# 分散型ＯＳの構築に関する一考察

小島　一記、　吉田　勇、　山崎　晴明
（沖電気工業株式会社　総合システム研究所）

## １．はじめに

近年、計算機の利用形態は従来の集中型の大型計算機を回線を介して端末から共同利用する形態から、各部所に中・小型の計算機を設置して分散処理をする形態へと移行しつつある。オフィスにおいても各部門にオフィスコンピュータやワークステーション、パーソナルコンピュータ等が多数導入されてきており、それらの資源を有効利用するために、それらをローカルエリアネットワーク(LAN)で結合するのが一般化している。

このようなオフィスの分散環境においては、真に分散資源の有効利用を実現するためには、ただ単に計算機間をLANで結合するだけでは充分ではなく、利用者に対してはLANを意識させずに、あたかもひとつの計算機を利用しているのと同じ環境すなわち、ネットワーク・トランスペアレンシィを提供することが重要である。

このような環境をオペレーティング・システムのレベルで提供するのが分散型オペレーティング・システム（以下分散ＯＳと略す）であり、現在国内外の大学・研究所を中心に研究が進められている。[1),2),3),4),5),6)]

分散ＯＳが満たすべき機能としては以下のものが考えられる。[7),8)]

(1)　ロケーショントランスペアレンシィ機能
　　ユーザが、資源の物理的位置を意識せずにアクセスできる機能
(2)　資源の共有制御機能
　　共有されるべき資源（周辺装置など）に対する排他制御、同時実行制御、デッドロック検出/回避機能
(3)　異機種間通信機能
　　各サイトのＯＳ、ハードウエアの差異を吸収しサイト間の通信を可能とする機能
(4)　プログラムのモビリティーを保証する機能
　　どのサイトにおいてもコードの変更なしにプログラムを実行可能とする機能
(5)　統一的コマンドインターフェース機能
　　ＯＳに対するコマンド言語をすべてのサイトで統一的にみせる機能
(6)　アクセス制御と機密保護機能
　　共有資源のアクセス権のチェックによる安全性保証と機密保護機能
(7)　信頼性保証機能
　　障害部分の検出と隔離、復旧時の処理及びリモート診断等の保守機能
(8)　負荷分散機能
　　各サイトの負荷状態の把握、スケジューリングと処理の割り当て機能
(9)　高水準言語のサポート機能
　　並列実行の特色を活かせる高水準言語をサポートする機能
(10)　アカウント機能
　　システム内におけるアカウントを取得する機能
(11)　既存ソフトウエアの実行機能
　　既存のソフトウエアをコードの変換することなしに実行可能とする機能

筆者等は、現在、上記の各機能を満足する分散ＯＳの開発を目指して、そのための基本のアルゴリズムを研究中である。

特にＬＡＮをベースとしたシステムにおいてはネットワーク資源の共有制御機能が重

要である。ネットワーク上に分散された資源の共有制御を考察する場合、デッドロックの防止、検出あるいは回復といった問題が重要な技術的課題となる。

そこで、本稿では特にこの問題を取り上げ、筆者等が考察を行った新しい分散型デッドロック検出アルゴリズムについて報告する。

## 2．分散型デッドロック検出アルゴリズム

分散システムにおける共有資源アクセスに伴うデッドロック検出法として、いくつかの方式が提案されている。[9),10),11)]

検出を分散型で行う方法としてＩＢＭのOBERMARCK[12)]により提案された方法がある。ここではその概要を述べ、次に、本提案の方式について議論する。

### 2．1　分散システムのモデル[12)]

対象とする分散システムを以下の定義および仮定によりモデル化する。

(1)　トランザクションとは、１つのサイトで作られるが、このトランザクションは、別のサイトに代理としてエージェントを置くことができる。あるトランザクションのエージェントは、そのサイト上で他のトランザクションのエージェントと資源を競合し、その結果待ち状態となる場合がある。

(2)　トランザクションが多数のサイトにエージェントを持つ場合、すべてのエージェントは、通信リンクによって直接または間接に結合される。ここで、もしトランザクションＮ個のエージェントを持つとするとただ１個のエージェントのみがアクティブであり、メッセージの送信を期待されており、他のＮ－１個のエージェントは、メッセージの受信を待っている。

