ブロードキャストとオブジェクト・モデルに基づく
協調分散処理機構の研究

平成 3 年 1 月

田 村 信 介

ブロードキャストとオブジェクト・モデルに基づく
協調分散処理機構の研究

平成 3 年 1 月

田 村 信 介

内容梗概

本論文は1985年から1989年にかけて、筆者が（株）東芝システム・ソフトウェア技術推進部およびシステム・ソフトウェア技術研究所において行った分散システムに関する研究をまとめたものである。

技術革新によるシステム利用者の要求高度化に伴い、多くの分野でシステムが大規模、複雑化し、従来の集中管理方式によるシステム構築が困難になりつつある。特定の部分に処理が集中する結果、それが負荷や信頼性の面でのネックになったり、あるいはその部分の巨大化によってシステム開発が長期化し、さらに開発後のシステム拡張や保守が困難になるためである。そのため分散管理方式のシステムが期待されているが、現状では集中管理機構を完全に排除することができず、このような問題を解決するまでには到っていない。

これらの問題を解決するため本研究では、集中管理機構を完全に排した「宣言形のシステム」の提案とその実現性確認を行っている。宣言形のシステムは、それぞれ互いに独立に定義されるシステム要素の単なる集合であり、そこには要素群を管理する集中機構は存在しない。また各システム要素はそれぞれ固有の機能を持つが、それらの役割は要素群の協力と協調によって定まるものであり、集中管理機構等によって与えられるものではない。従って個々の要素は、集中機構の制約を受けることなく最大の能力を発揮することができる。また特定部分への負荷集中を無くし、高信頼で大規模なかつ拡張性の高いシステムの構築が可能になる。尚、宣言形システムはソフトウェア工学の分野における宣言形プログラムと同様の性質を持つので、宣言形プログラムが知的プログラムと呼ばれるのに対応して、本論文では宣言形システムを「知的分散システム」と呼んでいる。

本論文の第１章では、宣言形システムの工学的な意義と過去の他の研究における位置づけを明らかにし、研究の範囲とその概要を述べている。

第２章では、宣言形システムの構造と個々の要素が備えるべき性質を明らかにしている。さらに要素群の協力と協調を次のように分離している。つまり、協力は同一の目的を達成するために要素群が

役割を分担することであり、協調は要素群が異なる目的間の競合を解消することである。

第３章、４章では、各々協力と協調を実現する分散機構を提案している。協力機構はプロダクション・システムの原理に基づいているが、推論エンジンやワーキング・メモリのような集中機構は存在しない。個々の要素が他の要素から放送されるメッセージを受け取り可能か否かを判断することにより、ある目的の達成に必要な要素群の組合わせを発見する。協調機構では単段スケジューリングと多段スケジューリングの二つの問題が扱われる。いずれの場合も放送された要求に各要素がその性能や状態を返答するビディング（bidding）形式の機構によって、最適ではないが実質的に満足できる競合の解消策が生成される。

第５章では、要素群の正しくかつ効率的な動作を保証する方法として、デッドロックの検出と防止および多重化要素管理の分散機構を述べている。また、これらの機構さらには第３章、４章における協力・協調機構の前提には要素間での放送による情報交換があるが、オーバヘッドの低い高信頼放送プロトコルも提案している。ここで第３章から５章で述べられた分散機構を実現するアルゴリズムは協調分散形という点で通常の機能分散形のアルゴリズムとは異なる。つまりこれらのアルゴリズムでは個々の要素が特定の機能を分担する訳ではないので、ある要素に異常があってもアルゴリズムは正常に動作するし、また要素の追加や削除があっても既に存在する要素群を変更する必要はない。

第６章では、知的分散システムの実現性と有用性が示される。つまりシミュレーションのレベルではあるが、知的分散システムを鉄道運行管理と電力系統の電圧一定化制御に適用し、宣言形システム構造と放送通信等の機構が正しく、また実用的な速度で動作することを、さらに第４章における協調機構等によって大域的な最適化が可能になることを述べている。最後に第７章では、今後の課題を述べている。

関連発表論文

［１］ S.Tamura,Y.Okataku,T.Endo,and Y.Matsumoto , "IDPS : Intellectual Distributed Processing Systems," Proc.of Pacific Computer Communications Symposium,Seoul,1985, pp.129-133.

［２］ S.Tamura,Y.Okataku,T.Endo,T.Seki,and M.Arai , "Intellectual Distributed Processing System Development," 7th International Conference on Multiple Criteria Decision Making,Kyoto,1986,pp.772-781.

［３］ S.Tamura,Y.Okataku,T.Seki,and M.Arai , "Distributed Scheduling in Intellectual Distributed Processing System, " Proc.of IEEE Intl. Workshop on Industrial Automation Systems,Tokyo,1987,pp.69-74.

［４］ S.Tamura,Y.Okataku,and T.Seki , "Development of Intellectual Distributed Processing System," Proc.of IFAC 10th World Congress,Munich,1987,Vol.4,pp.37-42.

［５］ S.Tamura,Y.Okataku,and T.Seki , "Distributed Deadlock Avoidance and Detection in Intellectual Distributed Processing System," Proc.of IEEE Conf.on Systems,Man and Cybernetics,Beijng,1988,pp.1279-1282.

［６］ 田村信介，岡宅泰邦，関俊文，「知的分散システムのアーキテクチャ」電気学会論文誌Ｃ，Vol.C108，No.6,1988,pp.393-400.

［７］ 田村信介，岡宅泰邦，関俊文,古沢均，「分散スケジューリングの一方式ー知的分散システムのスケジューリング機構」，電気学会論文誌Ｃ，Vol.C109，No.4,1989,pp.291-298.

［８］ 田村信介，岡宅泰邦，関俊文，「知的分散オペレーティング・システム」，電気学会論文誌Ｃ，Vol.C109，No.7,1989, pp.507-514.

［９］ 原嶋秀次，永瀬恵子，小川真一郎，田村信介，「知的分散シ

ステムの列車運行管理システムへの応用」，電気学会　交通・電気鉄道研究会資料，TER-89-20,1989,pp.29-36.

［１０］関俊文，岡宅泰邦，田村信介，「知的分散システムにおける高信頼放送通信機構」，電子情報通信学会論文誌D-1，Vol.J73-D-I,No.2,1990,pp.117-125.

［１１］S.Tamura,Y.Okataku,and T.Seki,"Distributed Computer Systems in FMS -Intellectual Distributed ProcessingSystem-," The Intl.Conf.on Manufacturing Systems and Environment,The Japan Society of Mechanical Engineers,Tokyo, 1990,pp.497-502.

［１２］S.Matsuda,H.Ogi,K.Nishimura,Y.Okataku,and S.Tamura, "Power System Voltage Control by Distributed Expert Systems," IEEE Trans.on Industrial Electronics,Vol.37,No.3,1990, pp.236-240.

［１３］T.Hasegawa,T.Seki,Y.Okataku,and S.Tamura, "The IDPS File System," International Conf.on Information Technology Commemorating The 30th Anniversary of the Information Processing Society of Japan,Tokyo,1990,pp.119-125.

［１４］T.Seki,Y.Okataku,and S.Tamura ,"A Fault-Tolerant Architecture of Intellectual Distributed Processing System," Proc.of 2nd IFIP Working Conference on Dependable Computing for Critial Applications,Tucson,1991 (to appear)

ブロードキャストとオブジェクト・モデルに基づく協調分散処理機構の研究

目次

第 1 章　　緒 論

システムを構成する要素群は元来個々独立に管理・制御されていたが、要素群のより効率的で矛盾の無い動作を実現するため、現在では多くのシステムが集中管理方式を採用している。つまり、システム中の特定の部分が要素群の動作を集中的に管理することによって要素間での協力、協調が円滑になり、全体の運転効率向上や高度な機能の実現が可能になっている。しかし一方では、要素群を集中管理する部分へ負荷や機能が集中する結果、集中管理部の仕様決定や開発に多大の時間を要する、あるいは部分的な修正に対しても集中管理部の変更が必要になるなど、システムの拡張性維持が困難になっている。またシステムの大規模化、複雑化とともに、集中管理部の故障の及ぼす影響が大きくなり、さらには集中管理部の実現に大規模で高速なハードウェアが必要になるなど、信頼性や処理能力の面でも集中管理方式の限界が明らかになりつつある。

分散システムは、システム中に分散した機構が要素群を管理することによって特定要素への負荷や機能の集中を防ぐものであり、システムの拡張性や信頼性、処理性を維持しながら要素群の効率的で矛盾の無い動作を実現する。そのため多くの分散アーキテクチャの提案や応用システムの開発が試みられているが［６］～［１１］、それらの多くは階層形のシステムや、マネージメント・バイ・コンセンサス（management by concensus）［２２］と呼ばれる形態であり、システムから集中管理部を排除するまでには到っていない。つまり階層形のシステムでは、上位層の要素が下位層の要素群を管理するマスタ・スレーブ（master-slave）関係によってシステム全体の正しい動作を保証するが、マスタの存在は集中管理部の存在に他ならない。またマネージメント・バイ・コンセンサスでは、全ての要素がシステム全体の状態を監視して自身の正しい動作を維持するが、これは集中管理部を個々の要素上に多重に配置するのと等価である。

そこで本研究では集中管理機構を完全に排したシステムとして、宣言形のシステム構造を持った知的分散システムの実現を試みる。そこではシステムはシステム要素の単なる集合からなり、それらの動作を管理する集中管理機構は存在しない。つまり個々の要素は各々独立でシステム内でのあらかじめ決められた役割は持たない。要素の役割は、要素群が状況に応じて協力、協調することによって定まる。従って分散システムの特徴である高い拡張性と適応性、信頼性や処理能力を実現することができる。従来の集中管理方式のシステムでは、集中管理部が個々の要素の役割をあらかじめ想定して、それを管理手続きとして記憶していた。そこで従来のシステムを手続き形システムと呼ぶのに対して、ここで考えるシステムは、個々の要素がその役割をあらかじめ定められることなく、他から独立にその存在を宣言するだけであるので宣言形システムと呼ぶ。

類似のシステムとして既に自律的な性質を備えた要素群の結合からなる構造が提案されているが、第２章ではシステム要素の自律性とそれらの協力・協調に関する新たな定義あるいは解釈を与える。従来の研究では要素の自律性と協力・協調の間の区別があいまいであり、要素の自律性に他の要素の構造や機能などの要素群の協力・協調に必要な知識の所有まで求めるものが多かったが［３］［４］［５］［１１］、ここでは要素の自律性をその内部動作に関する知識だけに限定して、自律性から協力・協調の概念を分離する。要素の自律性が他の要素の構造や機能に関する知識の所有を要求する場合は、個々の要素の能力が他の要素の持つその要素に関する知識の程度に束縛されることになるが、この分離によって要素の独立性が高まり各要素がその能力を最大限に発揮できるようになる。さらにここでは、協調分散形エキスパート・システム等［１２］［１３］［１４］ではあいまいであった協力と協調の概念を次のように明確に区別する。即ち要素群の協力とは、要素群が共通の目的を達成するために互いに役割を分担することであり、協調は異なる目的を持つ要素群が互いにその競合を解消することである。この区別により、

従来どちらか一方あるいは両者を不完全な形でしか実現できなかった要素群の協力と協調機構の確立が可能になる。

これらの概念によって、宣言形システムにおける互いに独立に定義されかつ協力、協調能力を備えた要素群の構造と機能が明確になるが、第２章では最後に、宣言形システムを計算機システムとして実現する際の基礎となる知的分散オペレーティング・システム（ＯＳ）の構造についても考える。ＬＡＮで結合された計算機群を統合管理するＯＳとして既に多くの分散ＯＳが提案されているが［１５］［１６］［１７］［１８］、知的分散ＯＳは次に示す２つの特徴を持つ。第一は、従来の分散ＯＳではシステム全体の管理はＯＳレベルの特定のプロセスが集中的に行っていたのに対し、本ＯＳでは全ての管理を個々の要素の結合で行うようにして、完全な分散管理を実現したことである。第二は要素間の情報交換に全面的に放送通信を導入したことであり、それによって要素の位置や存在、多重度からの完全な独立性を実現したことである。

第３章、第４章では要素群の協力と協調を実現する分散機構を提案する。第３章で示す要素群の協力機構は、基本的にはプロダクション・システムの原理［１９］に基づいているが、推論エンジンやワーキング・メモリのような集中機構は全く存在しない。そこでは個々の要素が、他の要素から送られてくるメッセージを自身が受け取り可能か否かを判断することにより、ある目的の達成に必要な要素群の組み合わせを発見する。要素群の協力を実現する分散機構として既に提案されているContract Net等は［１３］［１４］、与えられた目的が単独の要素で達成できる時には有効であったが、ここで述べる機構によって、目的が単独の要素で達成できない場合でもそれを達成する要素群の組み合わせが発見できるようになる。

要素群の協調機構はいわゆる動的スケジューリングと呼ばれる機能を実現するが、これには単段スケジューリングと多段スケジューリングの２つのアプローチがある。第４章ではこれら２つの考えに基づく要素群に分散したスケジューリング機構を考える。ある仕事

の完了に必要な一連の作業の間には、本来は実行順序の先行関係等が存在するが、これらの作業が互いに独立であるとみなすのが単段スケジューリングである。つまり将来発生の予想される要素間の競合を無視して、現在の競合だけを調整する。従って問題も比較的簡単で要素数の多項式オーダ計算量の最適化アルゴリズムも存在する［２０］［２１］。しかし最適化アルゴリズムは、多項式オーダとは言え大量の計算を必要とする。またその分散化に際しても、各要素が他の全ての要素の状態を監視したり、全要素が同期して動作しなければならないなど、宣言形のシステムに適した形態の実現は難しい。そこでここでは完全な最適性は保証しないが、宣言形システムの条件を備える方法として、ビディング法［２２］を拡張した両方向ビディング法を提案する。ビディング法では特定の要素に負荷が集中する恐れがあったが、両方向ビディング法によってその危険性が緩和される。さらに個々の要素には、それに関連した局所的な要素群の状態情報と、それら要素間での同期しか要求されず、各要素の自律性維持が可能になる。

多段スケジューリングでは、個々の仕事に必要な作業群のつながりから将来発生する要素間の競合も考慮した競合解消を行う。従って単段スケジューリングよりも優れた競合の解消策を生成するが、最適策を生成するには本質的には全ての可能性を数え上げるしかなく膨大な計算が必要になる。そのため状況に応じて動的に競合を解消する方法として現在実際に利用できる方法はディスパッチング・ルール［２８］［２９］ぐらいである。しかしディスパッチング・ルールは、例えば処理量の最も少ない作業から開始するなどのルールに基づいて要素間の競合を解消するものであるが、ルールと最適性との関係が経験的なものであり、生成する競合解消策に対する最適性の保証は全く無い。そこでここではより最適性と結びついたルールを導入することにより、競合解消策の質を高める。ここでルールの評価は個々の要素上でビディングあるいは両方向ビディング法と同じように行われるので、各要素は同時に宣言形システムの性質

も備えることになる。

要素群の結合動作には、第３章と第４章で述べた協力と協調機構以外にも、同時実行制御や要素間での情報交換機構等が必要である。第５章ではこれらの機構を実現する分散アルゴリズムを考える。分散同時実行制御の一つとして、既にデッドロックを検出あるいは防止する多くの分散アルゴリズムが提案されているが［３２］［３３］［３４］［３５］、それらのアルゴリズムでは要素数の２乗のオーダの通信回数が必要であった。ここでは通信回数を要素数のオーダにまで下げた、リアルタイム・システム等でも使用可能な方式を実現する。また、従来はデッドロックの検出と防止法の混用は非常に限られた形態でしか許されていなかったが［３６］、２つの方式の混用形態の制限も緩和する。

知的分散システムの協力・協調機構あるいは同時実行制御の機構では要素間での放送通信が中心的な役割を果たす。しかし放送通信では、送信側と受信側でメッセージ受け取り確認信号を交換するのが難しく、信頼性の問題から放送通信が実際のシステムで本格的に利用されることは無かった。そこでここでは、メッセージ受け取り確認信号の交換を行わない高信頼放送通信プロトコルを提案する。最近になって放送通信を高信頼で行う方式が多く提案されるようになったが、それらの方法では集中機構を仮定したり［３７］、各メッセージに大量の制御情報を付加する必要があるなど［３８］［３９］［４０］オーバヘッドが高く実用に耐えるようなものは無かった。第５章ではさらに、提案する放送通信プロトコルを利用して、要素の位置や多重度から完全に独立なシステム高信頼化方式を実現する。

最後に第６章では、知的分散システムのリアルタイム制御への２つの適用例を考える。第一は鉄道運行管理システムへの適用であり、第５章で述べた分散機構を実装した知的分散ＯＳによって高い処理能力と拡張性、信頼性を兼ね備えたシステムの実現が可能になることを示す。鉄道運行管理システムは元来局所的、分散的な性質を持

っており、要素群の協力や協調が本格的に必要になるシステムではない。そこで第二の例では電力系統の電圧一定化制御に第4章で述べたビディング法を適用し、大局的、集中的な性質を持つシステムであっても宣言形構造によるシステムの構築が可能になることを示す。

