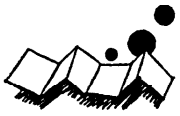
解　説

2. 基　礎　技　術



2.1 分散処理技術の基本的課題†

上　林　弥　彦††

1. まえがき

分散処理技術の応用は非常に広範囲にわたるため，計算機の研究・開発における一つの重点課題となっている．しかし，従来の集中型システムにみられない多くの問題があり，さらに分散処理の利点を生かそうとすると解決しなければならない問題が生じる．本稿では，分散システムを扱う場合に特に重要となる問題である構成要素の多様性から生じる不均一性，複数の計算機を同時に動かして効率を向上させるための分散並行処理，およびシステムの柔軟性や故障に対する処理に対応する動的再構成を中心に問題点をまとめている．

分散処理システムは，結合の強さや広がり，ネットワークの性質，構成要素の種類によって次のように分けることができ，それ自体が非常に多種多様である．

〔結合の強さによる分類〕

（1） 並列処理：これには主記憶を共有した強結合のものと，各構成要素のシステムが独立の主記憶を持つ弱結合のものとがある．一つの処理が複数個の構成要素の中に分散され並列に実行されるが，システム自体は分散していない．

（2） 地理的分散：各構成要素のシステムの独立性の強いシステムと，全体として一つの集中システムに近い形となるような独立性の弱いシステムとがある．

〔分散のしかたによる分類〕

（1） 分散の広がり：一つの室や一つの建物の中などに作られた局所分散システムと広い範囲にひろがった広域分散システムとがある．

（2） 水平分散と垂直分散：各部分システムが比較的対等に結合されるのが水平分散である．垂直分散には中央システム，ワークステーション，端末の階層や局所分散システムを結合して広域分散システムを作った場合の階層がある．

〔ネットワークの種類〕

（1） 通信の性質による分類：1対1通信，放送通信（バス線や衛星通信），送信順に着くか，ある有限時間内に着くか

（2） 通信速度による分類：電話回線から光ファイバによる超高速回線まで

（3） 信頼性による分類：通信線上の信号の誤り率の大きさや通信線が不通になる確率による分類

〔構成要素の種類〕

（1） 均一型システム：同質のシステムのみで構成されるもの

（2） 不均一型システム：種々の性質・能力の異なるシステムで構成されるもの

(2-1) （一般の）不均一型システム：各構成要素システムの機能にはかなり重複があるが実現方法が異なる．

(2-2) 機能分散型システム：各構成要素は異なる機能に特化している．

また，分散処理の利点も以下に示すように多様性があり，システムの目的によって特定の利点を目標とした構成が行われる．

〔分散処理の利点〕

（1） 均一性：分散システムのどの端局からでも同じサービスを受けることが可能となる．

（2） 即時性：ある端局での変更はただちに他の局で認識できる．

（3） ハードウェアの経済性：大量生産されたシステムの価格は性能よりも生産量の影響によって決まる要因が強く，このようなシステムを複数台結合する方が，目的によっては大規模システムを用いるより経済的になる．

（4） ソフトウェアの経済性：大規模システムのためのソフトウェアよりも，小規模システムのためのソフトウェアを種々開発して，小規模システムによる分

† Fundamental Problems of Distributed Processing Technologies by Yahiko KAMBAYASHI (Department of Computer Science and Communication Engineering, Faculty of Engineering, Kyushu University).

†† 九州大学工学部情報工学科

散システムを開発し，全体として必要な機能を実現する方が開発が楽でありソフトウェアの変更も容易となることが多い．

（5） 処理の経済性：データなどの資源はよく利用される局の近くに置けば処理コストを下げることができる．

（6） システムの拡張性：すでに存在しているシステムを統合して大規模システムを構築してゆくことができる．

（7） 変更容易性：分散システムを各構成要素の独立性が高いような構成にしておけば，各構成要素のハードウェアやソフトウェアを変更するときにシステム全体に及ぶような影響を避けうる．

（8） 保密性：データの重要度に応じて，ソフトウェア的な方法の他にハードウェア的な方法（記録する局を分けるなど）も併用でき，重要なデータに対しては強力な保護が行える．

（9） 高可用性：システムの一部が故障しても故障に関係のない処理は続行できる．さらにハードウェアやデータなどを重複させることにより利用者に故障を気付かせないようにできる．また修理中もシステムが止まらないことが望ましい．

(10) 高速性：現在最高速の計算機用素子の速度をたとえば数百倍程度にするには非常に大変である．このため，回路の集積化のほかに，パイプライン処理や分散処理といった並列処理技術による高速化が今後ますます重要となってくる．

(11) 遠隔操作性：遠隔故障診断など専門家を必要とする処理の場合でも専門家の移動を必要としない．

しかし，これらの利点を生かしたシステムを開発するために解決するべき問題点も非常に多い．これらは大きく分けて次のようにまとめられる．

a．システムの構成要素（ソフトウェア，ハードウェア，通信）の問題

b．システムが広域にわたるために生じる問題

c．システムの構成要素が多様であることにより生じる問題

d．上で述べた分散処理の特色を出すために解決するべき問題

e．構成要素が多様であり，また目標も多様であるため生じる，種々のトレードオフの問題

f．大規模分散システムが使われるようになるための環境の整備に関連した問題（非技術的問題を含む）

対応する主な課題は次のようになる．

（a-1） 分散処理のためのソフトウェアおよびハードウェア開発（通信ソフトウェア，分散オペレーティングシステム，分散データベースなどのソフトウェアやそれらを効率良く働かせるハードウェア，並列性の高いハードウェアの開発）

