マルチメディア通信と分散処理 41 － 7
（１９８９．３．１３）

知的分散ＯＳの分散管理アルゴリズム

関　俊文，　岡宅泰邦，　田村信介

（株）東芝，　システム・ソフトウェア技術研究所

知的分散システムは，完全に集中管理機構を排したシステムである．各処理要素は，システムに与えられた目的を遂行するため，要素間での情報交換に基ずく協調・協力メカニズムによって，自らの役割を動的に決定する．

本報告では，知的分散ＯＳ上に採用した分散管理アルゴリズム，特に同時実行制御法と故障管理法について述べる．同時実行制御法では，可到達集合法の概念を用いたデッドロック防止と検出の分散アルゴリズムを提案する．故障管理法では，フェイル・ストップ放送通信プロトコルと，フェイル・ストップ・オブジェクトを実現することにより，多重化されたオブジェクト群が各々他の存在を意識することなく，定義，運用できることを示す．

これらの分散管理アルゴリズムにより，拡張性・適応性や処理効率を低下することなく，高い信頼性を実現することが可能になる．

Distributed Managing Algorithms for The Intellectual Distributed Processing System

Toshibumi SEKI,　Yasukuni OKATAKU,　Shinsuke TAMURA

TOSHIBA corporation, Systems & Software Engineering Laboratory
70, yanagi-cho, saiwai-ku, kawasaki, kanagawa, 210 JAPAN

The Intellecutal Distributed Processing System(IDPS) has no centralized managing element. IDPS system elements find their roles autonomously, in order to complete objectives given to the system.

This paper describes IDPS's distributed managing algorithms for concurrency control mechanisms and fault-tolerant mechanisms. The concurrency control mechanism realizes distributed deadlock avoidance and deadlock detection. The fault-tolerant mechanisms are realized by the fail-stop broadcast communication protocol and first-N-come method for accomplishing fail-stop objects.

By these mechanisms, it is possible to make highly flexible and reliable systems easily without reducing extensibility, adaptability and processing efficiency.

1．はじめに

技術革新のスピードアップ，ユーザニーズの多様化とともにシステムには従来のように速度や精度だけでなく，拡張性・保守性・操作性の向上と，システムの一部分が故障してもシステム全体としては支障をきたさない様な高い信頼度が求められている．またシステムの大規模化に伴う開発期間の長期化により，ユーザニーズの変化に柔軟に対処することが困難になってきており，単純かつ固定的な機能の追求から，複雑かつ時間的に変化する評価基準を満足する柔軟性を追求したアーキテクチャが必要になってきている．集中機構を完全に排した自律型の分散システムはこれら要求を実現する有力なアーキテクチャと考えられている．

知的分散システムは，これらの要求を実現するアーキテクチャであり，集中的な管理機構を完全に排したシステムとして提案され，いくつかの分野に適用中である[1]-[4]．本システムは，固有の能力を宣言した個々のシステム要素の単なる集合として定義され，各要素の役割は要素間の自律的な協力と協調機構によって動的に決定される．各システム要素の動作が要素自身によって分散管理されるため，要素の追加，変更が他の部分に影響を与えることもなく，適応性・拡張性・信頼性・保守性に優れたシステムが実現される．

システム全体がこのような分散管理環境下で動作するためには，同時並行に実行される処理間での矛盾が生じないようにその進行を制御したり（同時実行制御），個々の資源の負荷が均等になるように処理負荷を配分する，あるいはオブジェクトの故障の影響が他に波及するのを防ぐなどの機構を実現する各種分散アルゴリズムが必要になる．

本報告では，知的分散ＯＳで用いている同時実行制御と故障管理の分散アルゴリズムについて報告する．

2．知的分散システム

2．1　知的分散システムのアーキテクチャ

知的分散システムは，集中管理部を完全に排した分散計算機システムのアーキテクチャであり，適応性・拡張性・信頼性・保守性等の向上を目的とするものである．従来の多くの分散システムは，システム全体の動作を集中的に管理する機構を有し，システム要素の結合形態や動作順序はこの集中機構が規定する範囲に束縛されていた．これに対し知的分散システムは，個々の能力が宣言的に定義されたシステム要素の単なる集合からなり，与えられた仕事を遂行して行く過程で必要なシステム要素が，システム要素間の協調・協力によって集中管理部を介することなく動的に結合する．従って，個々のシステム要素の役割が要素自身によって決められるので，各要素がその能力を最大限に発揮することができるようになり，システムの適応性が大幅に向上する．

知的分散システムにおけるこれらの機能は，各システム要素が個々のシステム要素の能力に関する固有の知識の他に，協調・協力を行なうための知識，例えば，負荷分散，排他管理，故障管理などの知識を共通知識として持つことにより実現される．ここで「協力」とは与えられたジョブの処理をシステム要素群が互いに処理結果をやり取りすることにより結合し，その役割を分担して実行することである．また，「協調」とはシステム要素間での異なる目的の競合を調整し，システム資源の有効かつ正しい利用をはかることを意味する．このように管理分散を実現することにより，集中管理部への負荷集中，管理部のダウンによるシステム全体のダウンなどが避けられる．また，システム稼働中のシステム要素の追加・削除が容易になる．