図１にグローバルなtransaction wait for graph(TWFG)の例を示す。グラフのノードは、トランザクションを表し、エッジは、ノード（トランザクション）の待ち関係を示す。ノード　１からノード　２へのエッジは、ノード　２のトランザクションをノード１のトランザクションが待っていることを意味する。

集中型デッドロック検出アルゴリズムは、図１のグラフ上で３個のデッドロックを検出する。集中デッドロック検出アルゴリズムは、資源待ち要求を持つ"VICTIM（いけにえ）"を選ぶことによりデッドロックサイクルを解除する.VICTIMはアボートし、その処理は.UNDOされ資源は解放される。

分散システム上においては図１のグラフは多くのサイト上のトランザクションの処理として分解される。前に述べた仮定の上で、図１のグラフの可能な分解を問題の一貫性を保ちつつ図２に示した。ここでは、トランザクション２はサイトＢでの仕事を終え、サイトＡに移住する。そこではトランザクション３と７によって共有されている資源をトランザクション２は待っている。トランザクション３はサイトＡで仕事をし、サイトＣに移住する。サイトＣではトランザクション３は、トランザクション４を待っている。トランザクション４は、サイトＢに移住する。そこではトランザクション２と６が共有している資源をトランザクション４は待っている。同様に、グラフのそれぞれの部分を説明することができる。

問題の定義によりＡ～Ｃのサイトそれぞれは、各トランザクションのエージェントのペアを結合した通信リンクを持たねばならない。各トランザクションのエージェント間の最小リンクを図３に示す。受信を待っているトランザクションエージェントから送信を期待しているトランザクションのエージェントへと矢印は、向いている。

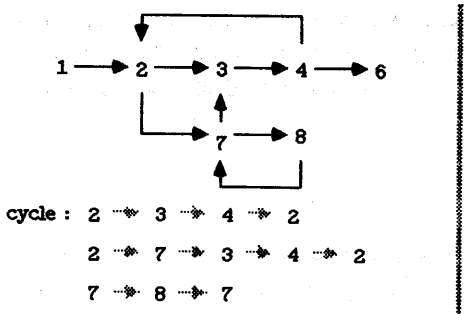


図１　集中型システムとして見たＴＷＦＧ

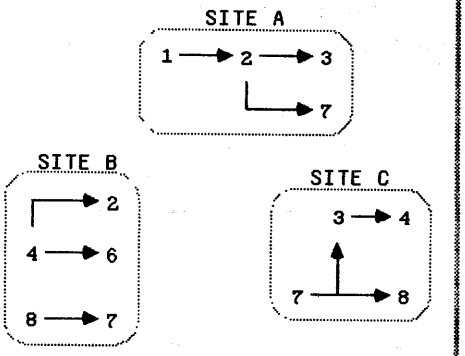


図２　３つのサイトに分割されたＴＷＦＧ

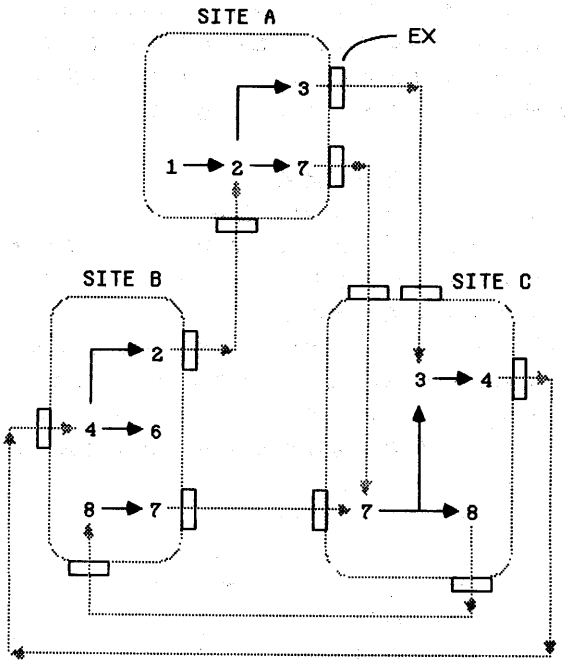


図３　コミュニケーションリンクを伴ったＴＷＦＧ

２．２　OBERMARCKのアルゴリズム[12),13)]

