、それによってupdate処理の実行順序の整合をとろうとする方式[8]、あるいはtime stampとlock方式の組合せ等種々の方式が考えられる。

各方式の詳述は行わないが、time stampを付加する方式では、各サイトが論理的な同期をとっておく必要があり、処理が複雑になることが予想され、また同期をとるための通信が新たに発生することも考えられ、通信コストも増加する。

一方、多数決方式では、サイトの位置によって多数の支持が得られ易いサイトとそうでないサイトとができてしまい、サービスのfairnessが失なわれる可能性もある。また、ネットワーク内の通信回線速度が場所によって著しく異なっていたり、ネットワークトポロジーが直線状であったりした場合には、データのconsistencyが失なわれる可能性すらあり、何らかの改良が必要となる。

さらに、これらとは全く異なるアプローチとして、すべてのupdate requestを統括管理するprimary siteを置く方式がある[9],[10]。集中処理形式としたために、この方式では分散データベースに特有のupdateのconflictを起させないという利点があるが、primary siteへのトラヒックの集中、primary siteの障害に対するリカバが困難等、分散処理の利点が生かせなくなる。さらには、updateの権利を持つprimary siteを順次、巡回させてゆくという方式もあるが、制御が複雑になり、また前述のprimary site方式の持つ欠点を補いきれない面もある。

表5.1に冗長データのupdate方式の類別を示す。

表5.1 冗長データupdate方式の類別

Centralized Approach	固定Primary方式
	巡回Primary方式(更新権回覧)
Decentralized Approach	完全一致方式
	多数決方式
	Time stamp方式
	Time stampと他方式との組合せ

更新アルゴリズムはなるべく簡単であること、またupdateの効率にも充分な配慮がなされねばならないこと等の観点から、上述の諸方式の利点、欠点を考慮し、本稿ではcentra-

lized と decentralized との並用案を提案する。つまり、Invisibility の実現方式で述べたと同様、ネットワークをいくつかのゾーンに分割し、ゾーン内では primary site を置く固定primary 方式を、各primary site 間では、conflict 生起時の制御アルゴリズムをできる限り簡単なものとするため、完全一致方式を採用する。従ってupdateの効率を向上させるためには、primary site 間で構成されるネットワークは、回線速度の面からも、またネットワークトポロジーの観点からも均質なネットワークにするのが望ましい。

次節では、各primary site 間で行われるupdate時のプロトコルを詳述する。なお、各ゾーン内のprimary site と local site 間で行われるupdateのプロトコルは通常の集中型で行われるため本稿では特に述べない。

5.2 データベースサービスプロトコル　前述の方式を採用した場合の、update時のデータベースサービスプロトコルの概略を図5.1 に示す。ただし図は、各ゾーンに置かれたprimary site のDBSPについてのみ示している。ローカルなDBSPは、図とは異なる。また、円内は状態を、多角形内は過渡的状態を示す。

updateが行われていないとき、データを管理するプロセスは、通常unlock状態にあり、このときデータアクセスが可能となっている。今、primary site に属するlocal siteから、冗長データのupdate要求があった場合を考察する。要求を受け付けたprimary site は、他のprimary site にリクエストを送出するため、broadcast状態に移行する。該当primary site から、他のすべてのprimary site へは、broadcasting により lock request が送出される。該当primary site は lock request をbroadcast 後応答待ちとなる。全primary site からAcknowledgement を受信した場合は、update data のbroadcast 状態に移行する。

あるサイトから否定応答が返送された場合は（updateのconflict が起っている場合）、再度broadcast を行う。あるサイトからの応答が返送されず、タイムアウトとなった場合は、障害サイトとみなし、エラーサイトリストにサイト番号を登録し、以降のbroadcasting の対象からはずす。Lock request broadcast 中に lock requestが受信された場合は（conflict生起）、request のソースサイト番号と自身の番号とを比較することにより（各サイトには、一意に番号が付されている）、priority判定を行う。もし自身のpriority がより高ければ、適当なタイミングをとって再度broadcast する。もし他サイトのpriority が高いのであれば lock release をbroadcastし、update放棄をソースである自身の管割下の local サイトに通知し、自身はunlock状態に戻る。

update data のbroadcast も同様に行われるが、このときには否定応答が返送されることはない（言い換えれば、この段階でconflictが生起することはない）。