図１に知的分散システムの構成を示す．

2．2　知的分散ＯＳ

知的分散システムを構築するための基盤として，既に知的分散ＯＳを開発している．知的分散ＯＳは，ＬＡＮによって接続された計算機群の上に知的分散システムを構築するためのオブジェクト指向分散オペレーティングシステムである．個々のシステム要素に対応するアプリケーション・プログラムは全て，オブジェクトとして表現され，非同期放送型を基本とするメッセージ通信により結合される．知的分散システムでは，各オブジェクトが固有の判断で自律的に動作するので，個々のオブジェクト毎に特別なＯＳサービスを提供する能力が特に重要である．このような能力を実現するため知的分散ＯＳは，知的分散ＯＳ核と前節の各オブジェクトの共通知識を扱う部分の２層からなる．知的分散ＯＳ核は，個々の計算機のプロセッサやメモリの管理，あるいはオブジェクト間で交換されるメッセージの送受信処理などローカルな資源管理を行なうもので，各オブジェクトへの役割割当などオブジェクトのグローバルな動作管理は行なわない．オブジェクトのグローバルな動作管理は，関連するオブジェクト群が持つ共通知識によって実現される．本ＯＳの最大の特徴は，このようにグローバルな管理機構がアプリケーション・プログラムに相当する個々のオブジェクトに

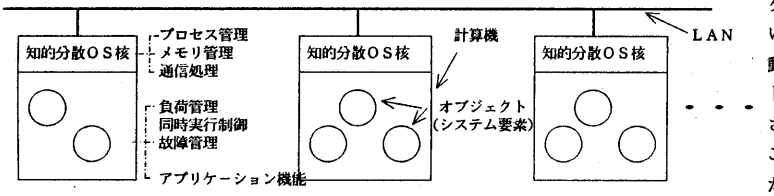

図１　知的分散システムの構成

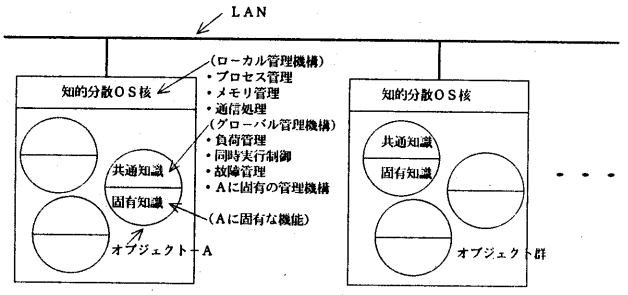

図２　知的分散ＯＳの構造

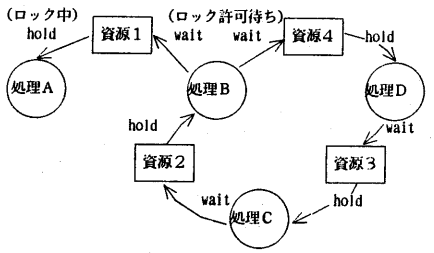

図３　Ｈｏｌｄ－Ｗａｉｔグラフ

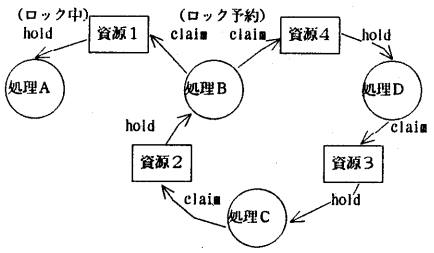

図４　Ｈｏｌｄ－Ｃｌａｉｍグラフ

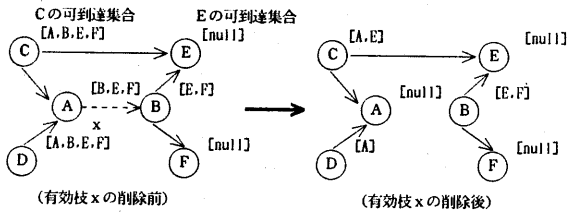

図５　有効枝削除時の可到達集合更新

分散していることであり，そのために負荷分散や同時実行制御，故障管理などの分散アルゴリズムを実装したことである．尚，各オブジェクトの共通知識の一致部分は，プログラムをリエントラントにする事により共有される．

図２に知的分散ＯＳの構成を示す．

３．同時実行制御

処理のアクセス順序が資源間で逆転し，処理結果に矛盾が生じないようにする制御は同時実行制御と呼ばれ，２相ロック法や時刻印法が広く使われているが[5]，知的分散ＯＳでは並列実行処理能力に優れた２相ロック法を採用している．本節では，２相ロック法を用いる際に発生するデッドロックの検出と防止の分散アルゴリズムについて述べる．デッドロックの対処法には，資源へのアクセスをデッドロック発生の危険がなくなるまで認めない防止法と，資源へのアクセスを無条件に認めてデッドロック発生後にそれを検出し，関連する処理の中止によってデッドロックから脱出する検出法とがある．従来２つの方法は別々に使われていたが，ここではさらにこれらの方法を併用可能なロック規約を示す．防止法を用いるには，処理はその実行に先立って使用する資源を予約しなければならない（ロック予約と呼ぶ）が，防止法では絶対にデッドロックを発生することが無く，デッドロックのために処理を中止，再実行する必要が無くなる．従って，データベースの検索のように使用する資源が処理の進行とともに判明し事前に確定できない場合には検出法が有効であり，外部への物理的な出力を伴い処理の中止や再実行が困難な場合には防止法が有効である．本報告で示す規約により，これら２つの方法が自由に使い分けられるようになる．