OBERMARCKのアルゴリズムは'External'(EX)と呼ばれるＴＷＦＧ上の特別なノードを利用する。External(EXと略す)は、自サイトの外部にある分散ＴＷＦＧ部分を表している。例えば図３のサイトＡではEX→２→３→EX　および　EX→２→７→EX　というサイクルが作られる。

(1)　各サイトはローカルにＴＷＦＧを作る。

(2)　他のサイトから送られたグローバルエッジリスト(GEL)を現在のＴＷＦＧに追加する。ＧＥＬは、ＥＸを含むサイクルの情報であり、例えばEX→２→３→EXのサイクルで考えるとEX,2,3のようなリストになる。

(3)　新しいＴＷＦＧ上のサイクルをリストする。

(4)　ＥＸを含まないサイクルがあればデッドロックであり、サイクルを破壊するためVICTIMを選択する。VICTIMを選ぶと、そのVICTIMを含むすべてのサイクルを取り除く。VICTIMが分割されたトランザクションであれば、関連するサイトにVICTIMの通知を行う。

(5)　ＥＸを含むサイクルを検査し、もしＥＸが待っているノードのトランザクション番号がＥＸを待っているトランザクション番号より大きいならば、そのサイクルの情報をＧＥＬに入れ、関連するサイトに送信する。

## ２．３　本提案のアルゴリズム

ここでは、トランザクション、EXternalノードの定義，アルゴリズムの分散化された実行環境、アルゴリズム説明のために導入された仮定については，OBERMARCKの論文[12]のものと同様とする。以下に、提案する検出アルゴリズムの記述に必要となる定義および検出アルゴリズムについて述べる。

### ２．３．１　用語の定義

（ａ）　possible pathの定義

今、ネットワークにより接続されている分散システムの内の１つのローカルなサイトに注目する（図４参照）。このサイト内において、トランザクションｉ，ｊとｊからｉに到る待ち関係が存在し、かつトランザクションｉがEXノードに対してSOURCEであり、トランザクションｊがEXノードに対してSINKであるとき、ｉからｊにはそのサイトにおいてpossible pathが存在するという。また、このようなpossible pathを（ｉ，ｊ）と記述する。これは、グローバルにｉからｊに到るpathが存在すれば、グローバルデッドロックが存在する可能性があるという意味である。

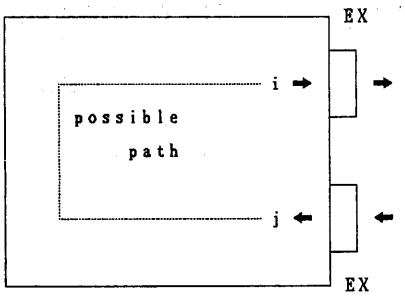


図４　possible pathの概念

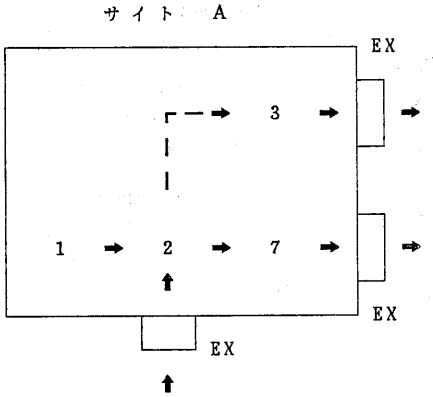


図５　possible pathの例

前述（２．２節）したOBERMARCKの論文のサイトＡの例においてpossible pathを考えると、（７，２），（３，２）となる（図５参照）。

（ｂ）　possible path間の演算

２つのpossible path（以後ppと省略する）、（ｉ，ｊ）と（ｊ，ｋ）の間の演算を式２－１，２－２のように定義する。

$$(i, j) \cdot (j, k) = (i, k) \quad \cdots\cdots \text{式}2-1$$

$$(l, m) \cdot (n, l) = (n, m) \quad \cdots\cdots \text{式}2-2$$

演算の結果として得られた（ｉ，ｋ）をpp（ｉ，ｊ）とpp（ｊ，ｋ）の結合ppと呼ぶ。この演算子は、前者ppの右要素または、左要素と後者ppの左要素または右要素が一致した時のみ演算可能であり、常にpp間に結合演算が定義されるとは限らない。pp（ｉ，ｊ）に対し、左側にｊまたは右側にｉを含むppの集合Ｒを（ｉ，ｊ）と結合可能なpp集合と呼ぶ。