（a-2） 種々の特性を持った通信ネットワークの実現．とくに信頼性，高速性が重要．通信のバックアップ方式．

（b-1） システムの同期問題

（b-2） データ名，利用者アドレス，システム名の管理（異なるシステム内では同じデータが異なる名前で呼ばれることがあり，同じデータであることを知らなければならない．利用者アドレスやその経路も一意でない．言語の異なる複数の国にまたがる場合にはより一般的な対応の管理が必要である）．

（b-3） 時差に対する対策（原則として 24 時間運転．地域的に負荷の大きな部分が移ってゆくことに対する対策）．

（c-1） 各種の標準化（システム間の通信や使い方の標準化により，多様性を解消しようとする方法）．

（c-2） 特別な仕事をするシステム（サーバと呼ばれる）の設計とそれらのシステムの統合（各種サーバとして，ファイル管理を専門に行うファイルサーバ，データベース管理機能に特化したデータベースサーバ，システム全体の管理を行うシステム管理サーバ，ほかのシステムとの通信を行う通信サーバなどが考えられる）．

（c-3） 既存のシステムに対する統合方式の開発（データモデルやデータベース言語の異なる場合に相互変換を必要とする．異なる並列処理制御方式が混在している場合や異なる回復処理方式が混在しているような場合はそのままでは容易に統合できない）．

（d-1） 分散並行処理制御（同時実行制御）方式（分散処理を行う場合，処理能力を持つ構成要素が複数個あり，できる限りそれらをすべて働かせることにより効率を向上させる必要がある．このようなことを実現しようとするのが並行処理制御である．各要素の速度に差があり，かつ通信線による結合を考えて同期の問題を扱わなければならないのであまり簡単ではない）．

（d-2） 分散質問処理（データのコピーのある場合のデータベース質問処理などが代表的である）．

（d-3） 分散回復処理（システムの故障に対する回復処理方式．故障の影響を受ける処理の検出が重要．故障についても，各局の中の部品単位のものから，天

災によって複数の局が破壊されるような場合まで種々の水準がある).

(d-4) 資源配置問題(データや処理単位をどの局に割り当てると処理コストを減らすことができるかという問題.分散システムは変更が容易という性質を持つため,システムの変更にともなって動的に変化できることも重要である).

(d-5) 保密性の確保(システムをハードウェア的に分離することによって保密性の保証が可能となる面の逆に,分散システムではシステム利用者の範囲が非常にひろがるため,保密性には特に注意を要する).

(d-6) 分散アルゴリズムや分散プログラムの開発(できる限り局所性を重視したものである必要があるが,正しさの判定などには困難な問題も多い).

(e-1) 分散処理の目標のうちどれを重視するか.処理速度と処理コストのトレードオフ.標準化と多様性との間のトレードオフ.

(e-2) 分散システム設計用システムによって複雑な問題の扱いを簡単にする.

fは技術以外の問題とのトレードオフで,技術以外の考え方を要する場合がある.4つの例をあげる.

(f-1) 標準化は種々のシステムを統合するためには不可欠な非常に重要な課題であるといえる.しかし,次のような問題を考える必要がある.標準化によって新しい技術開発がおさえられる可能性もあり,多様な技術開発の要素を残した標準化を考えなければならない.周辺技術(たとえばLSI技術)の発達により新しい標準化を必要とすることも多く,寿命の永い標準化が重要といえる.市場専有率の高いメーカにとっては標準化は自社製品の優位を失わせる可能性があるため協力しにくいこともある.多くの国が協力して標準を決める場合には技術水準の高い国でしか実現できないような進んだ方式が標準に採用されるとは考えられない.このように標準化には政治的要素も強い.

(f-2) 広域の分散システムとして実用化しているのは,銀行や証券システム,座席予約システムなどの均一性の高いものか,利用法としては構成要素のうちの一つのシステムのみしか使えない独立性の高い分散システムかである.不均一な分散処理システムの例としては,全国的な分散データベースシステムの構築が考えられる.このようなものができ,地方にいても大都市と同時に同質の情報をあまり変わらないコストで得ることができるようになれば,大都市への企業や人間の過度な集中をおさえることができる可能性がある.このようなシステムは,高度情報化社会の基本となるべきものであるため,大型プロジェクトによる研究開発(通産省の電子計算機相互運用データベースシステムの開発プロジェクト[10])とともに,政策的な実用化も重要である.しかし,どこでも同じ情報が利用できれば,逆に地方の特色をなくしてしまう可能性も持つ.地方の情報の大都市からの利用を抑制するか利用コストを上げるなど,政治的判断も必要である.

(f-3) 分散システムの保密性を実現しハッカなどからの被害を少なくするためには,どのような利用者がどのようにシステムを利用したかの記録が残ることも重要である.しかしこのような記録は個人のプライバシやシステムを利用している企業の秘密に関係することがあり,悪用できないようにしなければならない.このためには,利用状況記録の利用に関する政策的な判断や不正利用時の罰則などを決める必要がある.