３．１　可到達集合法

デッドロック発生，あるいはその危険性の検出は，有向グラフ上の閉路検出に帰着できることが知られている[6]．即ちデッドロックである必要十分条件は，図３に示す処理による資源ロックとロック許可待ちの関係を表すＨｏｌｄ－Ｗａｉｔグラフ上に有向枝の閉路が存在することである．また処理群が将来どの様な順でロック要求を出してもデッドロックを起こさずに全ての処理を終了する方法が存在する必要十分条件は，図４に示す処理による資源ロックとロック予約の関係を表すＨｏｌｄ－Ｃｌａｉｍグラフ上に有向枝の閉路の存在しないことである．既に有向グラフの閉路検出分散アルゴリズムとしてプローブ法[7]や可到達集合法[8]が提案されている．プローブ法ではプローブと呼ばれるメッセージをグラフの有向枝に沿って順次ノードからノードへ転送し，それがプローブ送出済みのノードに戻ったときに閉路を検出する．従って閉路検出操作が直列的に進み並列処理の効果が期待できず，また通信エラーによるプローブ紛失などからの回復操作が複雑になる．そこで知的分散ＯＳでは，個々のノードがそこから有向枝をたどって到達可能な全てのノード（可到達集合）を記憶する可到達集合法を用いる．可到達集合法では，資源のロックあるいはロック予約によって有向枝が追加されたとき，追加有向枝の根元ノードが行先ノードの可到達集合に含まれる場合に閉路を検出したことになる．この方法によればプローブが

順次転送されるようなことが無いので並列処理の可能性があるが，処理の資源ロックやアンロック，ロック予約に伴う有向枝の追加と削除に応じて各ノードの可到達集合を変更しなければならない．しかし，有向枝の削除にともなう変更は図５に示すように複雑であり，従来は有向枝削除時の効率的な可到達集合更新方法がなかったので，可到達集合法は２相ロック規約にしたがう場合のデッドロック検出にしか利用されなかった（デッドロック検出法では，デッドロック検出時以外に有向枝が削除されるのは，処理が資源をアンロックする場合だけであるが，２相ロック規約に従えばアンロック操作を行なう処理は必要な全ての資源のロックを済ませているので，図５のようにアンロック操作中の処理から出る有向枝が存在することは無く，可到達集合の更新が簡単になる）．ここでは可到達集合法がより一般の場合にも適用できるように，効果的な可到達集合更新の分散アルゴリズムを示す．本アルゴリズムによって可到達集合法が，２相ロック規約に従わなくても処理の整合性の保証できる場合にも，またデッドロックの検出だけでなく防止にも利用できるようになる．

次に，有向枝削除に伴う可到達集合の更新アルゴリズムを示す．後述するアルゴリズムでは，図３，図４に示した処理と資源ノード間の関係を図５のように処理ノード間だけの関係に変換した有向グラフを扱う．図５は，処理Ａのロックしている資源の少なくとも１つを処理Ｂがロック要求あるいはロック予約しているときに処理Ｂから処理Ａに向かう枝を定義する規則によって，資源ノードを含むグラフから変換される．知的分散ＯＳでは，個々の処理要求が発生する毎にその処理内容を表わすオブジェクトが生成され，本アルゴリズムはこれらオブジェクト群の共通知識上に実装される．各オブジェクトは対応するノードの可到達集合の他に，逆向可到達集合，隣接集合，逆向隣接集合を記憶しており，アルゴリズムの実行はこれらのオブジェクトがメッセージを交換する事により各オブジェクト上で並列に進行する．また，各オブジェクトのメッセージ送信先はオブジェクト自身が把握しているので，通信エラー等に際してもメッセージ紛失などの回復処理が容易になる．ここでノードＡの隣接集合とは，Ａからでる有向枝によって直接到達可能なノードの集合であり，逆向可到達集合はその可到達集合にＡを含むようなノードの集合，また逆向隣接集合はその隣接集合にＡを含むようなノードの集合のことである．