（ｃ）　逆possible pathの定義

pp（ｉ，ｊ）に対し，pp（ｊ，ｉ）を逆ppとする。

（ｄ）　関連possible path集合Ｒｉの定義

ppの集合Ｒが与えられたとき、要素ｉを右側もしくは左側に含むppの集合をｉに関連するpossible path集合として関連possible path集合Ｒｉと呼ぶ。ここでその集合Ｒｉの要素の数をｎ（ｉ）であらわす。

２．３．２　検出アルゴリズム

各サイトのデッドロック検出アルゴリズムは、possible pathの導出と解析を行なう。以後のアルゴリズムで用いる「possible情報」とは、ローカルなサイトのデッドロック検出の結果として一つのサイトから一つ以上の各サイトへ送られるものである。具体的には、送出すべき条件を満たしたpossible pathの集合またはVictimの番号である。

各サイトのデッドロック検出アルゴリズムは以下のように行なわれる。

（１）　トランザクションの間の待ち関係の情報を用いてローカルな各サイト中で、pp集合Ｒを導出する。

（ａ）　同一サイト内のpp集合Ｒの要素の間に逆pp関係［(i, j)と(j, i)のようなppの関係］が存在するものがあるか調べる。

（ｂ）　存在する時は、Victimを選択してUNDO処理を行なう。

Victim選択の基準は以下のようである。possible path（ｉ，ｊ）と（ｊ、ｉ）の場合で考えると、同一サイト中のｉまたはｊの関連possible path集合Ｒｉの要素数ｎ（ｉ）とｎ（ｊ）を求め、

ｎ（ｉ）〉ｎ（ｊ）ならｉを、
ｎ（ｉ）〈ｎ（ｊ）ならｊを、
ｎ（ｉ）＝ｎ（ｊ）ならmax(i, j)
をVictimに選ぶ。

（２）　各サイトでローカルに導出されたpp集合Ｒの中に、左要素のトランザクション番号が右要素のトランザクション番号より大きいppが存在する時、そのppの各要素のトランザクションのエージェントの存在するサイトにpp情報としてppを伝送する。

（３）　pp情報を受信したサイトでは、サイトの中のppの集合Ｒに、受信されたppを組み込む。次に新しいpp集合Ｒの中に、逆pp関係を持つppの組が存在するかを調べる。

（ａ）もし存在すれば、(1)と同様のUNDO手順を実行する。

（ｂ）もし存在しなければ、pp集合Ｒの中のpp間で結合演算を実行し、結合ppを求める。

（ｃ）結合ppの演算結果を含むpp集合Ｒの中に、逆pp関係が存在するかを調べる。もし存在すれば、Victimを選択し、UNDO手順を実行する。また結合ppが(2)の条件を満たすならば(2)と同様の処理を行なう。

以下すべての、結合ppを導出するか、逆ppが導出されるまで処理を繰り返す。

UNDO手順

（ａ）Victimの通知要求をすべてのサイトにブロードカストし、通知する権利(VICTIM通知権）を取得する。

（ｂ）選択したVictim番号をすべてのサイトにブロードカストする。(Victimの通知) この時、他のサイトのVictim通知は待たされる。（つまり、Victim通知は並列には実行しない。）

（ｃ）Victimを通知されたサイトでは、そのサイト中のVictimであるトランザクションを取り除く。ここで、Victimに選ばれたトランザクションのエージェントを持つサイトでは、今までに作成されたpp集合Ｒを消去し、再びローカルにppを導出する。またエージェントを持たないサイトでは、今までに受信したpp情報及びそれによって計算された結合ppを消去する。