第２章　知的分散システムのアーキテクチャ

第１節　序言

集中管理機構を排除した、より大規模で効率的な、しかも高い信頼性と拡張性を備えたシステムとして、「自律、協調形の分散システム」が期待されている［１］［２］。しかしこれまでに実現された分散システムは、いずれもシステムの局所的な動作の管理機構を分散しただけであり、システム全体に渡る動作の管理は依然として集中管理機構に頼っていた［６］～［１１］。そこで本章では、システム管理機構の完全な分散化を実現する「宣言形のシステム・アーキテクチャ」として「知的分散システム」を提案する。

まず第２節では、知的分散システムの目的と、それを実現する構造である宣言形のシステム・アーキテクチャの概略を述べる。宣言形のシステムはシステム要素の単なる集合から成り、要素群の動作を管理する集中機構は全く存在しない。個々のシステム要素は他の要素からは独立に定義され、それらの役割は相互の協力と協調によって非決定的に定まる。従って各要素は状況に応じて最大の能力を発揮することができる。

宣言形のシステムは自律的な要素の協力と協調によって動作するが、第３節では要素の自律性を定義し、それら要素間の協力と協調の機構を提案する。従来の研究では、要素の自律性と要素間の協力・協調の概念の区別があいまいで、要素の自律性に他の要素の構造や機能などの要素群の協力と協調動作に必要な知識を求めるものが多かった［３］［４］［５］。ここでは要素の自律性を要素の内部動作に関する知識だけに限ることにより、自律性から協力・協調の概念を分離する。要素の自律性に他の要素に関する知識をも含めると、個々の要素の能力が、他の要素がその要素に関して持つ知識の程度に束縛されることになるが、この分離によって個々の要素の独

立性が高まり、各々がその最大の能力を発揮できるようになる。また要素群の協力と協調の機構は、協調分散形エキスパート・システム等として広く研究されているが［１２］［１３］［１４］、協力と協調を明確に分離しているものは無く、どちらか一方を実現するかあるいは両者を不完全な形で実現しているものが多かった。ここでは協力と協調を明確に分離することにより両方の機構実現を可能にする。

第４節では、計算機システム上での自律性を備えた要素群と、それらの間の協力と協調機構の実現構造を述べる。要素群はＬＡＮあるいは内部バスで結合した複数の計算機上のオブジェクト群として実現される。オブジェクト・モデルを採用することにより、要素の自律性が容易に実現できる。ここではさらに、分散計算機群の管理からも集中機構を排除する目的で、知的分散オペレーティング・システムの構造を提案する。従来のオペレーティング・システム（ＯＳ）では、ＯＳレベルの特定プロセスがシステム中の定められた種類の要求を集中的に処理する場合が多かったが［１５］［１６］［１７］［１８］、提案構造ではＯＳ機能も放送通信によって結合するオブジェクト群の協力と協調によって実現される。

第５節では、分散システム構築の最大の問題点である通信路へのメッセージ集中を回避する方法についての考察をする。

第２節　宣言形システム

現在の情報システムは、本来は分散している構成要素群を統合して集中的に管理する集中形システムの形態をとることにより、構成要素個々が持つ能力の単なる結合の枠を超えた高度で複雑な機能を実現して来た。しかし技術の進歩とユーザー・ニーズの高度化・多様化とともに大規模かつ複雑化した情報システムを集中形システムとして構築するのは困難になりつつある。例えば、管理処理の集中する集中管理機構の仕様が決まらないとシステム全体の設計・製作が進まないが、その部分が大規模また複雑になり過ぎて仕様決定に多大の時間が必要になる。あるいはシステム規模が大き過ぎて集中機構の処理能力が追いつかなくなる。またシステムの大規模化とともにその信頼性の与える影響も大きくなっているが、集中形システムでは集中管理機構の故障がシステム全体の停止を招く。さらにシステムの部分的な変更に際しても複雑な集中管理機構の変更が必要になり、ユーザー・ニーズの変化に適応するのが困難になる。

これらの問題は全て集中管理機構への処理の集中に起因するものである。分散形システムの目的は、システムからこの集中管理機構を除くことであり、それによって大規模、高信頼でかつ柔軟性の高いシステムの構築を可能にすることである。しかし従来から多くの分散形システムが提案、構築されているが、これらの目的は達成できていない［６］～［１１］。それは既存の分散形システムが機能や負荷の分散を行なったものの、個々の分散要素の役割を決定する管理機能の分散を実現していないからである。例えば一般的な階層形分散システムでは、上位要素の目的はその下位に位置する要素群の役割分担によって達成されるが、上位と下位の要素はマスタとスレーブの関係にあり、下位要素群の役割決定等の処理は上位要素が集中的に行なう。そこで集中管理機構を排したシステム構造として「自律、協調形の分散システム」が注目されている［１］［２］。これらのシステムは、個々の要素の動作を要素自身が管理するので

自律形、あるいは要素群の結合によってそれらの動作を管理するので協調形の分散システムと呼ばれる。

ここでは、このような形で集中管理機構を排した知的分散システムの実現に当って「宣言形のシステム構造」を提案する。ここで宣言形システムとは、計算機言語の分野で用いられる宣言形プログラミングの概念をシステムに拡張したものである。宣言形プログラムの対立概念である手続き形プログラムでは、プログラム中の個々の文はあらかじめ目的とする処理（手続き）に沿って固定的に配列されているが、宣言形のプログラムでは個々の文は単にその存在が宣言されるだけで文の配列に意味は無い。個々の文の実行順序は、入力や他の文の実行状況によって非決定的に定まる。従って宣言形のプログラムは、個々の文のあらゆる組合せに対応する動作が可能であり、実行順序の固定している手続き形のプログラムよりもはるかに適応性が高い。宣言形のシステムにおいても個々のシステム要素は他の要素群の脈絡と関係無く単に宣言されるだけである。個々の要素の役割を定める集中管理機構のような手続きに相当する機構は無い。システム要素の役割は、与えられた仕事に対して個々の要素がその宣言された機能に関する情報を交換、組合せることによって（これを要素群の協力、協調と呼ぶ）非決定的に定まる。従ってシステム要素はその能力を、固定的な手続きに束縛されること無く最大限に発揮することができる。宣言形のプログラムが知的プログラムと呼ばれるのに対応して、ここでは宣言形のシステムを知的分散システムと呼ぶ。

自律、協調形分散システム実現の鍵は個々のシステム要素の自律性とそれらの間の協力と協調の機構であるが、知的分散システムでは要素の自律性を自身の内部状態に関する知識と機能に限る。つまり自律的な要素とは自身の内部状態へのアクセス手段と自身の能力に関する知識を備えるものであって、他の要素に関するものは、例えば要素間の通信プロトコルなどの最低限の知識以外は持たない。自律的な要素に他の要素の構造や機能に関する知識まで持たせる考

え方もあるが、それは宣言形のシステム構造に反する。それらのシステムでは個々の要素は他の要素群の機能や性質を考慮して定義され、他の要素群の脈絡と独立に定義することはできない。また個々の要素の能力が他の要素の構造と知識によって制限される。

知的分散システムではこのように限られた情報と機能から成る要素群の上の協力と協調機構を実現する。この時これらの機構が、例えば、システム全体に渡る最適化アルゴリズムをいくつかの機能に分割し、それらの１つ１つを個々の要素が担当するような機能分散の形態であっては、システムの拡張性や信頼性を向上することはできない。それは複数の要素上に集中管理機構を分離したに過ぎない。宣言形システムの性質は、個々の要素がその意思決定を互いに比較あるいは調整するような形態でないと実現できない。次節以後では、このような分散形態の実現を考える。

次に集中管理機構を排除した宣言形のシステム構造の性質をまとめる。

１）大規模性　　負荷集中部を排することにより高速、大量の処理を可能にする。またシステム開発負荷の特定部分への集中を無くして、要素群の並列開発を可能にする。それに伴ない、仕様決定部分からの逐次開発、テスト、運用が可能になる。

２）信頼性　　どの部分が故障しても全体故障にはならない。信頼度の要求に応じて、個々の要素の多重度設定やその変更が他から独立にできる。

３）適応性　　個々のシステム要素が、他の要素がそれに関して持つ知識の制約を受けることなく、状況に応じてその最大の能力を発揮できる。

４）拡張性　　機能の変更や追加、性能の向上などが、対応する要素の変更や追加だけで、他の要素から独立に行なえる。

５）保守性　　要素の修理や点検追加が、他の要素から独立にまたそれらの要素の動作を止めずに行なえる。

６）経済性　　処理負荷の要素群への適切な配分による資源の効率

的利用が可能である。また負荷に応じた要素の増減により、システム規模を負荷に対して常に最適に保つことができる。

第３節　知的分散システムの論理構造

知的分散システムは、互いに独立に宣言されるシステム要素の単なる集合から成る。これら要素群の役割は全てそれらの間の協力と協調によって分散的に定まり、要素群を管理する特別の要素は存在しない。集中的な管理機構を持たずに要素間の協力と協調を実現するには、個々の要素の自律性が不可欠であるが、ここでは要素の自律性を次のように定義する。

［定義２．１システム要素の自律性］

システム要素が自律的であるとは、要素がその能力に関する知識と、その内部状態を変更あるいは参照する機構を自身の内部に備えることである。

従来の自律、協調形の分散システムでは個々の要素の自律性に関連要素群の能力や構造に関する知識等を仮定するものが多かったが、知的分散システムでは要素の自律性をその内部構造に関するものに限り、他の要素の存在や位置、内部機構等に関する知識は要求しない。個々の要素が互いに他の要素の存在や内部機構を知っている場合は、要素群が矛盾無く動作するには都合が良いが、宣言形のシステムの特性は失われる。そのようなシステムでは要素が互いに独立で無くなり、個々の要素の能力が、他の要素が持つその要素の構造や機能に関する知識の程度に束縛される。つまり個々の要素の能力を最大限に引き出すには、各要素が全ての要素の構造や機能を知らなければならなくなり、要素の開発にシステム全体を開発するのと同程度の労力が必要になる。また１つの要素の変更や追加に対して、他の要素が持つその要素に関する知識も変更する必要が生じ、システムの拡張性や保守性が低下する。

自律形の分散システムの代表的な例はシナジェティクス［３］や、ニューラルネットワーク［４］［５］であるが、これらのシステム

では個々の要素は、互いに隣接する要素の存在や配置を知らなければならないと言う意味で［定義２．１］の自律性を満足しない。従って宣言形のシステムの持つ柔軟な性質を完全に実現することはできない。また要素の自律性に他の要素の構造や機能に関する知識を要求するシステムでは、異なる構造や機能を持つ要素が存在すると個々の要素が持つべき知識の量が急激に増加するため、一般に同質の要素群でシステムを構成することになる。しかし同質の要素群でシステムを構成するため、専用的な機能を持つシステムは実現できても多様な要求に答える複雑なシステムの実現は期待できない。実際このような構造のシステムに多様な機能を要求する場合は、上位に集中管理機構を仮定することが多い［１１］。

もちろん全く無秩序に動作する要素群が互いに協力、協調するのは不可能であるので、要素群は完全には独立ではなく、要素間での最低限の約束ごとが必要であるが、ここではそれを要素間の通信プロトコルに限る。つまり各要素は、他の要素の能力や負荷状態問合わせ等に関する全要素間で共通のインターフェイスを図２．１に示すように共通知識部として備える。但し、要素は全ての共通インターフェイスを備える必要は無く、自身に関連のあるものだけを備えればよい。あるインターフェイスを持たなくても、そのインターフェイスが必要な協力と協調動作に参加できないだけであって、システムに矛盾が生じることは無い。図２．１における固有知識部は、要素独自の機能を実現する機構である。

［定義２．１］で示した性質を備えた要素群の結合を、知的分散システムでは協力と協調の２つの形態に分離する。要素群が互いに結合することにより、集中管理機構を仮定すること無く意思決定や問題解決を行なう機構は、協調形エキスパート・システム等として既に広く研究されている［１２］［１３］［１４］。しかしそれらの研究では協力と協調が明確に区別されていないので、どちらか一方の機能だけが実現されるか、あるいは両者の実現が不完全である場合が多かった。協力と協調を次に示すように分離することにより、

２つの機能が明確になるとともに、これまで別々に開発された方法や機構の融合が可能になるなど、要素群結合機構の実現も容易になる。

［定義２．２要素群の協力］

要素群の協力とは、共通の目的を達成するために要素群が互いに役割を分担することである。

［定義２．３要素群の協調］

要素群の協調とは、異なる目的間の競合を解消するために要素群が互いにその要求を調整することである。

知的分散システムは図２．２に示すように、ジョブとプロセス、資源の３種類のシステム要素群から構成される。協力と協調の機構は明確に分離されていて、ジョブ要素とプロセス要素の結合が協力を実現し、プロセス要素と資源要素の結合が協調を実現する。ここでジョブ要素はシステムに投入される仕事と１対１に対応し、その要求仕様を保持する。従ってジョブ要素は、システムへの仕事の投入とともに生成され、仕事の終了と同時に消滅する。仕事の要求仕様は図２．３（１）に示すように、その入力と出力の対として記述される。図の例では、ジョブ要素Ｊは入力｛Ｉ１，Ｉ２｝の組から出力｛Ｏ｝を生成することを要求している。

プロセス要素は、システム内に分散したハードウェアやソフトウェアが有する個々の機能に対応して、その機能仕様を保持する。機能仕様は、仕事の要求仕様と同様に図２．３（２）に示すような入力と出力の対として記述される。この例ではプロセス要素Ｐは入力｛Ａ，Ｂ，Ｃ｝の組から出力の組｛Ｘ，Ｙ｝を生成する機能を持っている。プロセス要素が抽象的な概念である機能に対応したのに対して、資源要素はその機能を実行するハードウェアやソフトウェア資源そのものである。具体的には機器の制御プログラムや計算プログラムあるいはデータが資源要素を構成するが、［定義２．１］の

自律性を保証するために、その負荷状態や処理性能、入出力仕様等に関する情報も備える。

プロセス要素は資源要素が持っている機能に対応して定義されるが、その対応は必ずしも1対1ではない。つまり一般に資源は複数の機能を持っているので単一の資源要素が複数のプロセス要素に対応するし、逆に機器故障への対応や負荷の並列処理を可能にするため機能は複数の資源上に多重化されるので、単一のプロセス要素が複数の資源要素に対応する。尚、プロセス要素と資源要素は、システムへの資源追加やその変更に応じて定義、変更される。以下では誤解の恐れの無い場合には、ジョブ要素、プロセス要素、資源要素を各々単に、ジョブ、プロセス、資源と記す。

これら3種類の要素によって、協力と協調は次のように具体化される。つまり協力は、仕事の要求仕様の実現に必要な機能の組合わせを、ジョブ要素とプロセス要素間で互いの入出力仕様を交換することによって発見する過程である。また協調は、異なる仕事の要求仕様を実現する機能群に物理資源を割当てる過程であり、プロセス要素と資源要素の間で入出力仕様や負荷状態に関する情報を交換することにより、システム中の仕事が効率的に処理されるようなプロセスへの資源割当てを決めることである。協力と協調は各々分散プロダクション・システムと分散スケジューリング機構によって実現されるが、これらの分散アルゴリズムの詳細は第3章、4章で述べる。また実際には、要素群が並行に動作する際のデッドロック発生を検出または防止したり、あるいは多重化された要素群の状態を常に矛盾無く保つ等の機構が必要であるが、これらを実現する分散アルゴリズムについては第5章で述べる。ここではこれらの分散アルゴリズムが備えるべき性質をまとめる。

1）個々のシステム要素が、システムの大域的な構造や状態を知る必要が無い。例えば要素の存在個数や位置、その能力等に関する知識を仮定するアルゴリズムでは、要素の追加や変更の都度知識の修正が必要になり、システムの拡張性が損なわれる。

また各要素がシステム全体の状態知識を持つと、個々の要素の規模や負荷が増大して大規模なシステム構築が難しくなる。

2）個々のシステム要素の動作の大域的な同期をとる必要がない。大域的な同期機構は集中的な機構になりがちである。また処理速度等が能力の低い要素の性能によって制限される。もちろん局所的な同期は必要である。

3）個々のシステム要素が、アルゴリズムを互いに独立、並列に処理できる。各要素が、あるアルゴリズムの実現に必要な処理の一部を分担する機能分散の形態では、1つの要素の追加や変更が他の要素にも影響し拡張性の高いシステムを構築するのが難しくなる。また複数の要素を経由して順次進行する処理によってシステムが管理されるので、システムの応答速度や信頼性が低下する。