本アルゴリズムは処理ノードの総数をｎとすると，最大でも（５ｎ＋１）回の通信回数で実行できる．後述する手順２），４），６）で実施されるマルチキャストを各々１回と数えると，これは（２ｎ＋４）回に減る．単一の計算機が同時に発生する処理の最大数を設定すると，計算機間で情報を交換することなく，システム中の全処理に一意に定まる番号を与えることができ，可到達集合はｎに比例するビット長のパターンとして表現することができる．従って，後に示す手順で必要になる集合演算はビットパターンの論理和，論理積，排他的論理和に帰着され，個々のノードの計算量がｎのオーダに納まる．全ノードの計算量を合わせるとn^2のオーダになるが，この計算はノードが搭載される計算機群上で並列に実行することができる．総計算量に関しては，従来報告されている分散デッドロック検出，防止アルゴリズムもn^2またはn^3のオーダであり[9]，本アルゴリズムが特に優れていることはないが，他のアルゴリズムで必要になる通信回数はn^2以上のオーダであり，処理の並列性や通信回数を考えると実質的には本アルゴリズムの方が優れていることがわかる．

＜　有効枝削除に伴う可到達集合の更新手順　＞

以下の手順では，ノードＸの可到達，逆向可到達，隣接，逆向隣接集合をそれぞれＦＲ（Ｘ），ＢＲ（Ｘ），ＦＤ（Ｘ），ＢＤ（Ｘ）と表し，ノードＡからＢに向かう有効枝を除く際の可到達集合更新を考える．

１．ＢがＡに｛Ｂ，ＦＲ（Ｂ）｝を送る．

２．ＡがＦＤ（Ａ）からＢを除き，ＦＲ（Ａ）からＢとＦＲ（Ｂ）のノードを除く．ＡはさらにＢＲ（Ａ）に属するノード群に｛Ｂ，ＦＲ（Ｂ），Ａ，ＢＲ（Ａ）｝を送る．

３．ＢＲ（Ａ）のノードＣが，ＦＲ（Ｃ）からＢ及びＦＲ（Ｂ）のノードを除く．ＣはさらにＴ（Ｃ）＝ＦＤ（Ｃ）－ＢＲ（Ａ）を計算し，Ａに｛Ｃ，Ｔ（Ｃ），ＢＲ（Ｃ）｝を返送する．

４．ＢＲ（Ａ）の全ノードから返答を受信したＡが，｛ｉ，ＢＲ（ｉ）｝（ｉ∈ＢＲ（Ａ））を記憶し，Ｚ（Ａ）＝∪Ｔ（ｉ）∪ＦＤ（Ａ）のノード群に｛ｉ，Ｔ（ｉ）｝（ｉ∈ＢＲ（Ａ））の列を送る．

５．｛ｉ，Ｔ（ｉ）｝の列を受信したＤが，Ｄが列中のＴ（ｊ）に含まれるとき，Ａに｛Ｄ，ＦＲ（Ｄ），ｊ｝を返送する．

６．返送｛ｋ，ＦＲ（ｋ），k_j｝を受信したＡがｋがＦＤ（Ａ）に含まれるときは，ＦＲ（Ａ）＝ＦＲ（Ａ）∪ＦＲ（ｋ）∪ｋとする．Ａはさらに全てのｋ∈Ｚ（Ａ）から返答を受け取ると，先に記憶した｛ｉ，ＢＲ（ｉ）｝の列中にｉ＝k_jなるものが存在するk_jを集めて，｛ｋ，ＦＲ（ｋ），ＢＲ（k_j）｝の列を作りＢＲ（Ａ）のノード群に送る．

７．｛ｋ，ＦＲ（ｋ），ＢＲ（k_j）｝の列を受信したＥが，ＥがＢＲ（k_j）に含まれるとき，ＦＲ（Ｅ）＝ＦＲ（Ｅ）∪ＦＲ（ｋ）∪ｋとする．

本手続きは可到達集合の更新に関するものであるが，逆向可到達集合の更新もＦＲ，ＦＤの役割とＢＲ，ＢＤの役割を入れ換えれば同様に行える．

3．2　ロック規約

デッドロック検出と防止は，前述したように使いわけできることが望ましい．即ち，アクセス中の処理の中止と再実行が許される資源をタイプ1，そうでないものをタイプ2とする時，タイプ1資源に対しては検出法を用い，タイプ2資源に対しては防止法を用いる．この時，デッドロックのためにタイプ2資源にアクセス中の処理が中止されてはならない．ここでは資源タイプ1の優先度をタイプ2より高く設定することにより，処理が同時に異なるタイプの資源にアクセスできるロック規約を示す．

規約1）タイプ2資源をロック中の処理は，タイプ1資源へロック要求を出してはならない．
規約2）ロック予約をしていないタイプ2資源にロック要求を出してはならない．
規約3）処理は自身がタイプ2資源をロックしていない時にのみロック予約ができる．

規約1）は，タイプ2資源ロック中処理のタイプ1資源ロックを禁止するが，タイプ1資源ロック中処理のタイプ2資源ロックは禁止しない．従来のプロトコル[10]ではタイプ1とタイプ2への優先度の与え方に規定がないのでこの両方が禁止される．