また、エラーサイトリストがあれば、それをupdate data と伴にbroadcast し、それを受信した各サイトは、エラーリスト中のサイトをその処理対象からはずす。

update data をbroadcast すると同時に、該当サイトは自身のデータをもupdate する。全サイトから肯定応答を受信した場合、lock release をbroadcast して、updating を終了する。

一方、unlock状態で他サイトからlock requestを受信したサイトは、Acknowledgement を送信して、update data 受信待ち状態となる。このときは、プロセスは、lockされた状態であるので、lock requestを送出してきたサイトからのupdate data 以外のデータに対しては、すべて否定応答を返送する。また、この状態のまま、許容時間を越えて待ち状態におかれた場合は、request のソースであ

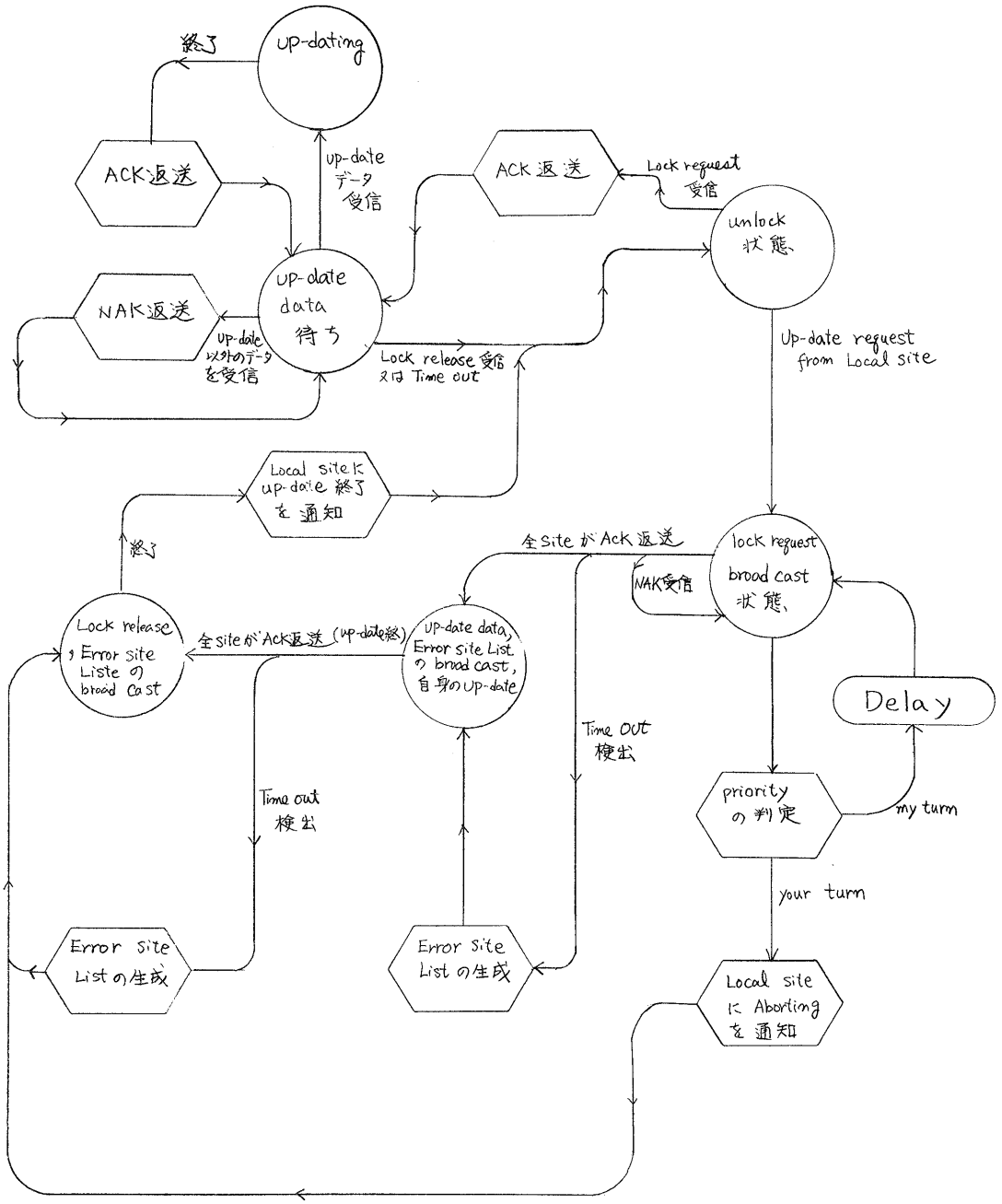

図5.1　up-date 時の正常および準正常シーケンス

るサイトの障害とみて、自らはunlock状態に復帰する。

update data の送信に対して、プロセスはデータのupdateを行い、肯定応答と返送し、ソースからのlock releaseにより、unlock状態に返る。