(f-4) 現行のほとんどの政策や法律がこのような分散システムの存在を仮定していないため,このようなシステムに応じた法律の整備や経済運営は特に重要である.

分散処理システムがわれわれの生活と密接にかかわるようになればなるほど,保密性,プライバシ保護,システム故障時の補償など,純粋に技術だけの問題では解決しない問題がいろいろ出てくると考えられ,その中で新しい技術的問題が生じる可能性も大きい.

本稿では分散処理にとって特に重要な技術的問題に絞ってまとめる.

2. 物理的分散と論理的分散

多様な構成要素からなり時間的に変化する可能性のある分散処理システムを使いこなすことは容易ではない.したがってシステム自体はこの多様性を生かしたものであったとしても,利用者にとってはできる限り均一で単純なものが望まれる.すなわち,実際のシステムの分散状況(物理的分散)と利用者から見たシステムの分散状況(論理的分散)が異なるようになることが要求される.

分散には,水平分散(図-1(a)に示すような網構造で示される)と垂直分散(図-1(b)に示すような木構造で示される)の2種が考えられる.水平分散の場合は,分散している資源どうしははぼ対等であるが,垂直分散の場合は上から下への従属性が仮定されることが多い.データを求めるのに上の方から順次下の方へ下る場合や,公共データ,グループデータ,私的デー

タといった階層性（公共データとその下のグループデータは矛盾のないこと，ある利用者の私的データとその上のグループデータ，公共データの間に矛盾のないことなどが要求されるが，同じグループに属する二つの私的データ間には矛盾があってもよい）が考えられる．もう一つの代表的な階層は処理条件による階層である（同じシステム内の資源，同じ室にあるシステム内の資源，同じビルにあるシステム内の資源といった階層を考えた場合，前のものほど通信コストが少なくなる）．

実際の分散はこの二つの水平分散と垂直分散とが混在したものと考えられ（図-1（c）に例を示す），一つの資源集合に対して複数個の垂直分散のある場合も考えられる．実際に使われるシステムにおける物理的分散と論理的分散の例には次のようなものがある．

（1） 一番単純なものは，利用者が使いなれているシステムと同じようにして，分散システム内の任意のシステムが使えるものである．実現方法としては，ワークステーションを用いたものが一般的で，利用者の使えるシステムの利用者インタフェースの変換を行う．目的システムが利用者の使えるシステムにない機能を持っているときの問題を扱う必要のある点や分散システムの構成要素のシステムが協調して行うような処理を実現できないという問題がある．

（2） 上記のシステムの論理的構成は図-2（b）に示すようになる．このシステムは利用者はどの構成システムでも利用できるが個々のシステムの独立性は高く，複数個のシステムによる協調処理の実現は困難といえる．システム自体の物理的構成は単純化のため図-2（c）のようになることが多い．

（3） システム全体を一つの集中システム（図-2（a））という形で利用者が扱えるようにするのが，使いやすい分散システムの構成法であると考えられている．このように利用者に分散を意識させないとき，分散に対する透明性（不可視性）を持つと呼ぶ[8]．利用者は，データなどの資源がシステムのどこにあるかを知らずに，自分の使っている局にすべての資源があると仮定して使うことができる．

（4） システムの構成要素の多様性を利用したい場合や，処理コストや処理時間を考えながら処理の内容を決めたい場合には，分散性がある程度利用者に分かる必要がある．このため，実際のシステムを単純化した論理的分散システムを利用者が扱うようにできるものも考えられる．