3．3　デッドロック検出・防止手順

他の処理が資源をロック中のため，ある処理のタイプ1資源へのロック要求が待たされた時，デッドロック検出アルゴリズムがＨｏｌｄ－Ｗａｉｔグラフの閉路検出を行なう．その結果閉路を検出すると，デッドロック状態から脱出するためにロック要求処理を中止するが，そうでない場合は単に資源がアンロックされるのを待つ．処理がタイプ2資源をロックする場合には必ずデッドロック防止アルゴリズムが動作し，Ｈｏｌｄ－Ｃｌａｉｍグラフ中の閉路検出を行なう．その結果閉路を検出すると，関連資源がアンロックされて閉路が無くなるまでロック要求を待たせるが，そうでない場合はただちに資源をロックする．タイプ2資源のロックを終了した段階でさらにデッドロック検出アルゴリズムが動作し，Ｈｏｌｄ－Ｗａｉｔグラフ中に閉路を検出すると，閉路に含まれるタイプ1資源のみをロック中の処理を中止してデッドロックから脱出する．閉路検出操作は，可到達集合法により瞬時に終わる．

本規約では，処理によって資源のタイプ分類が異なることを許しているが，許さない場合は規約1）によってＨｏｌｄ－Ｗａｉｔ関係とＨｏｌｄ－Ｃｌａｉｍ関係を示す有向枝の混在した閉路が生成されることはない．従って資源のタイプ分類が処理間で共通な場合には，Ｈｏｌｄ－ＷａｉｔグラフとＨｏｌｄ－Ｃｌａｉｍグラフを別々に管理する必要はなく，それらを混合したグラフを1つ管理すればよい．

4．分散故障管理

高信頼化方式には，同一の処理を複数の要素で並行に実行する並列多重処理方式と，処理は単一の要素で実行し，実行中の要素が故障した段階で予備要素が処理を継続する待機冗長処理方式とがある．待機冗長処理方式では故障時に予備要素が処理を引き継ぐのに時間がかかる，あるいは高速で処理を引き継ぐには高頻度で予備要素に処理の途中経過を通報しなければならない等の欠点がある．リアルタイム制御などでは高信頼化によるオーバーヘッドの増加や故障による休止時間も極力小さく抑える必要があるので，知的分散ＯＳでは並列多重処理方式を採用する．この時，システムの適応性や拡張性を保つためには，個々の要素は多重化の有無あるいはその程度と独立に定義，利用できなければならない．またリアルタイム制御等で要求される処理性能を得るには，多重要素間での同期操作等を極力省略する必要がある．

図6はこれらの要求を満たすために考案されている方式で，個々の要素は放送通信によって多重化された要素群を同時に呼び出す．要素Ａが放送通信によって多重化要素Ｂ1，Ｂ2，Ｂ3を呼び出し，さらにＢ1，Ｂ2，Ｂ3の各々が放送通信によって多重化要素Ｃ1，Ｃ2を呼び出す．各要素は多重化された要素群から同一内容のメッセージを受け取ると，最初に到着したメッセージのみ受取り他は無視するなどの手段で，同一のメッセージに複数回応答するのを防ぐ．この方式では多重化要素群を放送通信で呼び出すので，関連要素の多重化の有無やその程度を知る必要はなく，各要素を定義，利用することができる．しかし，この方法の実現には，次に示す2つの機構が必要になる．

1）正しく動作している多重要素群は全て同一のメッセージを同一の順序で受け取る．
2）個々の要素は正しいメッセージしか受け取らない．

1）の機構については2相コミット規約の応用やデッドロック検出機構を用いる方法が提案されているが[11][12][13]，多重要素間で同期をとる必要があったり，多量の処理あるいは記憶を必要とするなど多重化のオーバーヘッドは高い．知的分散システムでは，後述するフェイル・ストップ放送通信によってこの機構を実現する．2）の実現には，個々の要素の完全な自己診断機能[14]を仮定しない限りは，多重化要素群の出力の多数決をとる方法しか存在しないが，従来は個々の要素が関連要素の多重度を知らなければならなかったり，多重化された全ての要素からメッセージが届くのを待つ必要があるなどの好ましくない性質があった．知的分散システムでは，メッセージの先着優先方式に

有効個数の概念を導入し，これら問題を解決する分散機構を実現する．

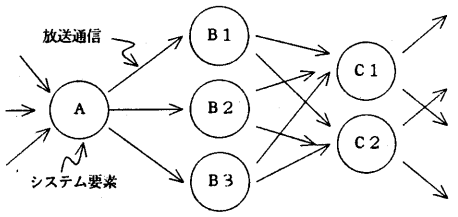

図６　放送結合による並列多重処理

４．１　フェイル・ストップ放送通信

知的分散ＯＳでは，多重化要素群は多重化されたオブジェクト群として実現され，これらは互いに異なる計算機上に搭載される．オブジェクトへの処理要求に必要なデータの伝達は，図６と同様に放送通信によって多重化オブジェクト群に対して同時に行なわれる．正しく動作している多重要素群が全て同一のメッセージを同一の順序で受け取ることのできる保証は，知的分散ＯＳ核の通信機能が行なう．即ち，知的分散ＯＳ核はＬＡＮを通過する放送モードのメッセージを常に監視しており（多重化の可能性のあるオブジェクトへのメッセージ伝達は全て放送モードで行なわれる），これまでに受け取った放送モードのメッセージを特徴づける量の累積量を記憶する．この時，正常に動作している計算機群は同一の放送モードメッセージ群を受け取るので，それらの上のＯＳ核が記憶している累積量は全て等しいはずである．そこで各ＯＳ核は送信すべきメッセージに自身の記憶している累積量を付加し，受信側ＯＳ核がメッージに付加された累積量と自身が記憶している累積量とを比較する．受信側ＯＳ核はさらに，２つの累積量が異なる場合は送信側が故障したと判断して，受信メッージを無視するとともに送信側ＯＳ核に対して故障通知を送る．各ＯＳ核は，一定数以上の計算機から故障通知を受け取ると自身が故障と判断して，その動作を停止する．ここで，メッセージを特徴づける量としては，受信メッセージ個数，長さ，あるいはチェックサム値などを用いる．