（ｄ）Victimを通知したサイトが、すべてのサイトからUNDO処理実行終了の通知を受信したならば、このサイトは、すべてのサイトに、Victim通知権の放棄の通知をブロードカストする。

（ｅ）別のVictim通知要求があれば、以上の手順を同様に実行する。

## ３．分散型デッドロック検出の例

実際の分散型デッドロック検出アルゴリズムの動作を理解するために例を示す。例題としては、前述の図３を使用する。以下に検出の流れを説明する。

| SITE | PP INF. (FROM) | PP (IN SITE) | UNION PP | PP INF. (TO) |
|---|---|---|---|---|
| A | —— | (3, 2)<br>(7, 2) | —— | (3, 2) B, C<br>(7, 2) B, C |
| B | —— | (7, 8)<br>(2, 4) | —— | —— |
| C | —— | (8, 7)<br>(4, 3)<br>(4, 7) | —— | (8, 7) A, B<br>(4, 3) A, B |

図６　検出アルゴリズムの反復　１

| SITE | PP INF. (FROM) | PP (IN SITE) | UNION PP | PP INF. (TO) |
|---|---|---|---|---|
| A | * (8, 7)<br>* (4, 3) | * (3, 2)<br>* (7, 2) | (4, 2)<br>(8, 2) | (4, 2) B, C<br>(8, 2) B, C |
| B | * (3, 2)<br>* (7, 2)<br>* (8, 7)<br>* (4, 3) | * (7, 8)<br>(2, 4)<br><br>n (7) >n (8)<br>n (3) >n (4) | * (7, 4)<br>* (3, 4)<br>* (2, 3) | VICTIM=7<br>VICTIM=3 |
| C | * (3, 2)<br>* (7, 2) | * (8, 7)<br>* (4, 3)<br>* (4, 7) | (8, 2)<br>(4, 2)<br>(4, 2) | (8, 2) A, B<br>(4, 2) A, B |

図７　検出アルゴリズムの反復　２

第一の検出アルゴリズムの反復（図６参照）は、各サイトにおけるデッドロック検出器のローカルな処理の結果として得られる。これは、各サイトにおけるデッドロック検出アルゴリズムの反復の結果である。ここでは、ローカルなpossible pathに結合させる他サイトからのpossible path情報はない。サイトＡではEXTERNAL（EXであらわす）は、トランザクション２を持つことを示す。つまりトランザクション２は、EXノードに対するSINKとなっており、このサイトでトランザクション２を表現するエージェントを関係づけるコミュニケーションリンクの状態を表している。また、トランザクション３

と７はEXノードを待っており、EXノードに対するSOURCEになっている。ここでは、２つのpossible pathがこのサイトで導出される。そしてpp（in site）の行に記入される。このppは、左要素が右要素より大きいので各要素のエージェントの存在するサイトにpp情報を送信する。つまりpp情報(3,2)，(7,2)がサイトＢとＣに送られる。サイトＢとＣにおいても同様にpp情報が導出される。サイトＢでは、降順のppが導出されないので他のサイトへはpp情報が送信されない。サイトＣでは、降順のpp(8,7)，(4,3)が導出されるのでサイトＡ，Ｂにpp情報として送られる。

図７は、分散型デッドロック検出アルゴアリズムの第２回目の反復を示している。第１回目の反復において送信され、受信サイトによって受け取られたすべてのpp情報がpp inf.の行に記入されている。そして、各サイトのpp情報を組み込み、逆pp関係が存在するかを調べるとともに結合ppの演算を行う。ここでは、(8,7)，(7,8)という逆pp関係が発見される。VICTIMの選択基準に従って、７又は８に関する関連PP集合の要素数ｎ（７）とｎ（８）を比較し、VICTIM=7を選択する。サイトＢの内に存在するVICTIMを含むppを取り除く。次に(4,3)，(3,4)という逆pp関係が発見される。これも、先と同様にｎ（３）＝４，ｎ（４）＝３を比較してVICTIM=3を選択し、サイト内のVICTIMを含むPPを取り除く。発見された２つのVICTIMは、そのエージェントを持つ他のサイトに通知される。