（5） 逆に物理的には分散していないのに，論理的には分散しているように扱う必要のあることがある．

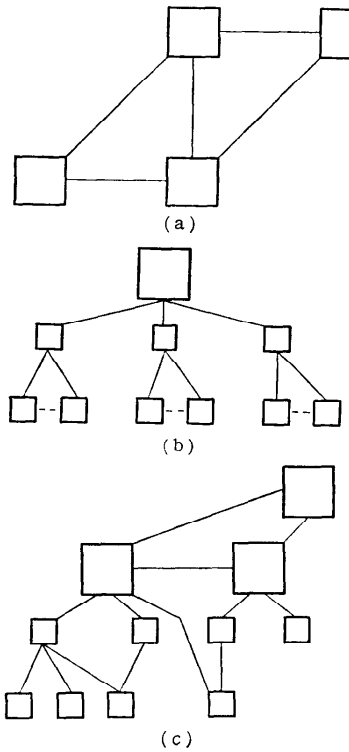


図-1　水平分散と垂直分散

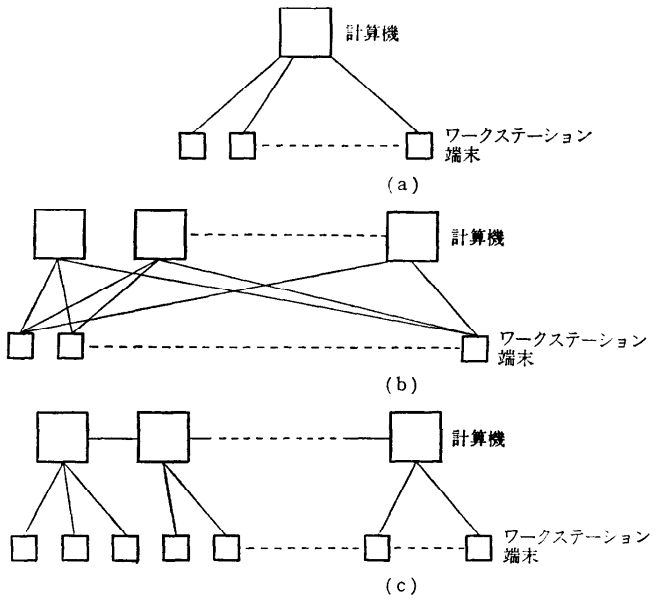


図-2　論理的分散と物理的分散

たとえば，上記(3)の場合に，ある種のデータがある部分では分散しており，他の部分では集中化しているとき，同じように分散していると考えた方が扱いやすいことがありうる．また，システムが動的に変化してゆく場合，分散していた部分が統合されても，利用者にとっては過去のソフトウェアが使えるなどの理由により分散しているように見える方が都合の良いこともある．保密性を保つ意味から，論理的にデータの階層性を仮定することがある．このように実際に存在しない分散が論理的に実現されるものを仮想分散と呼ぶ．

以上を要約すると，分散処理システムにはシステム自体の物理的分散のほかに，各利用者から見えるシステム（論理的分散）があり，それらは異なっていてよい．また，利用者によって異なる論理的分散になることも許されている．このように異なる分散形態を許すことにより，使いやすさを実現できると考えられるが，次のような問題を解決しなければならない．

（1） 異なる分散形態間の対応をどのようにして実現するか．

（2） システム自体が動的に変化しうるし，利用者も自分のみたいシステムに対する要求が変化することがある．このように分散形態の変化にどのように対処するか．

（3） 水平分散と垂直分散の複合をどのように扱うか．

分散処理システムにおける透明性は，システムの持つ種々の性質を利用者に気付かせないようにするためのもので，物理的分散と論理的分散を独立させるための有効な概念といえる．上記の分散に対する透明性の他に次のようなものが考えられる．

（a） 資源位置に関する透明性：利用者は資源の場所を知らなくてよい．

（b） 資源重複に関する透明性：とくにデータにコピーのある場合，どのデータにコピーがあるかを知らなくてよい．

（c） 処理方式に関する透明性：システム内でどのような処理方式をとっているかを利用者は知らなくてもよい．

（d） 並行処理制御に関する透明性：並行処理制御やその結果としてすくみ（デッドロック）や後退復帰（ロールバック）がおこっても利用者は知らなくてよい．

（e） 故障に対する透明性：故障がおこっても利用者は対処する必要がなく，処理を実行できる．

（f） 故障回復処理に対する透明性：故障回復処理中にも利用者は知る必要がない．

しかし，透明性は分散システムを一つの集中システムとして用いるためのものであるため，分散を有効に利用した使い方をする場合と分けなければならない．

このように論理的分散と物理的分散の異なる場合，特に問題となるのがコスト関数である．たとえば，異なる分散の間の機能の変換が可能であったとしても，処理時間がかかりすぎれば実用的とはいえない．

分散の良さや変換方式を評価するためのコストとしては次のようなものがある．

（1） 通信量最小：広域分散処理システムでは通信コストが全体の処理コストを決定する主要因となるので通信量を減らすことが重要である．

（2） 応答時間最小：並列処理をできる限り行って応答時間を最小にすることが重要な場合も多い．並列性を上げてもそれによって後退復帰が増加すると平均的な応答時間が減らないこともありうる．

（3） 計算量最小：局所的分散処理システムでは集中型と同じく計算量が処理コストを決める主原因となることもある．

（4） その他：信頼性や保密性の確保を最重点として，その良さでコストを決めるものもある．処理が複数個同時に実行されるため，上記のコストも単一処理の場合と異なる扱いを必要とすることもある．分散処理システムの目的によって種々のコストを用いうる．

集中型に比べて，種々のコスト評価が複雑にからみあっており，システムの資源配置，処理要求に対する処理方針の作成，並行処理制御などの方式が，コスト関数によって異なったものが選ばれる可能性がある．また，システムの利用状況も時間によって変化してゆくもので，それにともなうコスト関数の変化を扱うためには，上記の問題（資源配置，処理方針，並行処理）が動的に対処できることが望ましい．

3. 異なるシステムの統合問題

分散システムの構成方法には，全体を設計してから部分システムの詳細設計を行うトップダウン方式と，個々のシステム構成からはじめて全体を統合してゆくボトムアップ方式とがある．後者は既存のシステムを統合してゆく場合に特に有用といえる．実際のシステム設計の場合は，この二つの方法を併用することになるが，ここでは問題点の多い後者の方法に絞って考察する．