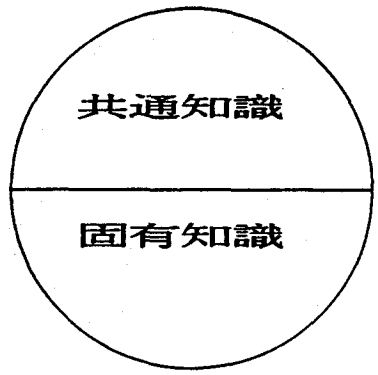


図 2.1　システム要素の構成

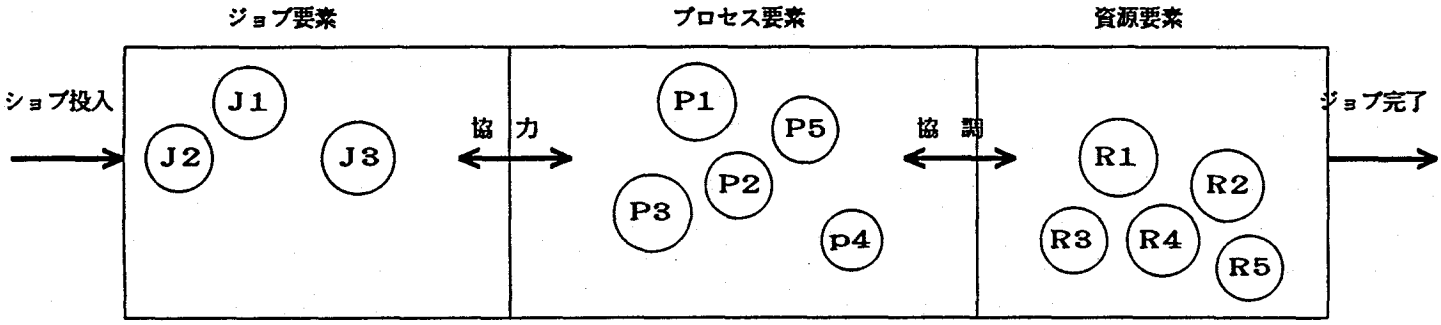


図 2.2　知的分散システムの論理構造

(1) ジョブの要求仕様　　(2) プロセスの要求仕様

図 2.3　ジョブ要素とプロセス要素の仕様

第４節　知的分散システムの実現構造

前節で示した知的分散システムの要素群は、実際には計算機のプログラムとして実現される。図２．４に知的分散システムを実現する計算機システムの構造を示す。ハードウェアはＬＡＮあるいは内部バスで接続された計算機群から成る。宣言形のシステム構造は、システム要素をオブジェクト・モデルにおけるオブジェクトとして計算機群上に配置し、オブジェクト間のメッセージ交換に放送通信を用いることによって実現する。オブジェクトの性質が個々のシステム要素に自律性を与え、放送通信がシステム要素の位置と多重度からの独立性を保証する。つまりオブジェクト群は計算機群の中にどのように配置また多重化されようとも全く同じ動作をする。

これら計算機群は知的分散オペレーティング・システム（ＯＳ）によって正しくかつ効率的な動作を保証されるが、これらのプログラムを搭載する計算機システムの個々のハードウェア自体も実はシステム要素であり、従って計算機システムそのものも宣言形の構造を持たなければならない。そこで知的分散ＯＳ自体もオブジェクトの単なる集合として実現する。つまり全てのＯＳ機能はこれらオブジェクトの結合によって実現される。従ってＯＳ自体が、負荷が特定の部位に集中することはないなど宣言形システムの性質を備える。

図２．５に個々のオブジェクトの構造を示す。図２．１に示したシステム要素の構造に対応して共通知識部と固有知識部とから成り、各部の個別の機能はメソッドとして実現される。オブジェクトとは「抽象データのインスタンス」と定義され、抽象データはその内部構造を外部に対して隠ぺいする。具体的には、外部からはオブジェクトに定義されたメソッドのインターフェイスを通してしかオブジェクトにアクセスできない。オブジェクトの内部隠ぺいの性質は、言い換えればその内部構造への必要なアクセス手段が全てオブジェクトの外ではなく内部に備わっていることであり、これは［定義２．１］で示したシステム要素の自律性に他ならない。

オブジェクトの自律性はオブジェクト個々の位置独立性を保証するが（オブジェクトの動作に必要な機能が全て内部に備わっているので、例えばその位置が変わっても同じように動作できる）、オブジェクト間の結合の位置や多重度からの独立性までは保証しない。つまりオブジェクトは他のオブジェクトの位置が変わるとメッセージ交換をできない場合がある。また多重度が変化するとオブジェクトへのメッセージ送信手順も変更しなければならない。そこでここでは放送通信によってオブジェクト間の結合の独立性を実現する。オブジェクト群が放送通信を用いて結合することにより、各オブジェクトはその存在位置や個数、名前が不明のオブジェクトともメッセージ交換ができるようになる。

システム要素間の協力、協調の前提となる要素間で共通のインターフェイスは、オブジェクト共通知識部のメソッドのインターフェイスを統一することによって実現する。各オブジェクトは共通知識部のメソッドを通じてシステム中の不特定のオブジェクトにもアクセスすることができる。但し、システム全体で共通であるのはメソッドのインターフェイスだけであって、メソッドの内部構造はオブジェクトによって異なる。また共通知識部に備わるメソッドの種類や数もオブジェクトによって異なる。固有知識部のメソッドはオブジェクト独自の機能を実現するので、そのインターフェイスはもちろんオブジェクト毎に異なる。従って他のオブジェクト固有知識部のメソッドにアクセスするには、一般にはその共通知識部中のメソッドを通して、必要とする機能に対応するメソッドのインターフェイスを知る必要がある。

これらオブジェクト群の分散計算機上での動作を実現するのは、知的分散ＯＳである。知的分散システムでは、システムに投入される仕事は要素間の協力と協調によって終了するが、これに伴ない知的分散ＯＳでは、アプリケーション・プログラムは異なる計算機上の非決定的に定まるオブジェクト群の動的な結合として構成されることになる。この時もちろん、一般には同一のオブジェクトが異な

るアプリケーション間で共有される。

宣言形システムにおけるオブジェクトおよびオブジェクト間結合の独立性を実現するため、知的分散ＯＳは次の２つの特徴を持つ。

１）ＯＳ機能をＯＳ核と個々のオブジェクト上の共通知識群の結合によって実現する。

２）オブジェクト間の放送によるメッセージ交換を可能にする。

第一の特徴である知的分散ＯＳの構造を図２．６に示す。図に示すようにＯＳ機能は各計算機に配置されたＯＳ核と、個々のオブジェクトの共通知識部の結合によって実現されるが、ＯＳ核はそれが配置された計算機のローカルな管理だけを行なうものでオブジェクト群の動作を集中的に管理するものではない。つまりＯＳ核は計算機資源自身を表わすオブジェクトに対応するもので、計算機内のオブジェクト群の要求に応じてプロセッサやメモリを割当てたり、通信要求の処理をするだけである。計算機間での負荷分散やオブジェクトの名前管理、オブジェクト間で並列に進行する処理の同時実行制御、多重化されたオブジェクト間での状態の整合性保証など、システム全体に渡る動作の管理は全てオブジェクト群の共通知識部の結合によって実現される。もちろんその中にオブジェクト群の協力と協調も含まれる。このような構造によって初めて、システムから完全に集中機構を排することができるようになる。

既に多くの分散ＯＳで、ＯＳ機能を核とプロセス（知的分散ＯＳではオブジェクトに対応する）に分離し、核の機能をできる限り小さく抑える試みがされている［１５］［１６］［１７］［１８］。ＯＳの機能をプロセスで実現することにより、例えばアプリケーション・プログラムを変更、追加するのと全く同じ手続きで、変更対象外の機能の動作中にもＯＳ機能を変更したり、また計算機の容量や負荷状態に応じてＯＳプロセスを移動するなど、柔軟性の高いＯＳの実現が可能になる。しかしこれらのＯＳでは、負荷管理や同時実行制御等のＯＳ機能の各々を専用のプロセスに割当てる機能分散の形が採られているので、全ての計算機からの要求がこれらのプロ

セスに集中するなど、分散計算機システムの利点を最大に生かすことはできなかった。もちろん単一のＯＳ機能を階層的に機能分割して、その各々を別のプロセスに割当てるなどの手段で負荷の集中を緩和することはできるが、本質的に機能分散の形態であって宣言形の構造ではない。知的分散ＯＳは図２．６に示した構造によってＯＳレベルでもアプリケーションのレベルでも宣言形の構造を実現する。宣言形のＯＳを実現するには構造だけではなく、もちろん同時実行制御や多重化オブジェクトの管理などの分散アルゴリズムが必要であるが、それについては第５章で述べる。

図２．６に示したＯＳ構造によって、さらに次のような利点が生まれる。一つはＯＳ機能のマルチ・スレッド化が可能になることである。機能分散の形態でプロセスを配置する場合は、１つのＯＳ機能が同時に複数のアプリケーションから要求された時、プロセスは同時には１つの要求にしか応えられないので、サービスを待たされる要求が生じる。従って各々異なる計算機から同一ＯＳサービスへの要求が出されると、大部分の計算機がサービス待ち状態に陥り、計算機資源の効率的利用ができなくなる。図２．６の構造の場合には個々のオブジェクトがＯＳ機能を持つので複数の要求が同時に受付けられ、大部分の計算機が待ち状態になるようなことはない。ＯＳ機能のマルチ・スレッド化はSprite［１６］なども実現しているが、そこではプログラムの排他的実行領域を細かく設定するなどの特別の機構が必要であるが、図２．６の構造では特別な機構は何ら必要ではない。

図２．６のＯＳ構造のもう一つの利点は、個々のオブジェクトに特有なＯＳ機能を簡単に実現できることである。個々のオブジェクトがＯＳ機能を持つのであるからこれは当然のことである。ＯＳ機能を機能分散の形でプロセスに割当てる形式でも、アプリケーションに特有の機能を追加するのは従来と比較すると飛躍的に簡単になる。しかしこの場合には、ＯＳ機能を実行するプロセスとそれを要求するプロセスとが異なるので、機能に変更があるとその両方を修

正する必要があり、ＯＳレベルとアプリケーション・レベルのプロセスの対応の保守管理が繁雑になる。尚図２．６の構造では異なるオブジェクトが同一の機能を持つ場合が少なくないが、同一の計算機上に同一のＯＳ機能を備えるオブジェクトが複数存在する場合は、それらの間で同一のプログラムを共有することにより、メモリ使用効率の低下を防ぐことができる。

知的分散ＯＳの第二の特徴はオブジェクト間のメッセージ交換に放送通信を用いることである。つまりオブジェクトの発するメッセージはＬＡＮに放送モードで送られ、メッセージのあて先と一致する名前を持つ全てのオブジェクトによって受取られる。この放送通信によってオブジェクト間結合の位置と多重度からの独立性が実現できる。放送メッセージやマルチ・キャスト・メッセージはＬＡＮには放送モードで１回送られるだけであるが、それを受取った計算機はそこに搭載された全てのオブジェクトにメッセージの到着を知らせる。従って実際には受信の必要が無いオブジェクトまでが励起され計算機の実行効率が大幅に下がる。そこで次に述べるような手段で放送の範囲を限定することによって、あらかじめあて先でないことのわかっているオブジェクト群の無駄な励起を無くし、実行効率低下を防止する。つまりオブジェクト名には図２．７に示すような階層構造を定義することができ、これによってメッセージの届く範囲を限定することができる。図では、あて先を「オブジェクト」としてメッセージを送出するとそれは全てのオブジェクトに届き、あて先を「Ａ」とするとそれはＡ１，Ａ２，Ａ３に届く。あて先を「Ａ１」とすると、もちろんそれは名前Ａ１を持つオブジェクトだけに届く。メッセージの送信範囲は目的によって異なるので、オブジェクト名の階層も多重に定義することができる。また動的に変化する送信範囲を設定する場合は別名定義機能を用いる。即ち、メッセージのあて先となるオブジェクト群に一時的に同一の別名を与える。実際には各計算機に通信専用のプロセッサを付加し、このプロセッサが受信メッセージのあて先オブジェクトの存在を判断するようにして、

オブジェクト間通信のオーバーヘッドをさらに下げている。

範囲限定放送は通常のシステムで用いられるマルチ・キャストとは次の点で異なる。つまりマルチ・キャストではメッセージの送出先である計算機の名前を複数個指定するのに対して、範囲限定放送では受信計算機はあくまでも不特定多数である。指定された名前を持つオブジェクトはそれがどこに存在していてもそのメッセージを受取る。従って放送通信の特徴である位置からの独立性を最大限に生かすことができる。

放送通信は前述したようにオブジェクトの位置独立性と多重度からの独立性を保証する。位置独立性によって、負荷の平準化や計算機故障等の理由でオブジェクトが移動しても、関連オブジェクトは移動前と全く同じように動作を続けることができる。また1対1通信の場合は、送信側のオブジェクトは受信オブジェクトが何重化されているかなどを知る必要があるが、放送通信の場合はメッセージは全てのオブジェクトに届くので相手の多重度を知る必要はない。つまり高信頼化などの目的でオブジェクトの多重度が変わっても、各オブジェクトは変化の前と同じ動作を続けることができる。オブジェクトの位置独立性や多重度からの独立性は、1対1通信であってもネーム・サーバの様な機構を設けることにより実現可能である。しかしネーム・サーバはシステム中の全てのオブジェクトの名前と位置の対応を記憶しなければならないし、そこにオブジェクトからの問合わせが集中する。またオブジェクトの移動等に応じてネーム・サーバの記憶内容の変更が必要になり、宣言形システムの目的を果たせない。

オブジェクト間の放送によるメッセージ交換は、協力や協調、同時実行制御等に必要な分散アルゴリズムの実現に不可欠である。即ち、知的分散システムでは固定的なオブジェクト群が機能分散の形態で役割りを分担するのではなく、不特定多数のオブジェクト群が非決定的に結合する。従って、その存在や名前が不明なオブジェクト群に同時にメッセージの送出ができる放送通信が有効である。し

かし放送通信では不特定多数の相手にメッセージを送るので、１対１通信のようにメッセージ到着確認信号の交換ができない。また可能であったとしても確認信号の数が増加して、通信オーバーヘッドが大幅に増加する。従って放送通信を高信頼でかつ実用的なオーバーヘッドの範囲内で利用するには、確認信号の交換が不要な通信の高信頼化方式が必要であるが、この点については第５章で述べる。