本方式では，オブジェクトが故障などの理由で，（実際にはオブジェクトの搭載された計算機の故障）メッセージを受信できなかったり余分に受信したりすると，そのオブジェクトが出すメッージに付加される累積量は，正常な計算機が記憶している累積量と異なるので，正しく動作しているオブジェクト群は同一のメッージを受け取ることができる．また，メッセージ送受信に異常のあるオブジェクトの出力は他から無視されるので，異常の影響が他に，波及することもない．多重化オブジェクト群が同一の順序でメッセージを受け取る機構は，ＯＳ核が放送モードメッセージをＬＡＮに正しく送出したのを確認してから，自計算機上のオブジェクトに転送することにより簡単に実現できる．本方式ではさらに，定められた個数以上の計算機から故障通知を受け取った計算機は停止するので，故障計算機の出すメッセージが無駄にＬＡＮを占有することもない．この時，瞬時的な故障であっても故障計算機の動作が止まるので，停止中に進行した処理と矛盾しないように計算機を再立ち上げするオーバーヘッドは高くなるが，これはＬＡＮを多重化することによって防ぐことができる．つまり，各計算機は少なくとも１つのＬＡＮを通じてメッセージの正しい送受信を行なっていれば，動作を続ける．あるＬＡＮを通じて一定数以上の故障通知を受け取った計算機はそのＬＡＮからは一旦切り離されるが，その累積量を他の計算機の累積量に強制的に一致させた後，再び接続する．このようにＬＡＮを２重化する事によりオブジェクト間通信の可用率が高まる．つまり，オブジェクトは少なくとも１本のＬＡＮが生きていれば正常にメッセージをやり取りする事ができる．アプリケーション・オブジェクトはＬＡＮが２重化されていることを気にする必要はなく，知的分散ＯＳ核がそれぞれのＬＡＮに対して同一メッセージを送信する．

各サイトのプロセッサ構成は図７に示すように，本体処理機構部と通信処理機構部よりなる．本体処理機構部には，知的分散ＯＳ核とアプリケーション・オブジェクトが組み込まれ，通信処理機構部にはフェイル

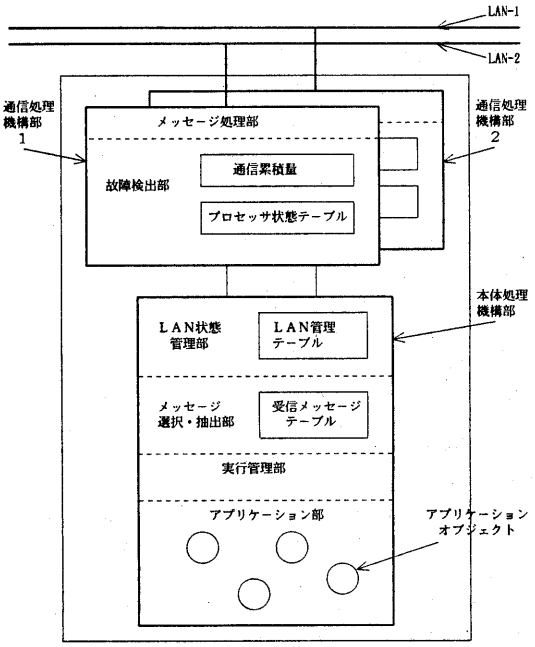

図７　各サイトの計算機構成

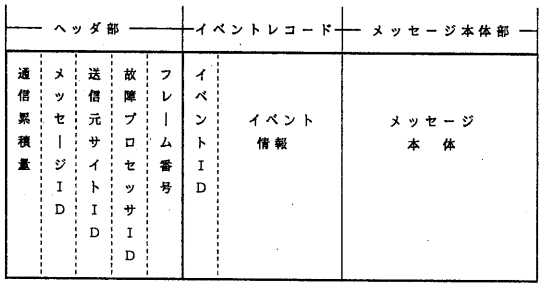

図８　メッセージ・フォーマット

・ストップ放送通信をサポートする通信プロトコルが組み込まれている．さらに通信処理機構部は２重化されたＬＡＮに対応するそれぞれ専用の処理装置からなっている．通信処理機構部は，メッセージ処理部とＬＡＮの故障検出部からなっている．本体処理機構部はＬＡＮ状態管理部，メッセージ選択・抽出部，実行管理部およびアプリケーション部よりなる．