システムが異なった方式を採用している部分から構成されているとき，次のような場合がある．

（1） 特に統合しなくても問題が生じない場合．

（2） 内部の方式が異なっていても，外部インタフェースを統一できる場合．この場合この統一インタフェースを用いて統合できる．

（3） 同じ外部インタフェースにそのままでは変換できない場合．

（1）の例として，すくみ（デッドロック）検出法をあげることができる．代表的な方法には，すくみ検出グラフによる方法と時間切れによる方法とがある．前者は二つの処理 T_i と T_j において，T_i が T_j の終了を待っているとき，節点 T_i から節点 T_j へ向かう有向枝を付加するとして定義された有向グラフを用いるもので，有向閉路ができるとすくみが存在することがわかる．**図-3** において，T_1 T_2 T_3 の部分で閉路ができているため，これら三つの処理は止まっていることになる．時間切れによる方法は，ある処理がある時間 t 以上止まっているとすくみが発生したと判定するもので，すべてのすくみは発見できるが，すくみになっていないものも待ち時間が長いとすくみと判定する可能性がある．t を短くすると，実際にすくみが生じてから検出するまでの時間を減らすことができるが，すくみでないものもすくみとして判断する可能性が高くなる．この方式は単純であるが，すくみが生じたときに後退復帰（ロールバック）して再実行をするのは一般に最初に発見された処理である．すくみ検出グラフを用いると，再実行コストを考えた上で有向閉路の中の一つの処理を選定できるので，再実行コストを減らす上で有利である．このすくみ検出グラフを用いているシステムと時間切れ方式を用いているシステムの二つを統合して全体としてのすくみ検出をどのようにしたら行えるかについて考えてみる．

まず，おのおののシステム内のすくみはおのおのの方式で検出できる．したがって問題となるのは両方のシステムの処理が互いに関係したようなすくみである．この場合，必ず時間切れ方式を用いているシステム内の処理が存在するため，このシステムによって時間切れとして発見できる．すなわち，統合したすくみ検出を行わなくても，すべてのすくみを検出することができる．

（2）の場合は，特に標準化は有効な手段といえる．

(2-1) 通信方式，通信プロトコルの標準化

(2-2) データ表現形式の統合

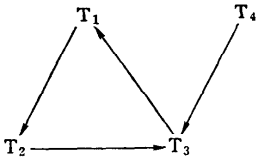


図-3 すくみ検出グラフ

(2-3) 各システムのシステムコマンドの統一

(2-4) データベース言語の統合

(2-5) データベースモデルの統合

(2-1)，(2-2)の例としては，それぞれ OSI 規格や文字コード（ASCII と EBCDEC 間の変換，各種漢字コード間の変換）がある．機能の同じコマンドに対しては同じ標準名を与えることにより，(2-3)は扱える．

(2-4)の例としては，関係データベースシステムの利用者言語の相互変換をとりあげる．代表的な言語としては，関係代数言語，QUEL のような組関係論理言語，SQL のような写像型言語，QBE のような定義域関係言語があり，これらの間の相互変換についての研究がある．たとえば，共通のインタフェースとして，手続き型に近い関係代数言語や関係データベース言語の標準となりつつある SQL（標準化が行われつつある）を用いればよい．

(2-5)のデータモデルの統合についても，多くの研究が知られている．代表的なデータモデルには，階層モデル，網モデル，関係モデルがあり，それぞれ異なる特色を持っている．関係モデルでは，ひとまとまりのデータを一つの表（関係と呼ぶ）にまとめ，この関係の集合によってデータの集合を表現する．この関係の間にさらに木構造の関連を与えたのが階層モデルで，この関連の構造を木構造より一般的なグラフの構造にしたのが網モデルである．これらの構造を示す関連もまた関係の形で表現できるため，階層モデルや網モデルの外部インタフェースとして関係モデルをもってくることができる．このため，これらのモデルが混在した分散データベースでは，すべてのデータベースを統合するためのモデルとして関係モデルをもってくるのがふつうである．データベースの設計には，データ集合の満足する意味制約が使われる．関係モデルの設計では特定の質問集合に対する効率向上はファイル設計にまかされているが，ほかのモデルではそうではなく，代表的な質問集合に対して処理効率を高めるように構造が決められる．したがって，これらのモデルの外部インタフェースを関係モデルとすると，これらの質問処理効率に関する情報が扱えなくなるおそれが

ある．

分散処理において，資源の利用によって決められた処理単位間の順序が，すべての処理で矛盾しないようにしなければならない．資源の利用には次のような種類があるが，それによって順序が生じるのは次の(c)の場合である．

(a) 計算や入出力のために資源を利用する

(b) データの読み出しのみを行う

(c) あるデータに対して読み出し操作と書き込み操作とがある．

できる限り資源を並列的に動かしてシステムの効率を上げ，かつこのような順序の矛盾を避けるのが並行処理（同時実行）制御である．上記の(c)の条件からデータベースの場合に特に重要といえる．

(3)の問題の代表例は，並行処理制御の統合である．データベースの並行処理制御として代表的なものに，2相施錠方式と時刻印方式がある．2相施錠方式は，各処理単位が必要なデータを順次施錠してゆき必要なデータがすべて揃うまではどのデータに対する解錠もしない方法である[4]．あるデータを使用するときは必ず施錠してから使うものとし，他の処理単位によって施錠されているデータを使いたいときは解錠されるまで待たなければならない．この方法の問題点はすくみの起こる可能性のあることで，すくみが起こるとその原因となった処理単位の一つを後退復帰させて再実行する．図-3 の例では，T_1, T_2, T_3 のうちの一つが後退復帰されることになる．時刻印方式では，各処理単位に異なる自然数を対応させこれを時刻印と呼ぶ[1]．自然数の大きくなる順の順序を考え各データに対する処理単位の処理順はこの時刻印の順と矛盾しないことが要求される．すなわち，データAに対してまず時刻印 50 の処理単位が処理し次に時刻印 65 の処理単位が処理することは許すが，その逆にまず時刻印 65 の処理単位が処理したあと時刻印 50 の処理単位が処理を要求するとこの要求は実行しないようにする．時刻印 50 の処理単位は後退復帰して新しい時刻印（たとえば 80）をもらって再実行する．こんどは時刻印が大きいためデータAを読んでも矛盾しない．