次に知的分散ＯＳの効果をまとめる。

１）ＯＳやアプリケーションの機能追加や変更は、対応するオブジェクトを追加あるいは変更するだけで、他の部分が動作中にも実行することができる。

２）システム内の計算機負荷が一様になるように、オブジェクトの計算機間移動や、オブジェクトへの負荷割当てを行うことにより、計算機資源が有効に活用できる。

３）ＯＳやアプリケーションの処理が複数の計算機上で並列に実行できる。

４）要求される信頼度に応じて、オブジェクトを任意の計算機に任意の数だけ多重化できる。またオブジェクトの多重度変更はシステム動作中にも行なえる。

５）負荷の変動に応じて、計算機をシステム動作中にも増減することができる。それに伴なうオブジェクトの移動も２）と同じようにできる。

６）計算機やオブジェクトの保守を、他の部分の動作中にも行なえる。

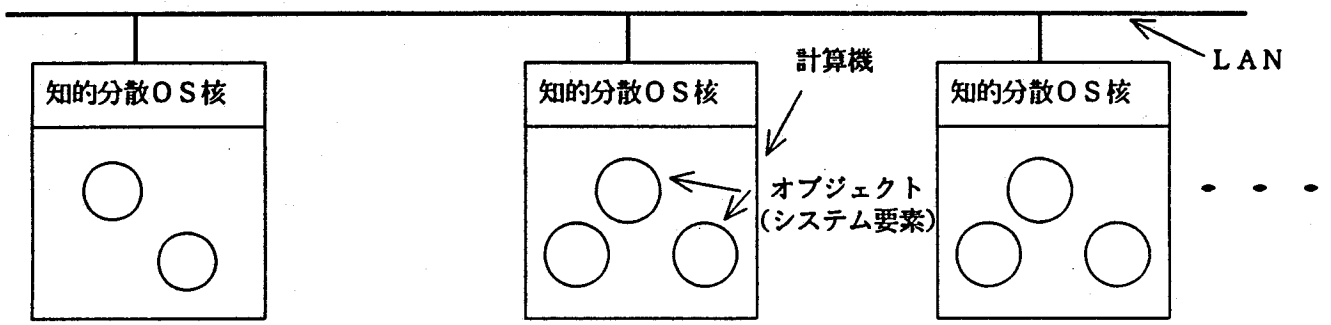


図 2.4　知的分散システムの構成

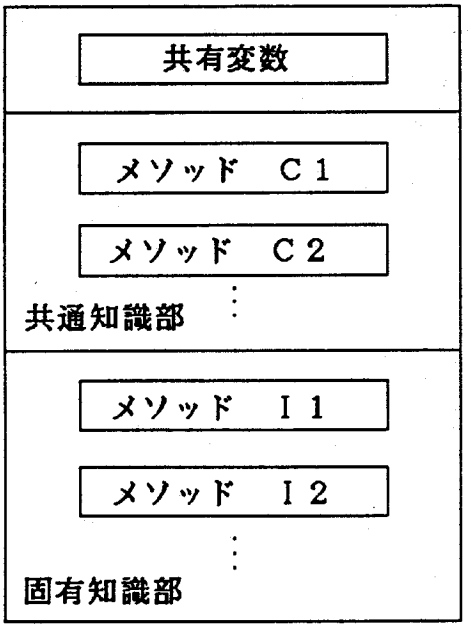


図２．５　オブジェクトの構造

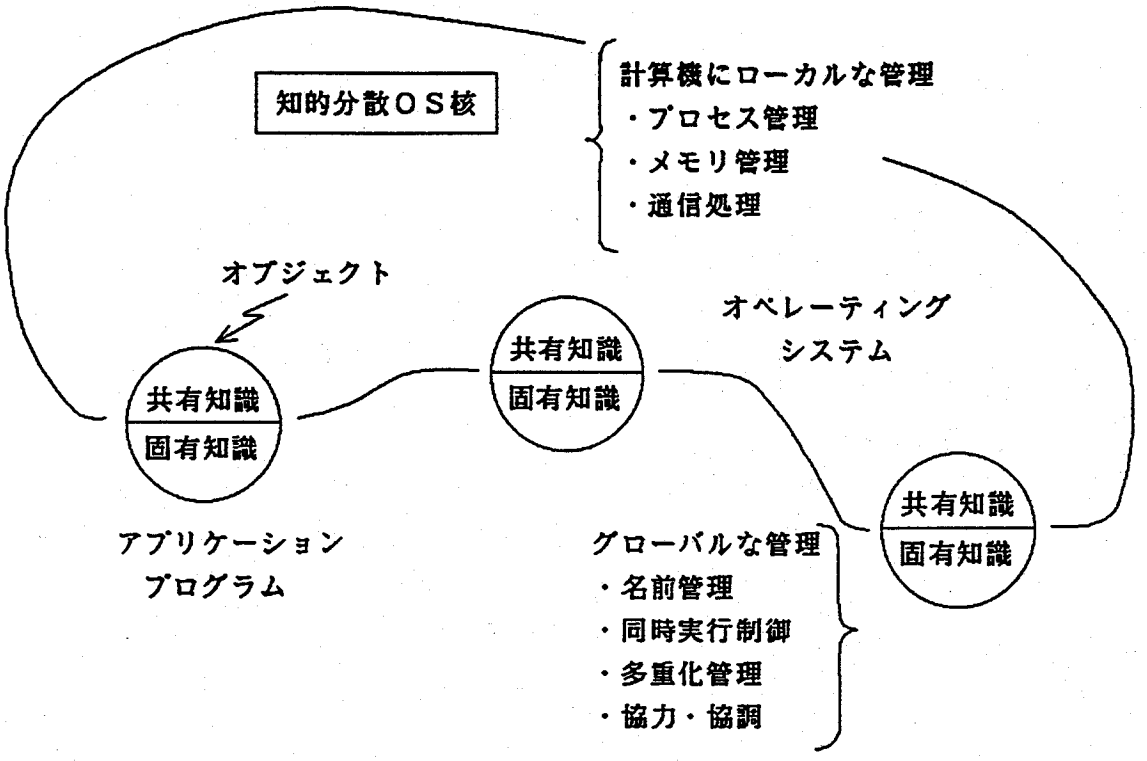


図２．６　知的分散ＯＳの構造

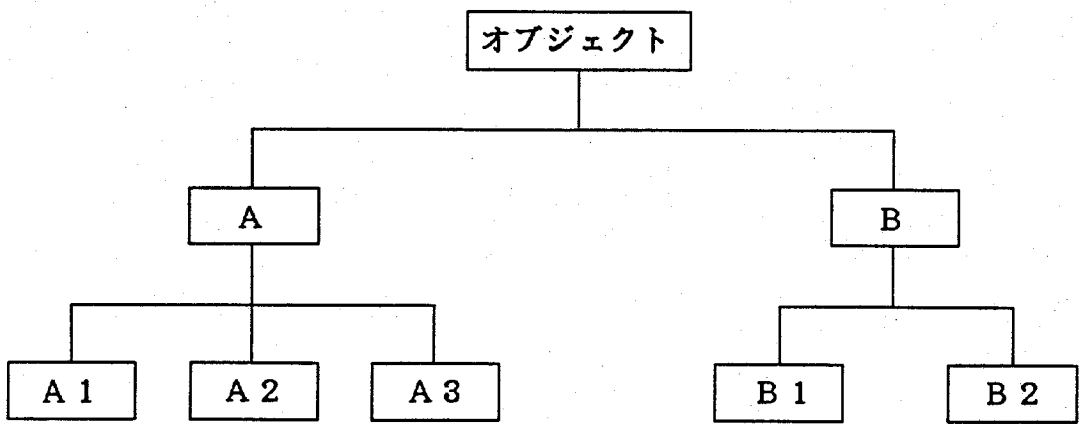


図２．７　オブジェクト名の階層

第５節　考察

分散システム構築の主目的の一つに大規模システムの実現があるが、その時問題になるのは要素間の通信である。例えば計算機群をＬＡＮで結合する場合を考えると、現在の通信技術は400Mbit/secぐらいの大量、高速のデータ転送を可能にしているが、それでも単一のＬＡＮに接続できる計算機の数には限界がある。これを解決するのはシステム要素のグループ化と階層化である。

多くの要素から成るシステムであっても、個々の要素が常時データ交換する相手の要素群は一般的には局所化できる。そこで常時データ交換を行なう要素のグループに専用の通信路を設けることにより、単一の通信路への負荷集中が無くなり多数の要素の接続が可能になる。もちろん頻繁ではないがグループ外の要素とのデータ交換も有るので、ある程度の通信遅れを許しても何らかの形で異なるグループの要素間での通信ができるようにしておかねばならない。また要素の多様な能力を活用するには、図２．８に示すように、同一の要素が複数のグループに属せるような構造が望ましい。さらに状況に適応してグループを形成するような機構が必要である。必要なグループ構成は第３章で述べるような方法によって状況に応じて分散的に求めることができる。

システム要素の階層化はグループ化の一種であるが、大規模システムの構築には不可欠の概念である。階層化ではシステム中の情報を集約、抽象化することにより、要素間でのデータ交換量と個々の要素の処理量を削減する。つまり、大域的な要素群の動作管理と局所的な要素群の動作管理の機能を分離し、より大域的な動作管理に関する機能を上位に位置する要素群に割当てる。そして下位要素群が自身に関するデータを集約、抽象化して上位要素群に渡すことにより、上位要素が多くの下位要素群からデータを受取ってもその総量が一定の範囲を超えないようにする。図２．９に階層化された知的分散システムの構成例を示す。通常の階層化システムとの違いは、

通常システムでは図２．１０のように単一の上位要素がそれに直接つながる全ての下位要素群の動作を管理しているのに対して、図２．９では直接下位のグループ群への動作指示を上位要素のグループが行なっている点である。

本章および次章以降では、全ての要素が同一グループあるいは同一階層に属するシステムを考えている。多くの議論は複数のグループや階層から成るシステムの場合にも単純に拡張可能であるが、要素間の放送通信に関しては多くの検討が必要になる。例えば単一グループから成るシステムでは、ＬＡＮ等を用いれば、複数の要素の発する放送メッセージを全ての要素が同一の順序で受取る機構等は簡単に実現できるが、複数グループから成るシステムでの実現は難しい。

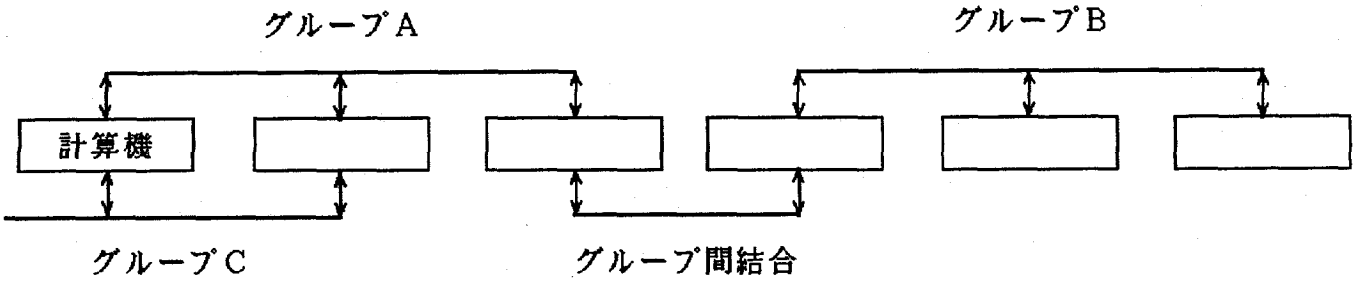


図２．８　計算機のグループ化

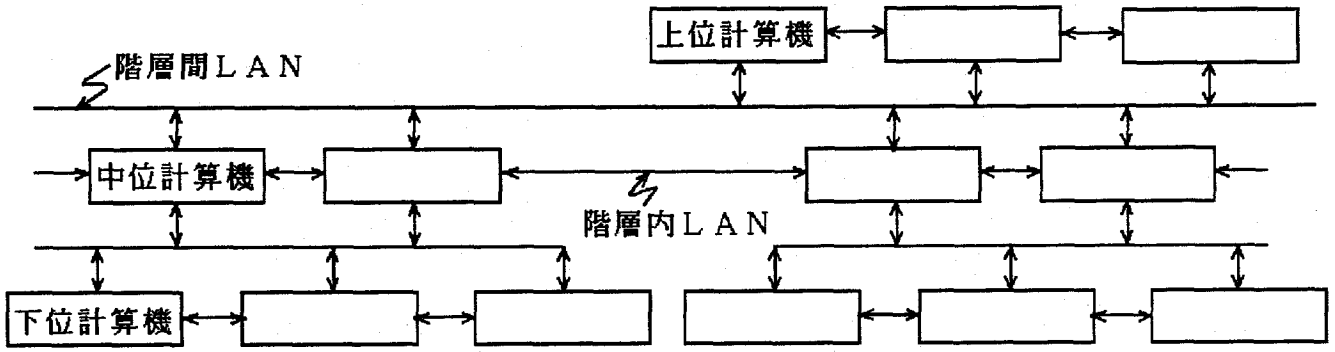


図２．９　階層化知的分散システム

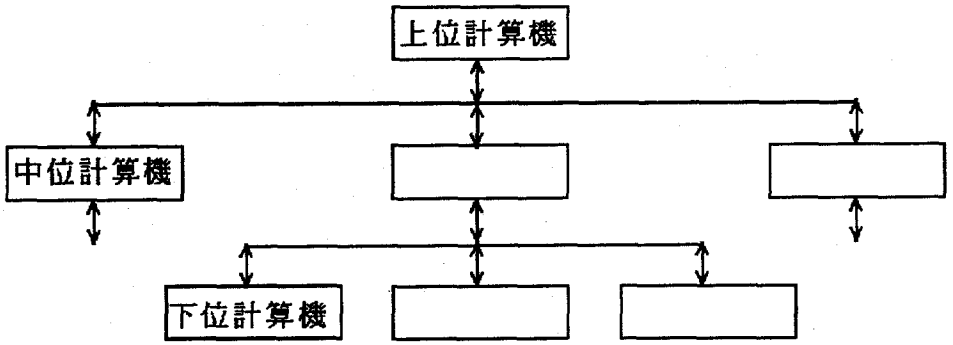


図２．１０　通常の階層化分散計算機システム

第６節　結言

本章では知的分散システムの構造として宣言形のシステム・アーキテクチャを提案した。宣言形のシステムは、各々が他と独立に定義されるシステム要素の単なる集合であり、これら要素群の管理は要素群自身が互いに協力、協調することによって行なわれる。要素の他からの独立性は、個々の要素が自律性を備えることによって実現した。即ち要素の自律性とは、要素が自身の内部状態の参照や変更に必要な機能を全て備え、さらに自身の外部に関する情報や機能を極力持たないことであって、これによって要素は他の要素から独立して、また自身の存在位置等からも独立して動作できるようになる。また従来は要素間の協力と協調の過程は明確に区別されていなかったが、ここでは協力を同じ目的を達成するために要素群が役割りを分担すること、協調を異なる目的間の競合を解消するために要素群が互いにその要求を調整することとして、２つの概念を分離した。従来は２つの概念を同時に実現するシステムは少なかったが、この分離によってその実現が容易になった。

宣言形システムの要素群が効率的にかつ矛盾なく動作するには、次章以降で示す協力、協調や、同時実行制御、多重化要素管理等のための分散アルゴリズムの開発が不可欠であるが、それらアルゴリズムの実現によりシステムから集中管理機構を完全に排することが可能になる。

第３章　システム要素の協力

第１節　序言

システムに投入される仕事の要求仕様は、一般には単独のシステム要素ではなく、複数の要素の結合によって満足される。知的分散システムでは、個々の要素上に分散した協力機構が要求仕様を実現するこの要素結合形態を発見する。この時、宣言形システムの条件を満たすには、協力機構においても個々の要素が他の要素から独立に定義かつ動作できるようになっていなければならない。本章ではこれらの条件を満たす要素群の協力機構として、プロダクション・システムを分散化した機構を提案する。提案する機構によって、要素の追加や変更、移動等が他の要素から完全に独立に行なえるようになる。

第２節では、要素群の協力を実現する分散プロダクション・システムの機構とその基本的な手順を提案し、第３節で手順実行において各要素、およびそれらの間で交換されるメッセージが備えるべき情報の詳細を述べる。また基本手順では、要求仕様の入力は何回でも重複して利用できると仮定しているが、入力の重複利用が許されない場合への手順の詳細化も行なう。第４節では提案手順実行の効率化を行なう。基本手順は、要求仕様を満足する要素群の結合形態を全数探索によって求めるが、不必要な探索を省略することによって処理を効率化する。第５節では、提案する機構の宣言形システムへの適合性を考察する。

要素群の協力形態を求める機構として、既に黒板モデル［１２］やContract Net［１３］［１４］等が知られている。しかし黒板モデルでは、全ての要素の情報アクセスを集中機構が管理しなければならなかった。またContract Netは集中機構の存在は仮定しないが、仕事の要求仕様を実現する単独の要素を発見するだけで、要求仕様の実現に複数の要素の結合が必要になる場合には適用できなかった。

本章で提案する機構によって、集中機構を仮定することなく複数の要素結合形態の発見ができるようになる。

第２節　分散プロダクション・システム

システムに与えられたジョブの要求は単独の機能によって満足されるのではなく、一般には複数の機能の組合わせによって満足される。知的分散システムでは、要素間に分散した協力機構がジョブ実行に必要な機能の組合わせを発見する。具体的にはジョブ要素の要求仕様の実現に必要な機能仕様を持つプロセス要素群が、ジョブとプロセスが互いにその要求仕様と機能仕様を交換することによって、分散的に求められる。要素群の協力機構としては、既に分散協調形エキスパート・システムの分野で、黒板モデル［１２］やContract Net［１３］［１４］などが提案されている。しかし黒板モデルでは、分散要素間の情報伝達が全て黒板を経由して行なわれるので黒板に負荷が集中する。黒板には全ての情報を蓄積するだけの容量が必要になるし、全ての要素が実行する情報読み書きの動作を、デッドロックの発生防止等の機能も含めて矛盾なく制御する能力が必要である。つまり黒板モデルはある意味では集中管理機構を持つシステムであって、宣言形のシステムには適さない。Contract Netでは、ジョブが放送する要求仕様にそれを実現可能なプロセスが応答する形で、ジョブ実行に必要なプロセスを求める。従ってこの方法には集中管理機構は存在しないが、ジョブが単独のプロセスで実行できる場合にしか適用できない。ここでは集中管理機構を仮定せずに複数のプロセスの協力形態を求めるために、プロダクション・システムを分散化した方法を提案する。

提案する協力機構は、ジョブ要素とプロセス要素が放送通信によって交換する要求仕様と機能仕様の単純な照合操作から成り、基本的にはプロダクション・システムの推論機構と同一である［１９］。ただし、ルールはルール・ベースに集中せずに個々のプロセス要素上に分散し、また推論を行なう推論エンジンのような集中機構も存在しない。個々のプロセス要素の記憶する機能仕様がプロダクション・システムにおけるルールに対応し、ジョブ要素およびプロセス

要素間で互いに交換されるメッセージがワーキング・メモリ上の事実に対応する。推論に当る要求仕様と機能仕様の照合操作は全て個々の要素上で非同期かつ分散的に進行する。次に要素群の協力の手順を示す。尚、ジョブ要素の要求仕様およびプロセス要素の機能仕様は、図２．３で示したように入出力の組として表わされる。

［要素群の協力の手順　Ａ］

(1) ジョブがその要求仕様［入力Ｘ，出力Ｙ］を、プロセス群に放送する。要求仕様は一般に多入力、多出力であるので、ＸとＹはそれぞれ入力あるいは出力の列｛ｘ１，‥‥，ｘｍ｝，
｛ｙ１，‥‥，ｙｎ｝で表わされる。

(2) 放送された入力の部分列Ｘ’＝｛ｘｋ|１≦ｋ≦ｍ｝を機能仕様の入力列の成分に持つプロセスＰがそれを受取り、それと既に受取り済みの入力の成分列Ｘｐを合併した列Ｑ＝｛Ｘ’，Ｘｐ｝を作る。ここでＰは、Ｑの成分がＰの機能仕様の入力列と完全に一致しない場合は、受取り済みの成分列Ｘｐをで置換えるだけであるが、一致する場合は、更にＰの機能仕様の出力列Ｒを生成する。そしてＲをジョブの出力仕様であるＹの成分に一致する部分列Ｒ１と、一致しない部分列Ｒ２に分離して、Ｒ１はジョブの要求出力であるのでジョブに返送する。またＲ２が空でない場合は、ジョブの要求仕様の入力であるＸをＲ２で、また出力ＹをＹからＲ１を除いたもので置換えて、再び要求仕様［入力Ｘ，出力Ｙ］をプロセス群に放送する。

上述の手続きで、ジョブがその要求仕様の出力の全成分を受取るまでに経由したプロセス列が、要求仕様の実現に必要なプロセス群と、それらプロセス群の結合形態である。図３．１の例では、入力仕様｛Ａ１，Ａ２｝、出力仕様｛Ｘ，Ｙ｝を持つジョブＪに対して、｛Ｐ１，Ｐ４，Ｐ５，Ｐ７｝　｛Ｐ２，Ｐ３，Ｐ６｝から成るプロセス列が求まる。次にこの手順が正しいことを示す。