図８にメッセージ・フォーマットを示す．メッセージヘッダ部内のメッセージＩＤは，２重化されたＬＡＮによって受信される同一メッセージを識別するために用いられ，故障プロセッサＩＤは故障していると思われるプロセッサ（警告メッセージを送信するあて先プロセッサ）のサイトＩＤを示す．イベント・レコード部には，多重化されたオブジェクトから発生されるメッセージから同一メッセージを識別するための識別子としてのイベントＩＤが含まれる．イベントＩＤの発番方法については別に報告する．

各通信処理機構部は各々専用のプロセッサを持つため，本体処理部とは独立して同時に並行して処理を行なうことができる．このためＬＡＮを２重化したことによるオーバーヘッドはなく，一重系の場合と同量である．

本方式は明らかに集中管理機構を必要としない．しかも個々のオブジェクトおよびＯＳ核が多重化されたオブジェクトの存在やその数を知る必要がないので，多重化すべき要素の追加や削除が完全に局所的な操作で実現され，高信頼化のために知的分散システムの適応性や拡張性が損なわれることはない．さらに２相コミットメントのような複雑なメッセージのやりとりやオブジェクト間での同期が不用であるだけでなく，実際には必要な処理が各計算機に付加される通信用プロセッサで実行できるので，オブジェクトの多重化による処理効率の低下もほとんどない．本方式が正常に動作する前提には，各ＯＳ核が累積量を正しく計算，記憶，比較する必要があるが，これについても個々の計算機の通信プロセッサを多重化することにより，他に影響を与えることなく完全に局所的な処理で任意の信頼度を実現することができる．

４．２　フェイル・ストップ・オブジェクト

本節では個々のオブジェクトが正しいメッセージだけを受信する機構を考える．個々のオブジェクトが完全な自己診断能力を持てば，オブジェクトは正しいメッセージしか出力しないので，この機構は不要になるが，高度な自己診断機能の実現は困難であるので，知的分散ＯＳでは多重化されたオブジェクト群のメッセージを比較する方式を用いる．多重化されたオブジェクト群のメッセージを比較する方式では，通常は多数決などによって正しいメッセージを抽出するが，そのためには各オブジェクトが関連オブジェクトの多重度を知る必要があり，また多数決に必要な数のメッセージが届くのを待たなければならない．そこで知的分散ＯＳでは，各オブジェクトが多重化オブジェクト群が送信したメッセージの最初に到着したものだけを受信し他を無視する，先着優先方式を用いる．先着優先方式は既に他のシステムでも利用されているが[11][12][15]，そのままでは誤ったメッセージが受信される可能性が残るので，ここではオブジェクトが定められた数（有効個数）の内容が一致するメッセージを受信した段階でそれを有効化する方式を用いる．但し，有効個数は対応するオブジェクトの多重度に等しいかそれ以下でなければならない．このことにより，各オブジェクトは関連オブジェクトに要求される信頼度（メッセージの有効個数）だけを知ればよく（この値はメッセージに付加することもできる），関連オブジェクトの多重度を知る必要はないので，拡張性や適応性が低下することもない．また，有効個数を調整することによってメッセージ内容の信頼度を任意を設定することができる．

全てのオブジェクトの有効個数が２つ以上の場合には，ここに示した機構だけでも多重化された各オブジェクトが同一メッセージを受信することの保証は可能であるが，実際のシステム構成では信頼度のあまり要求されないオブジェクトの多重度は１に設定される場合が多いので前述のフェイル・ストップ放送通信の機構が必要になる．一般に異常は計算機本体よりも通信路で発生することが多いので，多重度が１であっても前述のフェイル・ストップ放送通信機構だけでかなりの信頼度向上が実現できる．また故障オブジェクトが余分のメッセージを出力する場合などには，関連オブジェクトが比較すべきメッセージとして他の多重化オブジェクトからの出力を待つ必要があり，この機構だけでは処理効率が落ちるが，前述のフェイル・ストップ放送通信機構によって余分のメッセージ受信が防止されこのような待ち時間もなくなる．

４．３　信頼性評価

ＬＡＮによって接続されたシステムを構成するサイト数をＮ台とし，各プロセッサはＭ個の警告メッセージ［サイトFail］を受信した時，処理を止めるとする

（２Ｍ＜Ｎ）．この時，フェイル・ストップ放送通信プロトコルでは，Ｍ個以上のサイトが同時に同じ故障状態に陥ったときに誤った処理をしてしまう．言い換えれば，故障したオブジェクトの数がＭより小さいとき，故障した全てのプロセッサはその他の正常なＮ－Ｍ（＞Ｍ）個より多くのプロセッサから警告メッセージを受信するので処理を中止し，誤った処理を行なわない．一方，正常なプロセッサは最大Ｍ－１個の故障プロセッサからしか警告メッセージを受信しないので，それらは正常な処理を続ける．つまりシステムが誤った処理を行なう確率（故障プロセッサが処理を続ける，ないしは正常なプロセッサが処理をやめる確率）は，以下の式で示す値より小さい．

$$
\begin{aligned}
&{}_{N}C_{M}*P^{M}(1-P)^{N-M} + {}_{N}C_{M+1}*P^{M+1}(1-P)^{N-M+1} + \cdots\cdots \\
&\qquad \cdots\cdots + {}_{N}C_{N-1}*P^{N-1}(1-P) + {}_{N}C_{N}*P^{N} \\
&< N^{N-M}P^{M} \qquad (1)
\end{aligned}
$$