VICTIMを通知されたサイトＡとＣには、トランザクション３，７のエージェントが存在するので、アルゴリズムに従って今までに作成したpp集合を廃棄して再びローカルにppの導出を始める。最後の反復を図８に示す。サイトＡ，Ｃには、１つもppが導出されない。また、サイトＢでは、送信する必要のあるppは導出されない。この反復においてすべてのデッドロックサイクルは、解消される。

| SITE | PP INF. (FROM) | PP (IN SITE) | UNION PP | PP INF. (TO) |
|---|---|---|---|---|
| A | −− | −− | −− | −− |
| B | −− | (2,4) | −− | −− |
| C | −− | −− | −− | −− |

図８　検出アルゴリズムの反復　３

## ４．　結論と今後の課題

本論文で提案した分散型デッドロック検出アルゴリズムを、OBERMARCKのアルゴリズム[12]と比較すると下記のようないくつかの利点がある。

(1)　グローバルなデッドロックサイクルを検出し、かつ解消するために各サイトはPOSSIBLE PATH情報とVICTIMのみを送受信するだけでよい。

(2)　各サイトではローカルに導出したPOSSIBLE PATH集合Ｒと他のサイトから送られてきたPOSSIBLE PATH情報とを比較して、逆POSSIBLE PATHの検出を行う。その後、結合POSSIBLE PATHの作成が行われる。この操作は、代数的演算なので処理が容易である。

(3)　デッドロック検出の反復の回数を減らすことができる。

今後、本アルゴリズムの性能評価を解析的側面と実験的側面の両面より行って行くとともに、1節で述べた分散ＯＳの他の機能についても考察を進めてゆく予定である。

## 参考文献


[1] Bruce Walker, Gerald Popek et al.: THE LOCUS DISTRIBUTED OPERATING SYSTEM. ACM Operating Systems REVIEW. 1983.49-70

[2] E.T.Mueller, J.D.Moore.and Gerald Popek: A NESTED TRANSACTION MECHANISM FOR LOCUS. ACM Operating Systems REVIEW. 1983. 71-89

[3] Richard F. Rashid, George G. Robertson: Accent:A Communication Oriented Network Operating System Kernel. ACM operatiang systems REVIEW. 1981

[4] E. D. Jensen, N. Pleszkoch: ArchOs:A Physically Dispersed Operating System.IEEE DISTRIBUTED PROCESSING TECHNICAL COMMITTEE NEWSLETTER.1984

[5] 谷口秀夫、鈴木達郎：リモートアクセスによるＵＮＩＸの分散処理ＯＳ化，分散処理システム23-3.1984.1-8

[6] 村井　純、斉藤信男：コンピュータネットワークを用いたソフトウエア開発環境、オペレーティングシステム研究会資料24.1984.15-20

[7] 日本電子工業振興協会編：分散システムに関する調査，日本電子工業振興協会.1982

[8] 砂原秀樹、所　真理雄、中島達男：Ｓ＆Ｔｎｅｔ上に構築された分散型オペレーティングシステムの基本機能、オペレーティングシステム研究会資料24.1984.21-26

[9] Menasce.D.,and Muntz.R:LOCKING AND DEADLOCK DETECTION IN DISTRIBUTED DATABASE. IEEE Trans. Softw. Eng. SE-5.3.1979.195-202

[10] Virgil D. and Susan H. SHATTUK: ON DEADLOCK DETECTION IN DISTRIBUTED SYSTEMS. IEEE Trans. Softw. Eng. SE-6.5.1980.435-440

[11] Goldman B.: DEADLOCK DETECTION IN COMPUTER NETWORKS.TECH. REP. M.I.T.-LCS TR-185, MASSACHUSETTS INSTITUTE OF TRCHNOLOGY, CABRIDGE.MASS.,1977

[12] Obermack.R:DISTRIBUTED DEADLOCK DETECTION ALGORITHM.ACM Trans. DATABASE SYST.7.2.1982.187-208

[13] Stefano Ceri: DISTRIBUTED DATABASE PRINCIPLES AND SYSTEMS. McGraw-Hill Book Company.1984.59-66