［定理３．１］

任意のプロセス要素の集合Ｓと要求仕様Ｒが与えられた時、ＲがＳに属するプロセスの系列で実現可能であるならば、その系列は手順Ａが生成するプロセス系列の集合に必ず含まれる。逆に要求仕様の入力およびプロセス要素の出力が重複して利用できる場合は、手順Ａが生成するプロセス系列は全て要求仕様Ｒを実現する。

（証　明）

手順Ａでは、各プロセスはその出力の生成に必要な入力をジョブあるいはプロセス群から受取ると、出力をＳに属する全てのプロセスに放送する。さらに各プロセスは受取った入力を手順が終了するまで記憶している。従って、手順Ａは与えられたジョブの入力から出力を生成する全てのプロセス系列を網羅する。これはＲを実現するＳに属するプロセスの系列が存在するなら、それが手順Ａが生成するプロセス系列の集合に含まれることを示す。逆に手順Ａでは、各プロセスはその出力を、出力生成に必要な全ての入力を受取った後で生成、放送する。従って生成したプロセス系列中のプロセスが互いに入力待ちになってデッドロックを発生することはない。またプロセスは受取った入力を重複して利用できるので、各プロセスが手順終了まで放送した全ての出力は実際に生成可能である。以上のことより、手順Ａの生成したプロセス系列が実際にＲを実現することがわかる。　□

ここで「プロセスが要求仕様の入力を重複して利用する」とは、手順Ａの生成したプロセス系列の中で要求仕様の同一の入力が２度以上使われることを意味する。ジョブの要求仕様の入力がデータベース中のデータのようなものである場合は、プロセス系列中のプロセスが同一の入力を何度も重複して受取ることができるが、物理的な材料から製品を組立てるような場合は、複数または同一のプロセ

スが同一の入力を重複して受取ることは許されない。またたとえ異なる入力であっても、それらが同一の入力から生成された出力に当る場合には、それらを共に受取ることは入力の重複利用とみなされる。図３．１では、プロセスＰ６への入力ＤとＥはいずれも同一の入力Ａ２から生成されたものであるから、このプロセス系列は同一の入力を重複利用したことになる。但し同一の入力から生成された出力に当るものであっても、それらが同一の入力を分割した結果の異なる部分である場合は、それらを共に受取っても重複利用にはならない。図３．１でＰ７への入力ＦとＧはいずれもＡ１から生成されるものであるが、それぞれＰ１がＡ１を分割した結果の異なる部分であるＢとＣが加工された物に当るのでＰ７は入力を重複して利用することにはならない。入力の重複利用防止については次節で述べる。

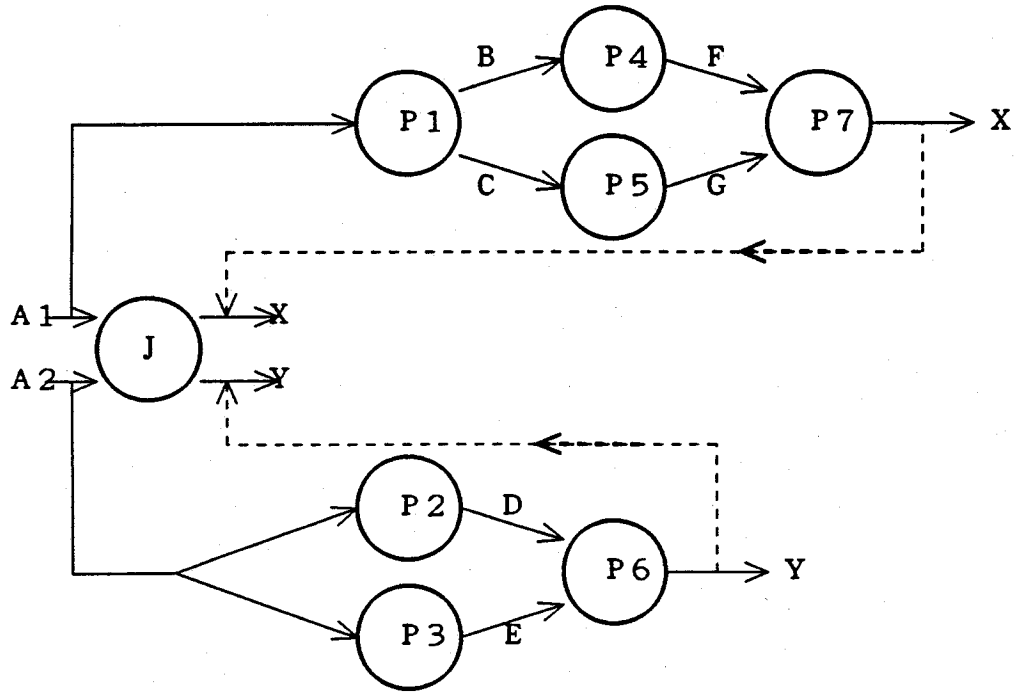


図3.1 プロセス列合成の例

第３節　協力手順の詳細化

第２節で示した手順Ａを実際に使用するには次のような問題がある。ここでは、これらの問題を解決するために手順を詳細化するとともに、そこで必要になる情報を明らかにする。

１）同時に複数のジョブが要求仕様を放送した時、異なるジョブに対応する手順の操作が互いに交錯する。
２）プロセス系列の無限ループができて、手順が永久に終了しない。
３）プロセス系列が、重複利用の許されていない入力を重複して利用する。

図３．２、図３．３は、これらの問題を解決する手順において、プロセス要素あるいはそれらの間で交換されるメッセージが含むべき情報を表わしている。メッセージはジョブＩＤ部、入出力部、パス部から成る。ジョブＩＤ部には、メッセージ送出の原因となる要求仕様を送出したジョブの識別子が格納される。ジョブ発生時刻とジョブを発生した計算機の識別子の対をジョブＩＤにすることにより、システム中で一意に定まる識別子を個々の計算機の局所的な判断で発行することができる。ジョブが最初に要求仕様を放送する時には、メッセージの入出力部にはジョブ要求仕様の入力名と出力名の組が格納される。以後メッセージがプロセスを経由する毎に入出力部の入力名がプロセス機能仕様の出力名で置換えられる。パス部にはメッセージが経由したプロセスの系列が格納される。ジョブはプロセスから目的の出力を受取った時、パス部を参照することによって出力生成に必要なプロセス系列を再生することができる。

各プロセスはその機能仕様として、ジョブＩＤ部と入力部、出力部から成る変換テーブルを持つ。即ち、プロセスは変換テーブルの入力部に記述された全入力を受取ると出力部に記述された出力を生成する。図３．３の例では、入力Ａ，Ｂ，Ｃを受取ると出力Ｘ，Ｙ

を生成する。

プロセスのメッセージ受取り後の処理は次のように進む。まず受取ったメッセージ入出力部の入力名に、自身の変換テーブルの入力部に記述されたものと一致する成分が無い場合は何もしない。一致する成分のある場合は、変換テーブルの一致成分に対応するジョブＩＤ部にメッセージのジョブＩＤ部の内容ｘを書込む。さらにｘが変換テーブルの全入力成分に対応するジョブＩＤ部に格納された時には、受信メッセージ入出力部の入力を変換テーブルの出力成分で置換えて、さらにパス部に自プロセスを追加してプロセス群に放送する。図３．３では、変換テーブルの全入力成分Ａ，Ｂ，Ｃに対応するジョブＩＤ部にＪ１が格納されたので、つまりジョブＪ１から全入力Ａ，Ｂ，Ｃを受取ったので、ジョブＩＤとしてＪ１を持ち、入出力部の入力にＸ，Ｙを持つ出力メッセージが送出される。

異なるジョブに対応する手順の操作交錯は、このようにプロセス間で交換されるメッセージと各プロセス中の変換テーブルにジョブＩＤ部を設けることによって防止することができる。つまり各プロセスは、同一のジョブＩＤを持つメッセージ群によって入力条件が満たされた時にだけ出力メッセージを送出するので、同時に多数のジョブが投入された場合でも、異なるジョブの送出したメッセージが交錯することはない。またプロセス系列の無限ループも、プロセスが同一の入力を、同一のジョブＩＤを持つメッセージとして、２回以上受取るのを検出することによって防止できる。

以上はプロセスが入力を重複して利用できる場合であるが、入力の重複利用が許されない場合も、入力の名前を次のよう拡張することにより同様の扱いができる。入力の拡張名は図３．４に示すように、入力名部とパス部、パスデータ部から成る。図３．４は図３．１に現われる全ての入力の拡張名に対応している。入力名部は拡張する前の入力名を表わし、パス部はその入力が生成されるまでに経由したプロセスの系列を表わす。但し、系列の先頭はジョブの入力成分である。パス部の内容は図３．２に示したメッセージ中のパス

部と同一である。パスデータ部は、その入力が生成されるまでにジョブの入力仕様がプロセス系列を経由して変化した過程を表わす。図中の入力Gのパスデータ部は、入力Gがジョブ入力仕様の第1成分が最初に経由したプロセスの出力第2成部に変換され、さらに次に経由したプロセスの出力第1成分に変換された結果生成されたものであることを表わしている。尚、パス部の内容は、入力X，Yのように2つ以上の入力の合成によって生成される場合は樹木状のプロセスの系列になる。

入力の重複利用は、各プロセスが変換テーブルのジョブID部にジョブ識別子とともに受取った入力の拡張名を記憶し、また出力メッセージ入出力部に入力の拡張名を書き込むことにより防止できる。つまり各プロセスは変換テーブルの全入力成分を受取った時、それらの入力の拡張名を比較し、それらが一致する場合はその組合わせを禁止する。但し2つの拡張名は、それらのパス部の先頭からの連続する部分列に一致するものがあり（これを一致部分列と呼ぶ）、かつ2つの拡張名の一致部分列に対応するパスデータ部が互いに等しい時一致すると定義する。図3．1の例では、プロセスP6の受取る2つの入力DとEの拡張名は一致部分列{A2}を持ち、また{A2}に対応するパスデータ部が共に{2}となって一致するので、これらの組合わせは禁止される。実際これらを組合わせることは、ジョブの入力成分A2を重複利用したことになる。プロセスP7の受取る入力FとGの拡張名には一致部分列{A1,P1}が存在するが、それに対応するパスデータ部がFの場合は{1,1}、Gの場合は{1,2}となって異なる。従ってFとGは、ジョブの入力成分A1を分割した異なる部分から生成されたものと判断され、その組合わせはA1の重複利用にはならない。拡張名の比較によって入力の重複利用が防止できることを示すには、次のことを示せばよい。

［定理３．２］

２つの入力の利用が重複利用になる場合は、それらの拡張名は一致する。逆に２つの入力の拡張名が一致するのは、それらの利用が入力の重複利用になる場合だけである。

（証　明）

２つの入力ＡとＢの利用が重複利用になる場合は、ＡとＢを生成するプロセスの系列である樹木のいずれかの列（１つの葉からルートに至るまでの分岐のないプロセスの系列）が、共通の入力成分を重複利用している。またそれらの列をα，βとするとαとβには必ず一致部分列が存在する。なぜなら一致部分列の無い場合は、αとβは異なる入力成分の変化過程を表わす分岐、合流の無い列であり、αとβの中にどのようなプロセスが含まれようともそれらが同一入力を重複利用したことにならない。同様にαとβの一致部分列に対応するパスデータ部が異なる場合も、αとβはあるプロセスが同一の入力成分を分割した結果の異なる部分の変化過程を列わす分岐、合流の無い列になるので、それらが同一入力を重複利用することはない。これは２つの入力ＡとＢの利用が重複利用になるには、ＡとＢの拡張名が一致しなければならないことを示している。逆にＡとＢの拡張名が一致する場合には、ＡとＢを生成するプロセス系列に一致部分列があり、それに対応するパスデータ部が等しいので、ＡとＢに同一のジョブ入力成分の同一部分から生成された結果が含まれるのは明らかである。つまりＡとＢの拡張名が一致すると、それらの利用は入力の重複利用になる。　□

重複利用の許されない入力のある場合は、プロセス系列の無限ループ防止の方法にも修正が必要である。つまり、プロセスが同一の入力を２回以上受取るのを無条件に禁止すると、実際には生成可能な出力が入力の重複利用と判定されて生成できない場合がある。例

えば入力ＡとＢから出力Ｃを生成するプロセスＰを考える。Ｐが先に入力Ａと、Ａと同一の拡張名を持つＢを受取ってしまうと、後でＡと異なる拡張名を持つ入力Ｂが来てもＰはそれを受取れず、実際には生成可能な出力Ｃが生成できなくなる。このような事態を防ぐためには、プロセスは一度受取った入力であっても、その拡張名が既に受取った入力の拡張名と異なる場合は受取らなければならない。

ジョブＩＤ部	入出力部	パス部

図 3.2　メッセージ形式

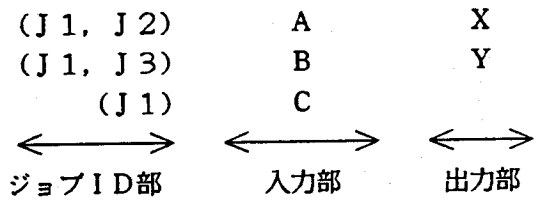


図 3.3　プロセスの変換テーブル

入力名	パ　　ス	パス・データ
B	[A1, P1]	[1, 1]
C	[A1, P1]	[1, 2]
D	[A2, P2]	[2, 1]
E	[A2, P3]	[2, 1]
F	[A1, P1, P4]	[1, 1, 1]
G	[A1, P1, P5]	[1, 2, 1]
X	[{(A1, P1, P4) (A1, P1, P5)}, P7]	[{(1, 1, 1) (1, 2, 1)}, 1]
Y	[{(A2, P2) (A2, P3)}, P6]	[{(2, 1) (2, 1)}, 1]

図 3.4　入力の拡張名の例

第４節　協力手順の効率化

手順Ａは、要求仕様を実現するプロセスの系列を全数探索によって求める。従って結果的に不必要な探索操作が多く、プロセス数が増加すると探索効率が急速に低下する。そこでここでは、各プロセスに隣接プロセス集合と可到達出力集合を持たせることにより探索効率の向上を図る。但し隣接プロセス集合とは、そのプロセスの出力を受取り可能なプロセスの集合であり、また可到達出力集合とは、そのプロセスから他のプロセスを経由して到達できるプロセス群が生成可能な全ての出力の集合である。各プロセスはその出力を全てのプロセスに放送する代りに、その隣接プロセス集合にだけマルチ・キャストすることにより、その出力の受取り不可能なプロセスまでが処理をすることがなくなる。さらに、各プロセスの出力はどのようなプロセスの系列を経由しても可到達出力集合に含まれる出力しか生成しないので、その可到達出力集合にジョブの要求出力を含まないプロセスが処理を中止することによって、不必要な探索を削減することができる。

システムの変更や拡張によってプロセス要素の種類や数は変化する。従って、システムの柔軟性を保つには、隣接プロセス集合と可到達出力集合もプロセス要素の増減に伴って動的に更新できなければならない。これらの集合の更新を効率的に行なうために、各プロセスは逆方向隣接プロセス集合、可到達プロセス集合、逆方向可到達プロセス集合も備える。プロセスＰの逆方向隣接プロセス集合とは、Ｐを隣接プロセス集合の要素に持つプロセスの集合である。またＰの可到達プロセス集合とは、Ｐの出力を受取り可能なプロセス、およびそれらのプロセスを経由したプロセスの出力を受取り可能な全てのプロセスの集合であり、逆方向可到達プロセス集合はＰをその可到達プロセス集合の要素に持つプロセスの集合である。システムにプロセスが追加あるいは削除された際の、これらの集合を用いた隣接プロセス集合および可到達出力集合の更新手順は、第５章で

述べるデッドロック検出、防止のための分散アルゴリズムと同じであるので省略するが、プロセスの総数をｎとすると、そのために必要な通信回数はｎのオーダーに、また総処理量はn^3のオーダーに納まる。但し更新手順は各プロセス上で並列に実行できるので、もし各プロセスが異なる計算機上に実現されていれば、処理に必要な時間はn^2のオーダーに抑えることができる。各プロセスの持つ隣接プロセス集合等の大きさもｎを超えることはない。

第５節　考察

第２節で述べた手順Ａは、次のような意味で第２章３節で示した性質1)～3)を満たす。つまり各プロセス要素は、他のプロセス要素との交信に必要なプロトコルを知るだけで、他のプロセス要素がどのような機能を持つか、どこに何個存在するかなどの情報を持つ必要はない。ここでプロトコルとは、図３．２に示したメッセージ構造と、そのメッセージの受信メソッド等のことである。またプロセス群の協力形態であるプロセス系列を求める手順も機能分散形態ではない。即ち、処理は全体で同期をとることなく並列に進行するし、あるプロセスの故障によって全体が停止することもない。またプロセスが追加されても既存のプロセスを変更する必要は全くない。

第４節で述べた手順Ａの効率化では、各プロセスが隣接プロセス集合や可到達出力集合等を持つ必要があり、必ずしも先に掲げた1)～3)の条件を完全に満足するとは言えない。つまり各プロセスは他のプロセスの存在やその能力を知らなければならない。しかし、それらの情報は外から与えられるのではなく、プロセス自身がシステムの変化に適応して構築するものである。従ってそれらの情報を格納するための記憶領域が必要になることを除けば、システムの信頼性や柔軟性、さらには処理性等の面で問題になることは無い。しかし、本章ではジョブ要素やプロセス要素の入出力仕様が変数を含まない場合しか考えなかったが、可到達出力集合等の概念を、より広範な協力機構として入出力仕様に変数を導入する場合にまで拡張するのは難しい。従って別の手段による効率化の研究が必要である。