並行処理制御が正しく行われるというのは，処理の結果が一つの集中システム内で一つ一つの処理を順序良く処理した場合（直列処理）と同じになるものをいう．実際には，分散システムでは複数の局があるため，効率を上げるには，複数個の処理単位が同時に実行されることになる．このように並行処理の実行結果が直列処理と等しくなるとき，並行処理制御によって生成されたスケジュールは直列可能であるという．2相施錠方式では，必要なデータをすべて施錠し終えた時間の順に処理単位を並べると，これらが対応する直列順になる．時刻印方式では実行の終わった処理単位に対しては時刻印順が直列順となる．この二つの方式で共通しているのは後退復帰の機構だけといえる．2相施錠方式では処理単位に時刻印を付ける能力はなく，時刻印方式は施錠や解錠の機能を持たない．このため，この二つの方法を共通のインタフェースで統合することは不可能といえる．このため，統合できるように，これらの方式を変更する必要がある．

つぎに，分散処理システムではかなり単純な場合でも並行処理制御が必要なことを示す．データに対する読み書きに注目した場合次のような種類が考えられる．ここで，複数の局にわたる処理を広域的処理，一つの局のみの処理を単一局処理と呼ぶ．

(1) すべての処理は読み出し操作と書き込み操作を含むが単一局処理のみである（図-2(b)参照）．

(2) 広域的処理はすべて読み出し操作しか許されていない．各局のデータの変更は，夜間などの利用者の処理が行われていないときに行う．

(3) 広域的処理はすべて読み出し操作しか許されていない．書き込み操作を含む処理はすべて単一局処理であるが，広域的処理と単一局処理をまぜて並行的に処理を行う．

(4) 広域的処理にも読み出し操作のほかに書き込み操作を許す．

上記(1)や(2)の形の分散処理システムが実現しやすいのは，特別な並行処理制御を必要としないためである．

同一のデータに対して異なる二つの処理単位から書き込み要求と読み出し要求のある場合，この実行順によって処理単位の直列実行順が決まる．このような場合を読み書き矛盾があるという．(3)のように広域的処理を読み出し操作のみに限っても局所処理と読み書き矛盾が生じるため並行処理制御が必要であることは次のようにして示される．

局 S_1 と S_2 を考える．S_1 にはデータ A があり S_2 にはデータ B がある．処理としては次の 4 つを考える．

T_1：まずAを読んでからBを読む

T_2：まずBを読んでからAを読む

T_3：Aの値を変更する

T_4：Bの値を変更する

S_1ではT_1 T_3 T_2の順で実行され，S_2ではT_2 T_4 T_1の順で実行されたと考える．図-4に示すようにこのようなことは可能である．この場合S_1（したがってA）によって決まる順序はT_1のあとにT_2であるが，S_2（したがってB）によって決まる順序はT_2のあとにT_1となり矛盾する．すなわちなんらかの制御によってこのようなことを避けなければならない．

T_2がなければこのT_1の実行は正しいとみなされ，T_1がなければT_2の実行は正しいとみなされる．しかし，T_1とT_2とを同時に実行するとそれぞれが得た処理結果の中に矛盾するものが含まれる可能性がある．

このような分散システム全体にわたる並行処理制御機構を実現する場合，各構成システムには次の三つの場合がある．

（a）　並行処理制御機構を持たない．

（b）　集中システム用の並行処理制御機構を持つ．

（c）　分散システム用の並行処理制御機構を持つ．

分散システムの構成要素がすべて全く同じ分散システム用の並行処理制御機構を持たない限り分散システム全体にわたる並行処理制御を容易に実現することはできない．一般的には，システム内には上記の3種のシステムが混ざる上，並行処理制御機構としても，2相施錠方式，時刻印方式を含めて実用的なものがいくつかあり，それらも混在する可能性がある．重要な場合として次の三つの場合がある．

（1）　各構成システムが並行処理機構を持たない場合

（2）　各構成システムが同種の並行処理機構を持つ場合

（3）　構成システムの持つ並行処理機構に異種のものが含まれる場合

（1）の場合は，各システムは処理単位の与えられた順に実行するため，並行処理制御機構によって順序が変えられないため，かえって扱いやすい．文献[6]にこのような場合の局所分散システムの構成が示されている．

特に問題となるのは（3）の場合で，2相施錠方式と時刻印方式の対応が重要といえる．まず考えられるのは，シミュレーションによる方法である．2相施錠方式には時刻印がないので時刻印に相当するデータを内部に生成する必要がある．時刻印に矛盾する順序になる場合は後退復帰させなければならず，このためには時刻印に矛盾するとすくみが生じるようにしなければならない．逆に時刻印方式で2相施錠方式をシミュレートするには，施錠にあたる操作を疑似的に実現しなければならない．施錠機構がないため，施錠を示すビットを用いて制御することになる．どちらのシミュレーションも非常に効率が悪い．