ここでＰは各プロセッサの故障確率であり，各々は独立に故障すると仮定する．（１）式より，本通信プロトコルの信頼性はＭの値を大きくすればするほど高くなることがわかる．但し，本通信プロトコルが効率的に動作するためには，Ｎ＊Ｐの値が小さくなければならない（少なくともＮ＊Ｐ＜１でなければならない）．例としてプロセッサの故障率Ｐ＝10^{-3}，Ｎ＝10^{2}，Ｍ＝49とするとき，Ｎ＊Ｐ＝10^{-1}となり，システムとしての故障確率は10^{-45}以下となる．但しここで，Ｍの値を大きくしたとしても，Ｍ個の応答を待ち合わせた後に処理を進めるわけではないので処理のオーバーヘッドは増加しない．ただプロセッサ状態テーブルのサイズが大きくなるだけである．

本報告で示す先着優先処理方式の信頼性も，上記同様に簡単に求めることができる．本方式において誤った処理に至るのは，故障したプロセッサの数が有効個数を越えたときである．その故障確率は，以下の式で示す値以下である．

$$
\begin{aligned}
&{}_{F}C_{CN}*P^{CN}(1-P)^{F-CN} + {}_{F}C_{CN+1}*P^{CN+1}(1-P)^{F-CN+1} + \cdots \\
&\qquad \cdots\cdots + {}_{F}C_{F-1}*P^{F-1}(1-P) + {}_{F}C_{F}*P^{F} \\
&< F^{F-CN}P^{CN} \qquad (2)
\end{aligned}
$$

ここでＦは各オブジェクトの多重度であり，ＣＮは有効個数である．本方式の信頼性は（２）式からわかるように，ＣＮの値を大きくすることによって高くすることができる．

５．　おわりに

管理分散を特色とする知的分散ＯＳ上に採用した分散管理アルゴリズム，特に同時実行制御法と故障管理法について述べた．同時実行制御では，可到達集合法を採用し，デッドロック検出だけでなく防止も同一アルゴリズムで効率よく実現できることを示した．故障管理法については，通信系及びオブジェクトを多重化しアベイラビリティを高め，かつフェイルストップ機能を実現することによって信頼性を向上させることを示した．これらの分散管理アルゴリズムは，その機能がアプリケーション・オブジェクト上に実現され分散されているため，ＯＳ機能も含めたシステムの拡張性や適応性・信頼性の向上が可能となる．

参考文献

〔1〕田村　他，”知的分散システムのアーキテクチャ”　電気学会論文誌 108-C〔6〕，1988

〔2〕関　他，”知的分散ＯＳ”，情報処理学会研究報告　88-DPS-39-3

〔3〕原嶋　他，”知的分散データベースシステム”，情報処理学会研究会　88-情報学基礎-11-3

〔4〕長谷川　他，”分散プロダクションシステム”，並列処理シンポジウムJSPP'89

〔5〕上林，”共有データベースの諸問題に対する理論”　情報処理，Vol.24，No.8，(1983-8)

〔6〕R.C.Holt,"Some Deadlock Properties of Computer Systems," Computing Surveys,Vol.4,No.3, (1972)

〔7〕K.M.Chandy, et al.,"Distributed Deadlock Detection," ACM TOCS, Vol.1, No.2, (1983)

〔8〕S.S.Isloor, et al., " An Effective ON-LINE Deadlock Detection Technique for Distributed Database Management Systems," Proc. COMPSAC'78, (1978)

〔9〕E.Knapp, "Deadlock Detection in Distributed Database," Computing Surveys, Vol.19, No.4(1987)

〔10〕J.H.Howard Jr., " Mixed Solution for the Deadloack Problem," Communication of ACM, Vol.16, No.7, (1973)

〔11〕Cooper,E.C.,"Replicated Distributed Programs," Proc. of the 10th ACM symp. on OS Principles, Operating Systems Review, Vol.19, No.5, (1985)

〔12〕D.Powell, et al.,"The Delta-4 Approach to Dependability in Open Distributed Computing Systems," Proc. of the 18th FTCS, (1988)

〔13〕M.Takizawa,"Cluster Control Protocol for Highly Reliable Broadcast Communication," Proc. of the IFIP Working Conf.on Distributed Processing, (1987)

〔14〕R.D.Schlichting, et al.,"Fail-Stop Processing : An approach to Designing Fault-Tolerant Computing Systems," ACM TOCS, Vol.1, No.3 (1983)

〔15〕R.F.Cmelik, et al. ," Fault Tolerant Concurrent C : A Tool for Writing Fault Tolerant Distributed Programs,"Proc.of the 18th FTCS, (1988)