第６節　結言

本章ではシステム要素群の協力機構として、プロダクション・システムを分散化した方法を提案した。既に黒板モデル［１２］やContract Net［１３］［１４］などが同様の目的で提案されているが、それらには各々集中管理機構に相当する部分が必要になるとか、複数の要素の結合が扱えない等の欠点があった。ここで提案する方法によって、単独の要素では実現できない要求仕様を満足する要素群の結合形態を、集中機構を全く仮定しないで発見できるようになった。また提案方法は宣言形のシステムの条件を十分に満たしている。即ち、個々の要素には、他の要素の存在や機能に関する知識は要求されず、従って要素の追加や変更、移動等を他の要素から全く独立に実行することができる。

尚本章では、ロボットの軌道生成のようにシステム要素の出力の連続的な変化の結合は扱わなかった。またジョブ要素の要求入出力仕様の量的な扱い（例えばｘ個の入力Ａとｙ個の入力Ｂからｚ個の出力Ｃを生成する等）は考えなかった。後者の扱いは本章で提案した方法の直接的な延長で解決可能と思われるが、前者の扱いは異なるアプローチも必要と予想される。第５節で述べた提案方法の効率化の問題とも合わせて今後の課題としたい。

第４章　システム要素の協調

第１節　序　　言

第３章では、ジョブ要素の要求を実現するプロセス要素（機能）の組合わせを、ジョブ要素とプロセス要素群上に分散した協力機構が発見したが、個々のプロセス要素の表わす機能を実行するのは実際には資源要素である。従ってシステム全体の処理効率を高めるには、プロセス要素の機能を実行する資源要素の割当てを適切に決定しなければならない。本章では知的分散システムの協調機構として、この資源割当ての方法を考える。効率的な資源割当て法に関しては、スケジューリング問題として古くから多くの研究がなされている。しかし、これらの研究には、定められた期間中に発生する全てのジョブ群にあらかじめ資源を割付けてしまう静的スケジューリングや、あるいはスケジューリングのために集中機構を仮定する方法が多く、宣言形のシステムに適する方法は少なかった。宣言形のシステムは本来動的なものであり、スケジューリングにもジョブの発生毎に資源割付けを行なう動的な方法が必要になる。また、それらはもちろん個々の要素上に分散した機構でなければならない。

そこで、ここでは個々のプロセスと資源要素上に分散した動的スケジューリング法を提案する。第２節で単段スケジューリングを、第３節では多段スケジューリングを扱う。ジョブ要素の要求仕様は、プロセス要素の系列に分解されるので、現在実行可能なプロセス要素群の要求だけを考えて資源を割付けるよりも、将来実行すべきプロセス間の競合をも考慮して資源を割付ける方が資源の利用効率が高まる。ここでは前者を単段スケジューリング、また後者を多段スケジューリングと呼ぶ。多段スケジューリングは一般に大量の計算を必要とするので動的な資源割付けに用いることは難しく、計算機システム等の資源割付けには単段スケジューリング法が採用されることが多い。しかし機械加工システムなどでは、個々の機械加工に

比較的長時間を要し、また実際に加工を行う機械と機械群を管理する計算機が別のハードウェアになることが多いので、機械の加工中にスケジューリング計算が可能になり、多段スケジューリングが有効になる場合がある。

第２節では、発見的な方法による単段スケジューリングの分散機構を提案する。単段スケジューリングの最適アルゴリズムとしては、ハンガリア法がよく知られているが［２０］［２１］、そこでは個々の分散要素がシステム中のプロセスあるいは資源要素の総数を知る必要がある、あるいはシステム中の要素全体が同期して動作しなければならないなど、宣言形システムの協調機構としての条件（第２章３節で示した３つの条件）は完全には満たされない。また最適スケジューリングの算出に必要な計算量もプロセスあるいは資源要素の総数をｎとする時、n^3のオーダーとなり必ずしも少なくない。そこでここでは、最適スケジュールは生成しないが、より高速でかつ宣言形システムの条件を備えた方法として両方向ビディング（bidding）法を導出する。同様の方法として、既にビディング法があるが［２２］、ビディング法では負荷が特定の資源に集中する場合のある欠点があった。両方向ビディング法はこの欠点を改善する。

第３節は、多段スケジューリングの分散機構を扱う。多段スケジューリングに関しては、全ての資源が同じ能力を持っている等の特殊な場合には効率的な最適化アルゴリズムが提案されているが［２３］［２４］［２５］、一般の場合には分枝限定法やディスパッチング・ルール等があるだである［２６］～［２９］。しかし、分枝限定法は本質的には全数探索に基づく方法であり膨大な計算量を必要とする。一方ディスパッチング・ルールは個々の資源が発見的、経験的に設定された規則に基づいて各々独立に処理すべき仕事を決定するものであり、宣言形システムに適した方法ではあるが、スケジュールの最適性に関する保証が全く無い。そこでここでは、ディスパッチング・ルールと同様の性質を持ち、かつスケジュールの最適性に関する保証を高めた方法を提案する。

第２節　単段スケジューリング

単段スケジューリングでは、スケジュール時に実行可能な要求の処理効率だけがなるべく高くなるように、プロセス要素群に資源要素群を割当てる。スケジュール時に実行可能であったプロセスの処理後に実行可能になるプロセス群の処理効率は考えない。最適な単段スケジュールを求める効率的なアルゴリズムとして、ハンガリア法がよく知られており、その分散化も可能であるが［２０］［２１］、それらは次のような意味で宣言形システムの条件を満たさない。つまりハンガリア法では、各プロセスと資源が、システム中の要素の数やそれらの要求あるいは能力、さらには負荷状態など、システム全体の構造と状態を把握していなければならないし、また各プロセスと資源に分散された処理間での大域的な同期が必要になる。従って、個々の要素の自律性が満たされず、大規模で柔軟なシステムの構築には適さない。

最適ではないが十分満足できる資源割当てを、集中機構を仮定することなく発見的・経験的に求める方法として、ビディング（bidding）やディマンド・マネージメント（demand management）、マネージメント・バイ・コンセンサス（management by concensus）法等が提案されているが［２２］、それらにおいてもビディング法を除いては必ずしも個々の要素が自律性の条件を満足することはできない。例えば、分散計算機群の負荷を平準化する場合を考えると、ディマンド・マネージメント法では各計算機が他の全ての計算機の機能をあらかじめ記憶し、かつ他の計算機群からの周期的な報告に基づいてそれらの負荷状態を監視する。そしてある計算機が過負荷になると、過負荷計算機が自身の記憶している他計算機群の状態から、その負荷を実行可能でかつ最も負荷の低い計算機を見つけ、そこへ負荷の再割当てを行なうことにより負荷平準化を実現する。従ってこの方法では、個々の計算機が全計算機の機能と負荷状態を常に監視、把握する必要があり、要素の自律性を保つことができない。

ディマンド・マネージメント法では過負荷計算機が複数存在する時、それらが各々独立に負荷最小の計算機に過剰負荷を再割当てするので、再割当て負荷が最小負荷の計算機に集中する。そこでマネージメント・バイ・コンセンサス法では、システム中の全計算機が同一の意思決定をする機構を備えることによってこれを防止する。つまり、複数の過負荷計算機が存在しても、各計算機が常に他の計算機群の負荷を監視し互いに矛盾のない基準で過剰負荷を再割当てするので再割当て負荷が競合することはない。しかし各計算機の同一の意思決定をする機構によって個々の要素の自律性はさらに損なわれる。

これに対してビディング法では、過負荷計算機が過剰負荷を他の計算機群に放送する。次にそれを受取った各計算機が、通知された過剰負荷を処理する機能を持っていればその負荷状態を返答し、最後に過負荷計算機が最も低い負荷を返答した計算機に過剰負荷を再割当てすることによって負荷の平準化を実現する。ディマンド・マネージメント法等では各計算機が全ての計算機の機能と状態を把握しなければならないが、ビディング法ではこのように、各計算機は自身の能力と負荷状態だけを知ればよく、またシステム全体の状態を常時監視する必要も無い。つまり、ビディング法は第2章3節で示した1)～3)の条件を満足し、個々の要素の自律性を保った宣言形システムの実現を容易にする。しかしビディング法には、ディマンド・マネージメント法と同様に、個々の要素が各々独立に意思決定を行なうので、特定の要素に負荷の集中する欠点がある。分散計算機群の負荷を平準化する例では、最も低い負荷の計算機に再割当て負荷が集中する。そこで本節では、ビディング法のこの欠点を改善した両方向ビディング法を提案する。マネージメント・バイ・コンセンサス法は個々の要素に同一の意思決定をする機構を備えることによって、ディマンド・マネージメント法の欠点を改良したが、両方向ビディング法は個々の要素に同一の意思決定をする能力は要求しない。次に通常のビディング法によるプロセス要素群への資源要

素群の割当て手順を示す。

［ビディング法の手順］

(1)プロセス要素が資源要素群にその機能仕様を放送する。

(2)各資源要素が、受取った機能仕様を処理可能であれば、機能仕様に対する貢献度をプロセス要素に返答する。

(3)プロセス要素が、最大の貢献度を返答した資源要素に処理を依頼する。

(4)プロセス要素から処理依頼を受取った資源要素が処理を実行する。

ここで資源Ｒのプロセスに対する貢献度とは、ＲがＰを実行する時の利得で、ＲによるＰの実行コストや実行時間（Ｒに割当て済みのプロセス処理時間も考慮する）等に反比例する量として定義される。上述の手順では、(3)において個々のプロセスが各々独立に特定の資源を選択するので、複数のプロセスが存在する時、それらの処理負荷が貢献度最大の資源に集中する恐れがある。そこで通常は、特定資源への負荷集中が解消されるまでビディングを繰返したり、同時には１つのプロセスに対してしか資源の割当てをしないなどの方法が採られるが、再ビディングや異なるプロセス間での同期のためのオーバーヘッドが大きくなる。両方向ビディング法は、次に示す手順によってプロセス間で同期をとることなく、この問題を解決する。

［両方向ビディングの手順］

(1)プロセス要素Ｐｉが資源要素群にその機能仕様を放送する。

(2)資源要素Ｒｊが、Ｐｉの機能仕様を処理可能であればＰｉを自身の処理依頼リストに追加して、ＲｉのＰｉに対する貢献度ａ(ｉ,ｊ)をＰｉに返送する。

(3)プロセス要素Ｐｉが、資源要素群から受取った貢献度より、各資源要素Ｒｊに対する要求度ｒ(ｉ,ｊ)を計算し、対応する資源要素に通

知する。

(4)Ｐｉからｒ(ｉ,ｊ)を受取ったＲｊが、処理依頼リストから要求度最大のプロセスＰを削除し、Ｐに実行許可を通知する。

(5)Ｒｊから実行許可を受取ったＰは、既に他の資源が割当てられていれば割当て済みを、割当てられていなければ、割当て依頼をＲｊに返送する。

(6)ＲｊはＰから割当て依頼を受取ると、Ｐの処理を開始する。Ｐから割当て済みを受取った時、あるいはＰの処理が終了すると、処理依頼リスト中のプロセスの存在を調べ、プロセスが存在する場合は(4)に戻る。

ここでプロセスＰｉの資源Ｒｊに対する要求度ｒ(ｉ,ｊ)とは、Ｐｉの機能仕様がＲｊでしか実現できない程度であり、例えばＲｊのＰｉに対する貢献度ａ(ｉ,ｊ)を用いて(4.1)式のように定義する。

$$r(i,j) = \frac{a(i,j)}{\sum_{k \in R(i)} a(i,k)} \quad \cdots\cdots\cdots (4.1)$$

但しＲ(ｉ)はＰｉを処理可能な資源の集合である。(4.1)式によれば、ＰｉがＲｊでしか処理できない時にはｒ(ｉ,ｊ)＝１となり、Ｒｊ以外にも多くの資源で処理できる場合にはｒ(ｉ,ｊ)は０に近くなる。従って、ｒ(ｉ,ｊ)がＰｉをＲｊでしか実現できない程度を表わすことがわかる。

通常のビディング法では、個々のプロセスが各々独立に最大の貢献度を持つ資源を選択するので、特定の資源に負荷が集中し易かった。しかし、両方向ビディング法では、各資源がプロセス群の要求度を比較することによって、その資源でしか処理できないようなプロセスを優先し、他の資源でも処理できるようなプロセスの実行は後まわしにするので、特定の資源に負荷が不必要に集中することがなくなる。

図4.1に例を示す。図中の行列は第0列に示したP_1～P_4のプロセスが第0行に示した資源R_1～R_3を要求している状態をを表わし、行列(i,j)　要素の上段はj番目の資源Rjのi番目のプロセスPiに対する貢献度である。＊はRjがPiを実行できないことを示す。通常のビディング法では、各プロセスが貢献度が最大の資源を選択するので、この状態では○で示したような割当てになり、P_2とP_3の実行要求およびP_1とP_4の実行要求が各々R_2，R_3上で競合する。しかし、両方向ビディング法では、プロセスPiが資源群の貢献度から資源Rjに対する要求度を(i,j)要素下段に示したように計算し各資源が要求度が最大のプロセスを実行するので□で示すような割当てになる。従って通常のビディング法で未使用であったR_1が利用され、R_2およびR_3上での競合が解消されることがわかる。このように両方向ビディング法によれば、多数のプロセスからの要求がほぼ同時に発生しても、ビディング法のようにシステム全体での同期をとったり、あるいは再割当てを何度も繰り返すことなくプロセス群に資源を割当てることができる。

両方向ビディング法は、次のような意味で宣言形のシステムでの協調機構としての条件、つまり第2章3節で述べた条件を満たしている。まず、各プロセスおよび資源は、その意思決定に必要な情報を、放送形式の質問に対する回答として他の要素群から受取るので、他の要素の存在や位置などのシステム構造を知る必要がなく、またそれらの状態を常時監視する必要が無い。従って、個々の要素のシステムの大域的な構造からの独立性が保証される。通常の分散アルゴリズムが動作するには、個々の要素が関連要素の存在や個数を常時把握していなければならない。また両方向ビディング法では、例えば要素上の記憶域の制限や要素の故障などのためプロセスがそれを実行可能な全ての資源から貢献度を受取れなくても、処理に矛盾が生じることは無い。関連する全ての要素から貢献度や要求度を受取ることができなくても上述の手順は動作する。さらに、上述の手順では関連する要素群だけがメッセージの交換に伴なってその動作

の同期をとればよく、システム内の全ての要素が歩調を合わせる必要はない。これは協調機構の分散形態が、個々の要素が特定の機能を分担する形ではなく、各々が必要な情報を集めて自律的に動作する形になっているためである。

プロセス ＼ 資源	R1	R2	R3
P1	4. 2 0. 29	3. 6 0. 24	6. 9 ○ 0. 47
P2	1. 3 0. 33	2. 7 ○ 0. 67□	*
P3	3. 1 0. 40□	4. 7 ○ 0. 60	*
P4	2. 6 0. 29	1. 8 0. 20	4. 0 ○ 0. 51□

(注) 上段は貢献度、下段は要求度

図4.1 両方向ビディング例

第３節　多段スケジューリング

単段スケジューリングでは、スケジューリング時に実行可能なプロセス群だけの処理効率を考えればよかったが、多段スケジューリングでは、各々がプロセスの系列から成るジョブ群が与えられた時、それらプロセス系列全体の処理効率が最大になるような資源割当てを考えなければならない。従って、全ての資源が同一の機能と能力を持つような特殊な場合を除いては［２３］［２４］［２５］、単段スケジューリングにおけるハンガリア法のような効率的なアルゴリズムは存在しない。分枝限定法を用いる方法等が多く提案されているが［２６］［２７］、実際には実行負荷が膨大で少なくとも大規模システムでの動的スケジューリングには適さない。しかし、宣言形システムでは非決定的な動作が本質的であり、高速の動的スケジューリング機構が不可欠である。

大規模システムでの動的スケジューリングには、従来からディスパッチング・ルールを用いる方法が広く使われている［２８］［２９］。個々の資源が各々独立に、ＳＰＴ（Ｓhortest Ｐrocessing Ｔime）やＭＷＫＲ（Ｍost Ｗork Ｒemaining）などのルールを用いて、次に処理可能なプロセス群から実際に処理すべきプロセスを選択する方法である。ＳＰＴでは、処理候補プロセス群から処理時間が最短のものを選択し、ＭＷＫＲではプロセス系列が構成するジョブの残り処理時間が最大になるものを選択する。ディスパッチング・ルールによる方法では、このように個々の資源が、自身で処理可能なプロセスに関する情報から各々独立に、次に処理すべきプロセスを決定するので、宣言形システムに求められる多くの性質を満足し、必要な計算量も少ない。しかし、ＳＰＴやＭＷＫＲ等のルールがあまりにも経験的であるため、ルールとジョブ処理効率との関係に必然性が無く、効率的な資源割当てを保証することはできない。そこで本節ではディスパッチング・ルールを拡張して、各資源が処理候補となるプロセスの評価値としてそれを最初に処理した場合の