6. おわりに　本稿では、ネットワーク・アーキテクチャ上に、分散データベースシステムを構築する場合の、具体的方式の提案、発生する諸問題、およびデータベースサービスプロトコルの例を、updatingの場合について提示した。

なお、本稿では触れることができなかったが、分散データベースの構築に伴なう問題点として、今後の検討が必要であるいくつかの重要な事項がある。

その第一のものとして、分散ネットワークのpartitioningとそのdetection およびrecoveryがあげられる。つまり、分散ネットワークが障害等により、2つ以上の部分に分割（パーティション）されてしまう現象である。このとき各部分ネットワーク内で、それぞれ独立にupdateが行われたときには、冗長データのconsistencyを保持することが不可能となる。また、障害復旧により、各部分ネットワークが再度統合されたとき、どのようにして、consistentではなくなってしまっている冗長データを再統合するかが、困難な問題として残る。

例えば、主要都市間を結合するような分散データベースシステムを構築した場合、特に我国のような環境では、直線状の、partitioningの生起し易いネットワークトポロジーになることが容易に予想されるため、充分に配慮がなされなくてはならない事項であろう。

第2に、本提案のupdate protocolでは、各サイト対応にpriorityを付加しているが、この場合、conflictの生起時に、優遇されるサイトとそうでないサイトとが、固定化されてしまう可能性がある。サービスのfairnessという観点から、このことを避けるため、priorityが順次移行してゆく方式や、time stampとサイト番号とにより、priorityを決定する方式も考えられる。さらにまた、priorityは、ネットワークトポロジーにも依存しなければならない。たとえば、直線状のトポロジーを考えてみた場合、明らかに、update時に、conflictの生起し易いサイト（両端）とそうでないサイト（中央）とができてしまうため、両端に近いサイトほど高い優先度を付加しないとサービスのfairnessが保てなくなる。このように、priorityの割り当て方法は、サービスのfairnessという観点から、充分に配慮されなくてはならない事項である。

第3に、本稿では、updateを行う際のlockingの対象範囲については、述べられなかったが、この範囲をたとえば、データベース全体について行うのか、データベースの部分集合について行うのか、あるいは特定のレコード、フィールド等について行うのか等は、updateの頻度、システムのアベイラビリティの基準、処理効率等を充分考慮した上で決定しなくてはならない[11]。

［参考文献］

1. 杉浦,西沢,松下,関 "An effective utilization method of public PSN in DONA" ICCC 78
2. 松下,他 "An overall network architecture suitable for implementation with either Datagram or Virtual circuits facilities" *Computer Communication Review*, 1978
3. 松下,他「DONAにおける仮想端末・ユーザ/アプリケーション層の設計思想」電子通信学会 電子計算機研究会 EC 78-8 (1978. 5月)

4. Rothnie J., Goodman N. "A survey of research and development in distributed database management" Proc. Very Large Data Base 1977
5. 山崎, 疋田, 他「分散データベースにおけるInvisibility 実現方法についての一考察」昭和53年度情報処理学会第19回全国大会
6. Thomas R.H., et. al. "A solution to the concurrency control problem for multiple copy databases" COMPCOM, spring, 1978
7. Thomas R.H. "A solution to the update problem for multiple copy databases which uses distributed control" BBN report No. 3340, July 1975
8. Ellis C.A. "A robust algorithm for up-dating duplicated databases" Proc. of the second Berkeley workshop on distributed data management and computer networks, May 1977
9. Alsberg P.A., Day J.D. "A principle for resilient sharing of distributed resources" Report from the Center for advanced computation, University of Illinois, 1976
10. Stonebraker M., Neuhold E. "A distributed version of INGRES" Proc. of the second Berkeley workshop on distributed data management and computer networks, May 1977
11. Ries D.R., Stonebraker M. "Effects of locking granularity in a database management system" ACM Trans. on Database Systems, Vol. 2, No. 3, Sept. 1977