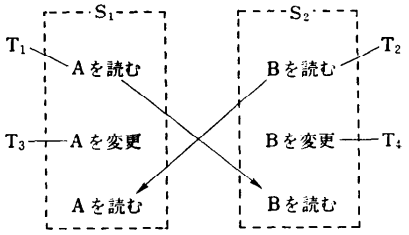


図-4　分散並行処理の必要性

このための別の方法として，二つの並行処理制御方式をそれぞれ変更して，統合しやすくする方法が考えられる．このために，並行処理制御の可観測性と可制御性という概念が導入された[7]．

処理単位$T_1, \cdots, T_m$に対するスケジュールSによって満足される$T_1, \cdots, T_m$上の半順序P_sは次のように定義される．T_aとT_bの間に読み書き矛盾があるとP_sは$T_a > T_b$か$T_b > T_a$のいずれかを含む．これはSによって決まるものである．

並行処理制御機構は半順序Pを与えるとPを満足するスケジュールSを生成できるときに可制御であるという．

並行処理制御機構は，実行スケジュールに対応する直列順（ないしは半順序）が外部から観測できるとき可観測であるという．

時刻印方式は，終了時の時刻印が実行の半順序と矛盾していないため，時刻印を観測できれば可観測となる．

このような機構を統合するには次のような場合がある．

（1）　すべての並行処理制御機構が可制御である．

（2）　すべての並行処理制御機構が可観測である．

（3）　可制御のものと可観測のものが混ざっている．

（4）　すべての並行処理制御機構が可制御かつ可観測である．

（1）の場合は，特定の直列順を決めてすべてのシステムを制御するとよい．（2）の場合は，観測結果が矛盾すると矛盾がなくなるまで後退復帰をさせる方式であり，実用的とはいえない．（3）の場合は，可観測のシステムの観測結果が矛盾のないとき，それを用いて

可制御システムを制御することになる．一つの可観測な分散システムと一つの可制御な分散システムの統合には有用である．(4)の場合は実用的と考えられる．システム1から順に，まずシステム1で実行しそれを観測する．この観測結果と矛盾しないようにシステム2を制御する．以下 $i=2,\cdots$ でシステム i の実行順を観測しそれでシステム $i+1$ を制御するということを繰り返す．

4. メタデータの管理

分散管理システムにおける，動的構成や安全性と大きく絡む問題にメタデータ管理がある．メタデータは主としてシステムの物理分散の情報に対応するものである．

(1) 動的構成に対応するもの．システムの故障情報など，現在利用可能なシステムやその状態．データのコピーがいくつかあれば，その位置や利用可能性．

(2) 処理状況に対応するもの．システム内で実行中の処理の状況を記憶し，新しく実行するべき処理の処理方式を決める．

処理効率の観点から複数の局に記憶されることが多いため，メタデータは高度に重複したデータになると考えられる．メタデータが動的に変更されることを考えると，並行処理はデータのみならず，メタデータに対しても行わなければならないことになる．

メタデータを考えた場合の並行処理については，従来ほとんど扱われていない[11]．この場合次のような問題点がある．

(1) メタデータとデータの内容が矛盾してはならない．

(2) メタデータ参照が多く，このための効率低下を避けなければならない．

(3) データとメタデータの両方を利用する処理単位の扱いを考えなければならない．

(4) 処理コストをできるだけ下げるための方法を開発しなければならない．

(5) メタデータとデータの性質の違いを反映した処理方式を開発しなければならない．

まず(1)について説明する．並行処理の正しいアルゴリズムはメタデータとデータの一致を保障しなければならない．すなわち，任意の処理単位に対してそれが参照するメタデータと操作するデータが矛盾してはならない．特に分散の場合関連するメタデータとデータが異なる局に置かれる．したがって，メタデータを考慮した並行処理は全域的に分散しているメタデータの変更および参照を含めた直列可能性を保障する制御となる．

データに対するすべての操作はメタデータの参照を経て行われるので(2)の問題も重要である．メタデータに対する変更操作が存在するとき，メタデータ変更による並行処理に関する矛盾する操作が非常に起こりやすくボトルネックとなる．したがって，メタデータの操作上の矛盾をいかに減少させるか，あるいはボトルネック問題をいかに緩めるかということは重要である．たとえば，一つの局を回復するときそこに記憶しているデータコピーが他の局に記憶している同一名のデータコピーと異なっていれば無効になる．そのときデータの回復を行わなければならない．またデータの回復にともなってすべての局に格納しているメタデータの変更も必要となる．このメタデータの変更のうち一つでも矛盾すると大きなコストがかかったデータ回復が無効となる．

メタデータとデータやシステムの相互関連には，制御的なものと観測的なものとがある．制御的なものはメタデータを変更するとそれに対応してシステムが変わるもので，観測的なものはシステムを変更するとそれにともなってメタデータが変わるものである．このため，この両者を利用するような処理単位では，このような相互関連による変更の時間的関係などを考えて正しく働くように構成しなければならない．