システム中の全ジョブ終了時刻の下限値を計算し、その評価値が最小になるものを次に処理するプロセスとして選択する方法を提案する。これによって、個々の資源が次処理プロセスを決定するための評価関数の意味が明確になり、従来のディスパッチング・ルールよりも効率の高いスケジュールの生成が可能になる。ここで、各プロセスの評価値は、ある資源からの要求に他の資源が返答する形で計算されるので、提案方式は前節で述べたビディングあるいは両方向ビディング法の一種でもある。従って、個々の要素が互いにその状態を交換することにより、それぞれの動作が自律的また並列的にスケジュールされるので、集中管理機構を排した高い拡張性と適応性、信頼性の維持が容易になる。

3.1 問題の定式化

システムに投入されるジョブJは、第2章で示したように協力機構によってプロセスの系列 $\{P_J(1), P_J(2), \cdots, P_J(n_J) \mid n_J \geqq 1\}$ に分解され、Jは系列中のプロセスがこの順に処理された時終了する。この時、ある時刻において最も早く処理すべきJのプロセスを、その時刻におけるJの先頭プロセス、$P_J(n_J)$をJの最終プロセスと呼ぶ。また、$P_J(i)$に対して、$P_J(i-1)$, $P_J(i+1)$, $P_J(k)(k<i)$, $P_J(l)(l>i)$を各々、直前、直後、先行、後続プロセスと呼ぶ。個々のプロセスの機能は実際には資源が実現するが、資源は同時に2つ以上のプロセスは処理できないし、直前プロセスの終了していないプロセスを処理することもできない。また、個々のプロセスの資源上での処理時間はあらかじめわかっている、あるいは、何らかの方法で予測可能であるとする。さらに、ここでは代替資源の無い場合、つまり各プロセスを処理可能な資源がシステム中に1つしかない場合を考える。代替資源が存在する場合の考察は第4節で行なう。

このような前提で、与えられた全てのジョブが終了する時刻を最小にするスケジュールの生成を考える。つまり、各資源に割当てら

れたプロセス群の処理順序を（各プロセスを処理可能な資源はシステム中に１つしかないので、資源割当ては一意的に決まる）、システムに投入済みの全てのジョブの終了時刻が最小になるように決定する方法を考える。もちろんこの方法は、第２章３節で述べた条件を満足しなければならない。次に本節で必要になる用語の定義をする。

［割当てプロセス集合　Ａｔ(Ｒ)］　時刻ｔに資源Ｒに割当てられているプロセスの集合。

［処理候補プロセス集合　Ｃｔ(Ｒ)］　時刻ｔに資源Ｒが処理可能なプロセスの集合と、これらプロセスの処理時間の最小値をｍとする時、時刻（ｔ＋ｍ）までに処理可能になるプロセスの集合の和集合。Ｒが時刻ｔに、次に処理すべきプロセスとしてＣｔ(Ｒ)以外のプロセスを選択しないことは明らかである。

［後続プロセス処理資源集合Ｓ（Ａ）］　プロセス集合Ａに属するプロセスの後続プロセスを割当てられている資源の集合。

［最終プロセス処理資源集合Ｆ（Ａ）］　プロセス集合Ａに属するプロセスの最終プロセスを割当てられている資源の集合。

［最早開始時刻、最早処理区間］　時刻ｔに未処理であるプロセスＰの全ての先行プロセスが最優先で処理される場合に、Ｐが処理開始可能になる時刻ｔ（Ｐ）を、時刻ｔにおけるＰの最早開始時刻と呼ぶ。また時刻ｔ（Ｐ）から、ｔ（Ｐ）にＰの処理時間を加えた時刻までの間を、Ｐの最早処理区間と呼ぶ。ここでプロセスＰが最優先で処理されるとは、Ｐの直前プロセスが終了して、かつＰを処理すべき資源が現在処理中のプロセスを終了すると、Ｐが待ち時間無しで処理されることを意味する。

［連続処理区間］　時刻ｔに資源Ｒが処理中、および時刻ｔ以後にＲが処理すべきプロセス群の最早処理区間を、互いに重ならないように、しかも各プロセスの処理開始時刻がその最早開

始時刻よりも早くなることのないように移動させて接続した時、その結果できた処理区間の合併を時刻tにおけるRの連続処理区間と呼ぶ。図4.2に連続処理区間の例を示す。

3.2 プロセスの評価関数

提案する方法では、資源Rは次に処理すべきプロセスとして、処理候補プロセス集合Ct(R)中の各プロセスの評価値を計算し、その値が最小のものを選択する。ここで、時刻tにおいて資源RがCt(R)中のプロセスPに与える理想的な評価関数TI(t,R,P)は、時刻tにRがPを処理する仮定の下で、全ての資源が割当てられたプロセス群の処理を終了できる時刻の最小値である。つまり各資源Rが、あるプロセスの処理を終了した時刻tで、次に処理するプロセスとしてTI(t,R,P)を最小にするプロセスを選ぶことにより、システムに投入済みの全ジョブの終了時刻が最小になるのは明らかである。しかし、TI(t,R,P)を求めるには、Rが最初にPを処理する条件下での、全ての資源上でのプロセス処理順序の可能性を考える必要があり、膨大な計算量が必要になる。そこでここでは、TI(t,R,P)を実用的な計算量で算出可能な量で近似することを考える。TI(t,R,P)の近似TA(t,R,P)を［近似１］、［近似２］［近似３］の３段階に分けて定義する。

TI(t,R,P)を求めるには、資源Rが最初にPを処理する条件下での、全ての資源上でのプロセス処理順序に関するジョブ終了時刻の最小化が必要であるが、［近似１］では、各資源R'がRが最初にPを処理する仮定の下で、各々独立にR'に関するプロセス群の終了時刻だけを最小化する。近似値T_1(t,R,P)は、その結果のR'に関する最大値として求まる。

［近似１］　個々の資源R'は、Rが時刻t以後最初にPを処理する条件の下で、R'に割当てられたプロセス群At(R')の最終プロセスを処理する資源群F（At(R')）の全てが処理を終える時刻の最

小値$e_1(t,R',P)$を与えるRおよびR'上でのプロセス処理順序を求める。この時、資源群F（At(R')）に割当てられたプロセスはその最早開始時刻に処理可能になると仮定する。

ただし、At(R)とAt(R')の後続プロセス群の最早開始時刻は、$e_1(t,R',P)$を与えるRおよびR'上でのプロセス処理順序に従って遅らせる。すなわち、R'がプロセスP'の処理開始をその最早開始時刻よりもτ時間遅らせるような順序でプロセスを処理する時、P'のすべての後続プロセスの最早開始時刻もτだけ遅らせなければならない。R'がこのように$e_1(t,R',P)$を求めた時、Rは近似値$T_1(t,R,P)$を次のように計算する。

$$T_1(t,R,P) = \max_{R' \in S(At(R)) \cup R} e_1(t,R',P) \quad \cdots\cdots \quad (4.2)$$

□

上述の近似では個々の資源R'は、資源群F（At(R')）上だけでの処理終了時刻を、しかもAt(R)、At(R')の後続プロセス以外のプロセス（この集合をBt(R,R')とする）は全て最早開始時刻に処理可能になると仮定して最小化する。従って、TI(t,R,P)の算出に比較すると、計算量を大幅に削減できる。また、資源群F（At(R)）上のジョブ終了時刻の最小化だけを考えると、異なる資源の選択した次処理プロセスの後続プロセス群の処理要求が競合し、全体の処理効率の低下することがあるが、［近似1］ではAt(R)の後続プロセスを処理する全ての資源R'に対してF（At(R')）上でのジョブ終了時刻を最小化することにより、後続プロセス群の処理要求の競合も考慮したスケジュール生成が可能になる。

さらにプロセス群Bt(R,R')が最早開始時刻に処理可能になると仮定することにより、(4.3)式が成立することは容易にわかるが、これは［近似1］を用いたスケジューリングが、次のような意味でMWKRルールなどによる方法よりも優れていることを示している。

$$T_1(t, R, P) \leqq TI(t, R, P) \quad \cdots\cdots \quad (4.3)$$

資源Rの処理候補プロセスPが所属するジョブの残り処理時間をu(P)とすると、Rが時刻tにPを処理した後、全ての資源上での処理が終了するまでには、(4.4)式で求まるv(t,P)以上の時間が必要である。

$$v(t, P) = \max_{P' \in Ct(R),\ P' \neq P} u(P') \quad \cdots\cdots \quad (4.4)$$

従って、T'(t,R,P)=v(t,P)+tをTI(t,R,P)の近似と考えることができるが、RがT'を最小にするプロセスを選択するのは、u(P)を最大にするプロセスを選択することに他ならない。つまりMWKRルールは、TIをT'で近似する方法と等価であるが、この時明らかにT'(t,R,P)≦T_1(t,R,P)が成立つので、(4.3)式と合わせると近似値としてはT_1の方がT'よりも優れていることがわかる。即ち、T_1(t,R,P), T'(t,R,P)を最小化するプロセスをそれぞれP_1, P_2とすると、$T_1(t,R,P_1) \geqq T'(t,R,P_1)$, $T_1(t,R,P_2) \geqq T'(t,R,P_2)$が成立するので、Rが$P_1$あるいは$P_2$を選択した場合、全資源上での処理が終了する時刻は各々$w(P_1) = T_1(t,R,P_1)$, $w(P) = T_1(t,R,P_2)$以上であるが、P_1が$T_1(t,R,P)$を最小化することにより $w(P_1) \leqq w(P_2)$となる。しかし、これはT_1を用いるスケジューリングが常にT'を用いるものよりも優れた結果を生成することを保証するものではない。

[近似1]におけるプロセス群Bt(R,R')が最早開始時刻で処理可能になる仮定の下では、RおよびS(At(R))以外の資源R''に対しては、e_1(t,R'',P)の値はPに依存しないかあるいはe_1(t,R,P)と一致する。従って、(4.2)式で実際にe_1(t,R',P)を計算するのは、Rおよび資源群S(At(R))だけで十分である。このように

［近似１］ではe_1を計算すべき資源Ｒ'の範囲は限定できるが、Ｒ'はそこに割当てられた全プロセスの処理順序に関する最小化を実行しなければならない。［近似２］では、個々の資源が最小化に際して処理順序を考慮すべきプロセスの範囲も縮小する。つまり［近似１］では(4.2)式算出時に、各資源Ｒ'が最も都合の良いプロセス処理順序を仮定するが、この処理順序はＲ'以外の資源が仮定する処理順序との競合や、新たに投入されるジョブのためシステムが実際にプロセスを処理する順序とは一致しない。そして処理時刻の遅いプロセスほどこの可能性は高くなる。そこで［近似２］では、処理時刻の遅いプロセスの開始時刻をその最早開始時刻に仮定する。

［近似２］　資源Ｒの処理候補プロセスＰに対して、資源Ｒ'が算出する(4.2)式における$e_1(t,R',P)$の近似を考える。Ｒの処理候補プロセス集合$Ct(R)$が構成する連続処理区間に含まれるプロセスの集合を$\overline{C}t(R)$、$\overline{C}t(R)$とその後続プロセスから成る集合を$Vt(R)$とする。また、Ｒ上でのＰを先頭に固定したプロセス群$\overline{C}t(R)$の処理順序Gpを固定して、$Vt(R)$のプロセス最早開始時刻をGpに従って遅らせる時、Ｒ'に割当てられたプロセスでその最早開始時刻が$Vt(R)$中のプロセスを含む、最も遅い連続処理区間の終了時刻以前であるものの集合を$\overline{V}t(R,R',Gp)$とする。この時、$e_1(t,R',P)$の近似を$\overline{V}t(R,R',Gp)$の最終プロセス群を処理する資源の集合Ｆ($\overline{V}t(R,R',Gp)$)上でのＳ($\overline{V}t(R,R',Gp)$)中のプロセスを含む最も遅い連続処理区間終了時刻のGp、およびＲ'上でのプロセス群$\overline{V}t(R,R',Gp)$の処理順序に関する最小値$e_2(t,R',P)$とする。もちろんＲ，Ｒ'およびＦ($\overline{V}t(R,R',Gp)$)上では、$\overline{V}t(R,R',Gp)$とその後続プロセス群以外のプロセスはその最早開始時刻で処理可能になると仮定する。$TI(t,R,P)$の近似$T_2(t,R,P)$は次式によって求まる。

$$T_2(t,R,P) = \max_{R' \in S(\overline{C}t(R)) \cup R} e_2(t,R',P) \quad \cdots (4.5)$$

□

この近似によって個々の資源が処理順序を考慮すべきプロセスの範囲が縮小され、スケジューリングに必要な計算量をさらに削減できる。また、$e_2(t,R',P)$を算出すべき資源の範囲も［近似１］より狭くなる。さらに(4.6)式が成立するのは自明であり、T_2がT'より良い近似である事実は変わらない。

$$T'(t,R,P) \leqq T_2(t,R,P) \leqq T_1(t,R,P) \leqq TI(t,R,P) \quad \cdots\cdots \quad (4.6)$$

$e_2(t,R',P)$の計算には、Pを先頭に固定したプロセス群$\overline{C}t(R)$の処理順序Gpの全てに対して$\overline{V}t(R,R',Gp)$を求め、さらに$\overline{V}t(R,R',Gp)$中のプロセスの可能な全ての処理順序について、それらプロセスの最終プロセス処理資源上での処理終了時刻を計算しなければならない。そこで［近似３］では、資源R'が仮定するR上でのプロセス群Ct(R)の処理順序を、Rが$e_2(t,R,P)$を算出する時に仮定する順序に固定する。また$\overline{V}t(R,R',Gp)$内のプロセス数をnとすると、それらの可能な処理順序の数はn！であり、$e_2(t,R',P)$の算出にはn！に比例する計算量が必要であるが、これをn_2オーダの計算量で算出可能な値で近似する。

［近似３］　まず後述のようにして求まるPが先頭になる$\overline{C}t(R)$の処理順序$\overline{Gp}$を固定する。そのうえで資源R'は、プロセス群$\overline{V}t(R,R',Gp)$の処理順序を以下のようにして求める。

(1) $\overline{V}t(R,R',\overline{Gp})$中の処理順序の決まっていないプロセスの集合をX、処理順序の決まっているプロセスの集合をYとする。Xに属するプロセスQがXの中で最後に処理されると仮定して、Qの後続プロセス群の最早開始時刻をその分だけ遅らせる。さらにY、$\overline{C}t(R)$およびその後続プロセス群の最早開始時刻を、それまでに定めたY

の処理順序と$\overline{Gp}$に従って遅らせて、他のプロセス群はそれらの最早開始時刻で処理可能になるとする。これらの仮定の下でR'が、Xに属する各プロセスQに対して、$\overline{V}t(R,R',\overline{Gp})$の最終プロセス集合を処理する資源群F（$\overline{V}t(R,R',\overline{Gp})$）上での$\overline{V}t(R,R',\overline{Gp})$の後続プロセス集合S（$\overline{V}t(R,R',\overline{Gp})$）を含む最も遅い連続処理区間の終了時刻$\tau$（Q，X）を求める。

(2) R'がτ（Q，X）を最小にするプロセスQ*をX中の最後に処理するように順序を決める。

(3)Q*をXから除きYに追加する。さらにXが空でなければ(1)からの処理を繰返す。

(1)～(3)の手順で$\overline{V}t(R,R',\overline{Gp})$中の全てのプロセスの処理順序が決まった時、F（$\overline{V}t(R,R',\overline{Gp})$）上の$\overline{C}t(R)$および$\overline{V}t(R,R',\overline{Gp})$の後続プロセスを含む、最も遅い連続処理区間の終了時刻$e_3(t,R',P)$を、(4.5)式における$e_2(t,R',P)$の近似とする。TI(t,R,P)の近似値TA(t,R,P)は次のようにして求まる。

$$TA(t,R,P) = \max_{R' \in S(\overline{C}t(R)) \cup R} e_3(R,R',P) \quad \cdots (4.7)$$

また$\overline{C}t(R)$中のプロセス処理順序$\overline{Gp}$は、$\overline{V}t(R,R',\overline{Gp})$を$\overline{C}t(R)$で置換えた集合に対して(1)～(3)の手順を施すことによって得られる。

□

［近似３］ではプロセス集合Xの処理順序は、Q∈Xの処理順序を最後にしてX中の他のプロセスが、その最早開始時刻で処理可能になると仮定して求める。つまり、このような仮定の下でのジョブ終了時刻を最小化するプロセスQ*をXの最後に順序付けるという操作を、X中の全プロセスの順序付けが求まるまで繰返す。従ってX中のプロセスの数をnとすると、$n^2/2$に比例する計算量で処理順

序を求めることができる。しかし、この方法で求めたTA(t,R,P)はTI(t,R,P)よりも大きくなる可能性があり、もはや(4.3)や(4.6)式に対応するような関係が成立するとは限らない。

3.3 スケジューリング手順

次に［近似３］を用いたスケジューリングの実行手順を示す。本手順は、各資源Rが(4.7)式を最小化するプロセスを次処理プロセスとして選択することによって進行するが、(4.7)式の算出はRの問合わせに関連資源群が返答する形で行なわれる。従って、手順全体は4.2節で示したビディング法と同一の形態をしており、宣言形システムの条件を満足する。つまりシステム全体での処理の同期をとるような集中機構の存在を仮定する必要はなく、また各要素がシステムの全体構造を知る必要がないのでシステム運転中にジョブ投入計画やシステム構成の変更があっても、個々の要素は何の影響を受けることもなく処理を続行することができる。