広域分散処理システムでは，局間のデータ転送がコストを決めると考えられ，(4)に対応して通信コストを下げるのが特に重要である．データが重複している場合，データの回復に要するコストは大きい．したがって各局の回復にあたって，できる限りデータ転送を減らすようにすべきである．ある局を回復するとき，他の局に記憶されている同一名のデータコピーが変更されていなければ，その局のコピーを回復しなくて良い．したがって，メタデータをうまく扱えばこの検査は可能である．

メタデータとデータの違いは次のとおりである．

(1) すべての処理はまずメタデータを参照するので非常に参照が多い．

(2) データベースのデータに比べて量的に少ない．

このためメタデータはすべての局で重複させるのが良いと考えられるが，変更時にはメタデータ間の矛盾を避けるための施錠も必要となりうる．

上記の問題は一般的な立場から考えると，メタデータを含めた直列可能性の保障と，いかにしてメタデータ上のボトルネック問題を解決するかという問題になる．

次にいくつかの例を示す[3),5),11)]．

局の停止：ある局を保守上の理由などで停止させる．

（1） メタデータにより現在生きている局を探す．

（2） これらの局に「停止する」というメッセージを送る．

（3） これらの局のメタデータが変更されたのを確認して停止する．

（4） 処理はメタデータをみて行うので，たとえばデータの変更時には停止中の局にあるデータには変更要求を出さない．

局の再実行：停止中の局を働かせる．

（1） 局が実行開始予定であることをすべての局に伝える．

（2） 各局は対応するメタデータを変更しその旨を実行開始局に知らせる．

（3） 実行開始局は返事を受け取った局のメタデータを合成し自分のメタデータを作る（この局にとっては返事を受け取った局のみが実行中の局となる）．

（4） データを回復する．各データの変更時刻印をみて，それが停止前と変わっているデータのみについて回復処理を行うことでデータ転送コストを減らす．

（5） 各局のメタデータを変更し実行開始局を使えるようにする．

メタデータとデータの両方を扱う処理についてはメタデータを主に扱う部分とデータを主に扱う部分に分けて扱うことになる．

5. むすび

本稿では，分散処理の種々の技術的課題を整理するとともに，集中システムに比べて分散システムで特に扱うべき課題に絞って詳述した．分散処理では，システム自体が地域的に広がった大きなものになりうるため，構成要素（ハードウェアおよびソフトウェア）の多様性や利用者の多様性に対処することが重要である．また，このようなシステムでは，どこかのシステムを保守のために止めたり，利用中に新しいシステムを増設するなどのことが行えることが望ましく，動的な構成変更の能力が重要である．複数の局を同時に動かすために並行処理も不可欠であり，安全性の確保のためにはデータの重複も重要である．並行処理やデータの重複と動的な構成変更という問題が絡み合うと複雑な問題になるため，メタデータ管理という形で扱う方法を示した．2〜4 で示したこれらの課題は分散システムの特色を生かすために不可欠であるが，まだ研究途上のものであり，今後さらに実用的な方式に向けた研究開発が必要であると考えている．

謝辞 本稿の一部の内容についての共同研究者である三菱電機近藤誠一氏ならびに九州大学工学部大学院学生仲興国氏に感謝します．

参考文献

1) Bernstein, P. A. and Goodman, N.: Timestamp-Based Algorithms for Concurrency Control in Distributed Database Systems, Proc. 6th Int. Conf. VLDB (Oct. 1980).
2) Bernstein, P. A. and Goodman, N.: Concurrency Control in Distributed Database Systems, ACM Computing Surveys, Vol. 13, No. 2 (June 1981).
3) Bernstein, P. A. and Goodman, N.: An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases, ACM Trans. on Database Systems, Vol. 9, No. 4, pp. 596-615 (Dec. 1984).
4) Eswaran, K. P., Gray, J. N., Lorie, R. A. and Traiger, I. L.: The Notions of Consistency and Predicate Locks in a Database System, CACM, Vol. 19, No. 11 (Nov. 1976).
5) Goodman, N., Skeen, D. and Chen, A.: A Recovery Algorithm for a Distributed Database System, ACM SIGMOD, pp. 8-15 (1983).
6) Kambayashi, Y. and Kondoh, S.: Global Concurrency Control Mechanisms for a Local Network Consisting of Systems without Concurrency Control Capability, Proc. National Computer Conference (1984).
7) 上林弥彦：分散システムにおける平行処理，情報処理学会「知識情報処理シンポジウム」，pp. 191-198 (Sep. 1985).
8) Traiger, I. L., Gray, J., Galtieri, C. A. and Lindsay, B. G.: Transactions and Consistency in Distributed Database Systems, ACM Trans. on Database Systems, Vol. 7, No. 3, pp. 323-342 (Sep. 1982).
9) Papadimitriou, C. H.: The Serializability of Concurrent Database Updates, JACM, Vol. 26, No. 4 (Oct. 1979).
10) 植村俊亮：大型プロジェクト「電子計算機相互運用データベースシステム」の研究開発について，情報処理学会九州支部講習会資料（計算機技術の動向―教育・研究・開発―），（情報処理学会九州支部で発行）(Oct. 1986).
11) Zhong, X. and Kambayashi, Y.: Consistency Mechanisms for Metadate and Database Data in a Duplicated Distributed Database System, Proceedings of the International Pre-VLDB Symposium, pp. 118-127 (Aug. 1986).

（昭和62年2月3日受付）