手順実行に際して、各資源は必要な情報を以下に示すテーブルの形で記憶する。

［処理予定プロセス・テーブル］　各資源は、割当てられたプロセス群に関する情報を、プロセス毎に｛プロセス名，所属ジョブ名，最早処理区間｝から成るレコードとして記憶する。ここで所属ジョブとはそのプロセスを含むプロセス系列が構成するジョブのことである。従って、PとQが互いに先行あるいは後続プロセスの関係にある時、それらは同一の所属ジョブ名を持つ。

［最終プロセス処理資源テーブル］　各資源は、割当てられたプロセスの最終プロセスを処理する資源群の状態を、最終プロセスを処理する資源毎に｛資源名，後続処理区間，一般処理区間，変更フラグ｝から成るレコードとして記憶する。ここでテーブルを所有している資源をR、最終プロセス処理資源の１つをR'とすると、後続処理区間と一般処理区間は次のような意味を持つ。つまり、Rに割

当てられたプロセスとその先行プロセスを処理する資源群に割当てられたプロセスが所属するジョブの集合をＪとすると、Ｒ'に割当てられたプロセスＰ'の所属するジョブｊがＪの要素である時、後続処理区間はｊの名前とＰ'の最早処理区間の組（Ｒ'にｊに所属する複数のプロセスが割当てられている場合は、この組にはそれらのプロセスに対応して複数の最早処理区間が含まれる）の集合である。一般処理区間は、Ｒ'に割当てられたＪの要素でないジョブに所属するプロセス群が構成する連続処理区間の合併である。また変更フラグは、処理の進行や新たなジョブの投入により、後続あるいは一般処理区間の内容に変更のあることを示すフラグである。

スケジューリングは資源間で、これらの情報を交換することによって、次のように進む。

(1) 資源Ｒが、時刻ｔにプロセスの処理を終了すると、処理予定プロセス・テーブルより処理候補プロセス集合Ｃｔ(Ｒ)と、Ｃｔ(Ｒ)を含むＲ上の連続処理区間を構成するプロセスの集合$\overline{C}$ｔ(Ｒ)を求める。

(2)資源Ｒは次に、Ｃｔ(Ｒ)中の各プロセスＰに対して、Ｐを最初に処理する場合に資源群Ｆ（$\overline{C}$ｔ(Ｒ)）上の$\overline{C}$ｔ(Ｒ)の後続プロセスを含む、最も遅い連続処理区間の終了時刻を最小化する$\overline{C}$ｔ(Ｒ)の処理順序$\overline{G}$ｐと、その時の終了時刻ｅ^{3}(ｔ,Ｒ,Ｐ)を求める。これは、Ｒがその最終プロセス処理資源群テーブルの各レコードに対して、その一般処理区間にＲの想定する$\overline{C}$ｔ(Ｒ)の処理順序に応じてその開始時刻を遅らせた後続処理区間を接続することによって得られる。計算が終ると、Ｒは他の資源群に｛Ｐ, $\overline{G}$ｐ ｜ Ｐ∈Ｃｔ(Ｒ)｝と$\overline{C}$ｔ(Ｒ)の各プロセス処理時間を放送する。

(3)｛Ｐ, $\overline{G}$ｐ ｜ Ｐ∈Ｃｔ(Ｒ)｝と$\overline{C}$ｔ(Ｒ)の各プロセス処理時間を受取った資源Ｒ'は、自身が$\overline{C}$ｔ(Ｒ)の後続プロセスを処理する場合のみ、各Ｐに対して$\overline{V}$ｔ(Ｒ,Ｒ',$\overline{G}$ｐ)を求め、手順(2)と同様にして資源群Ｆ（$\overline{V}$ｔ(Ｒ,Ｒ',$\overline{G}$ｐ)）上での$\overline{V}$ｔ(Ｒ,Ｒ',$\overline{G}$ｐ)の後続プロセスを含む最も遅

い連続処理区間の終了時刻を最小化する$\overline{V}t(R,R',\overline{Gp})$の処理順序と最小値$e_3(t,R',P)$を計算する。計算結果は｛P，$e_3(t,R',P)$｜P∈$Ct(R)$｝としてRに返送する。

(4)RはR'から返答を受取ると、各P∈$Ct(R)$に対してR'（Rも含む）に関する$e_3(t,R',P)$の最大値を計算し、それを最小にするプロセスP*を次に処理する。さらにRは、$\overline{C}t(R)$の後続プロセスを処理する資源群にP*が選択されたことを放送する。

(5)$\overline{C}t(R)$の後続プロセスを処理する資源R'は、RからP*の選択通知を受取ると、それに応じて処理予定プロセス・テーブルの最早処理区間、さらには最終プロセス処理資源テーブルの後続処理区間を遅らせる。テーブル内容を変更した場合には、R'はさらに｛P*，R'｝を変更通知として他の資源群に放送する。

(6)｛P*，R'｝を受取った資源R''は、既に手順(5)でP*に関するテーブル修正を済ませている場合は何もしない。そうでない場合は、最終プロセス処理資源テーブルの資源名R'を持つレコードの変更フラグを立てる。

次に最終プロセス処理資源テーブルの内容変更手続きを示す。各資源Rは、手順(2)あるいは(3)で最終プロセス処理資源テーブルのR'に関するレコードを参照する場合には、必ずそのレコードの変更フラグを確認する。そして変更フラグが立っていない場合はレコード内容をそのまま利用するが、そうでない場合はR'から処理予定プロセス・テーブルの内容を受取って、以下のようにレコードを修正する。つまり、R'から受取った処理予定プロセス・テーブルから、修正対象レコードの後続処理区間に登録されたジョブに所属するプロセスの最早処理区間を除去する。この時残ったプロセス群の最早処理区間を接続したものが、修正後の一般処理区間であり、また除去したプロセス群の最早処理区間が、修正後の後続処理区間となる。但し、上述のテーブル内容変更手続きでは、R'は処理予定プロセス・テーブルの内容を放送通信によって転送する。このことによって

R'は1つの資源上で発生したスケジューリング要求に伴なう、複数資源からのテーブル内容転送要求に対して、放送通信を1回行なうだけで答えることができる。また、テーブル内容を受取った各資源R''は、自身に$e_3(t,R'',P)$を計算する必要がある時のみテーブルを修正することによって、関連資源の状態が変化するごとにテーブル内容を変更するむだを省略することができる。

3.4 スケジューリング法の評価

ここでは、本節で提案した多段スケジューリング法を、その実行に必要な通信量，記憶量，処理量，生成スケジュールの効率の面から評価する。以下ではシステム中の資源の総数をN、1つの資源に同時に割当てられるプロセスの最大数をMとする。

手順(2)～(4)では、資源Rが$\overline{C}t(R)$の後続プロセス処理資源群S($\overline{C}t(R)$)に$e_3(t,R',P)$の計算を依頼し、各資源から結果を受取る。この時、計算依頼には放送通信が1回、結果の受取りにはS($\overline{C}t(R)$)中の資源数だけの通信回数が必要になる。従って、各資源の単位時間内にスケジュールを実行する回数が資源への割当てプロセス数Mに比例し、システム中の全資源数がNであることを考えると、手順実行による単位時間当りの通信量は$N^2 \cdot M$のオーダを超えることはない。次に各資源上の最終プロセス処理資源テーブルの内容変更に必要な通信量を考える。各資源からの関連資源へのテーブル修正内容の要求は1回の放送通信で行なわれ、単位時間内に1つの資源がテーブル修正内容を要求する回数は、システム中の全資源がスケジューリングを実行する回数より少なく、$N \cdot M$オーダ以下であるから、システム全体でテーブル修正内容要求に必要な単位時間当り通信量は$N^2 \cdot M$オーダを超えない。テーブル修正内容の転送に関しては、1回の要求に対して最大N個の資源が情報転送を行なうが、ある資源上のスケジューリング操作に派生して発生する複数資源からのテーブル修正内容要求には、各資源は1回の放送通信で答える。従って、単位時間内に全ての資源がスケジューリング

を実行する回数がN・Mに比例することを考えると、テーブル修正内容の転送に必要な通信量もN^2・Mのオーダを超えない。結局、単位時間内にシステム全体でスケジューリング手順(1)～(6)を実行するのに必要になる全通信量は、N^2・Mオーダを超えないことがわかる。

システム中の全資源が必要とする処理予定プロセス・テーブルと最終プロセス処理資源テーブルの容量も、容易にわかるようにN^2・Mのオーダを超えることはない。

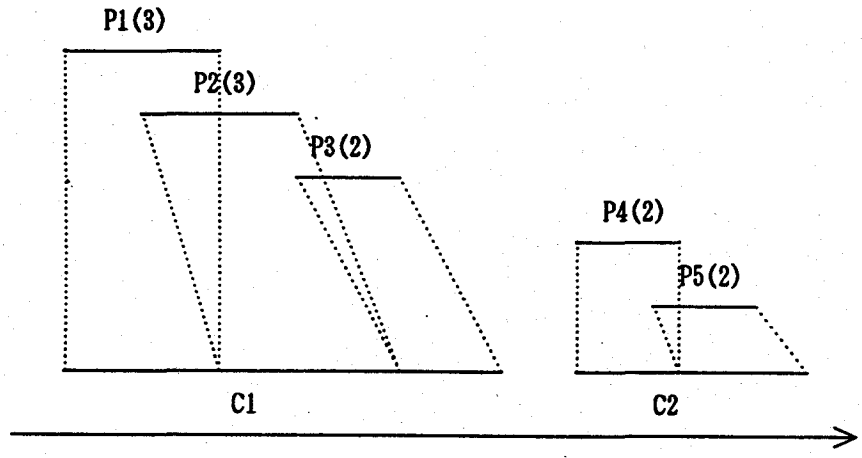
資源で発生する1回のスケジューリング要求に必要な処理量のほとんどは、手順(2)と(3)における$e_3(t,R,P)$の算出に要するものである。$e_3(t,R,P)$の算出には、各資源が最終プロセス処理資源上の連続処理区間を最大で$M^2/2$回求めなければならない。連続処理区間の計算自体にはMオーダの計算量が必要であるので、結局、各資源は$e_3(t,R,P)$の算出に当ってM^3オーダの演算をすることになる。(4.7)式のTA(t,R,P)を求めるには、さらに$e_3(t,R,P)$を処理候補プロセス集合中の各プロセスに対して算出する必要があるので、qを処理候補プロセスの数とすると、全体ではq・M^3オーダの演算が必要である。但し、q，Mは共にNに比べると小さい。

次に提案方法によるスケジューリングの性能をシミュレーションによって評価する。表4.1にシミュレーション条件を示す。システムは6つの資源R_1～R_6から成り、そこへ時刻0にジョブ群J_1～J_6が、また時刻5にJ_7～J_{10}が投入される。表4.1は、これらのジョブを構成するプロセスと、プロセスを処理する資源および処理時間との関係を示している。例えば、時刻5に投入されるJ_7は3つのプロセスから構成され、最初のプロセスはR_1が処理し、処理には3単位の時間が必要である。第2、第3のプロセスはそれぞれR_3、R_4が処理し、2単位，9単位の時間が必要である。

図4.3、4.4にスケジューリング結果を示す。図4.3は時刻0にJ_1～J_6だけが投入された場合に、また、図4.4は時刻5にさらにJ_7～J_{10}が投入された場合に対応する。図はR_1～R_6が処理するプロセ

スの時間変化を示しており、図中のＰｉ（ｊ）はジョブＪｉのｊ番目のプロセスを処理していることを表わす。また白抜き、ハッチング、塗りつぶしの箱は、各々ＳＰＴ、ＭＷＫＲ、提案手法を用いたスケジューリング結果に対応する。図4.3では、提案手法で全ての処理の終了する時刻が２１であるのに対し、ＳＰＴ，ＭＷＫＲルールでは終了時刻は２４になる。また、図4.4では、提案手法では時刻２５に全ての処理が終了しているのに対して、ＳＰＴ，ＭＷＫＲルールではそれぞれ３０，３１に処理が終了する。いずれの場合も提案手法の方が良い成績を上げていることがわかる。図4.4の方が成績の差は、絶対値としても相対値としても大きいが、これはプロセス数の増加とともにスケジューリング性能の差が加算されるためと考えられる。

同図からもわかるように、提案手法は全ての資源上での処理終了時刻がなるべく等しくなるように働く。実際、図4.4ではR_3上での処理はＳＰＴルールによる方がむしろ早く終了している。従って、システム中の全資源の処理量が等しい時にジョブの終了時刻が最小になることを考えると、多くの場合に、提案手法がＳＰＴやＭＷＫＲによる方法よりも良いスケジュールを生成することが期待できる。



(C1,C2 がプロセスP1～P5の連続処理区間となる)　　時刻

図4.2 資源上の連続処理区間

表4.1 システムへの投入ジョブ

(a) : 時刻0で投入されるジョブ

	1stプロセス	2ndプロセス	3rdプロセス	4thプロセス
J1	R1 (7)	R3 (5)	R5 (3)	
J2	R1 (2)	R4 (1)	R3 (2)	R6 (2)
J3	R1 (2)	R2 (2)	R3 (2)	R4 (1)
J4	R2 (7)	R6 (5)		
J5	R2 (4)	R4 (2)	R5 (4)	
J6	R2 (3)	R3 (2)	R4 (2)	R6 (4)

(b) : 時刻5で投入されるジョブ

	1stプロセス	2ndプロセス	3rdプロセス	4thプロセス
J7	R1 (3)	R3 (2)	R4 (9)	
J8	R1 (1)	R4 (2)	R5 (2)	
J9	R2 (2)	R5 (2)	R6 (2)	
J10	R2 (1)	R3 (4)	R6 (2)	

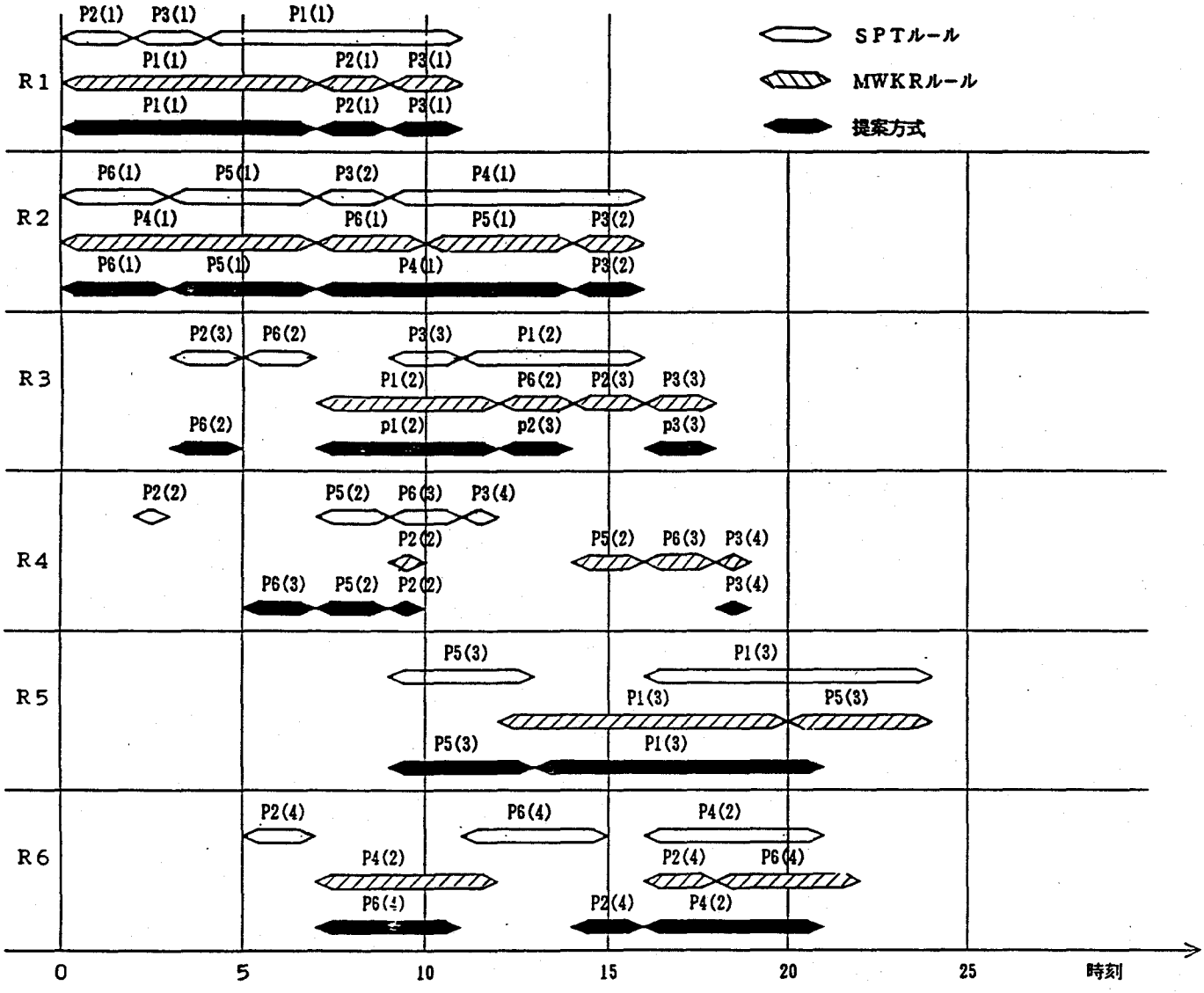


図4.3 スケジューリング結果　（J1 ～J6 を投入）

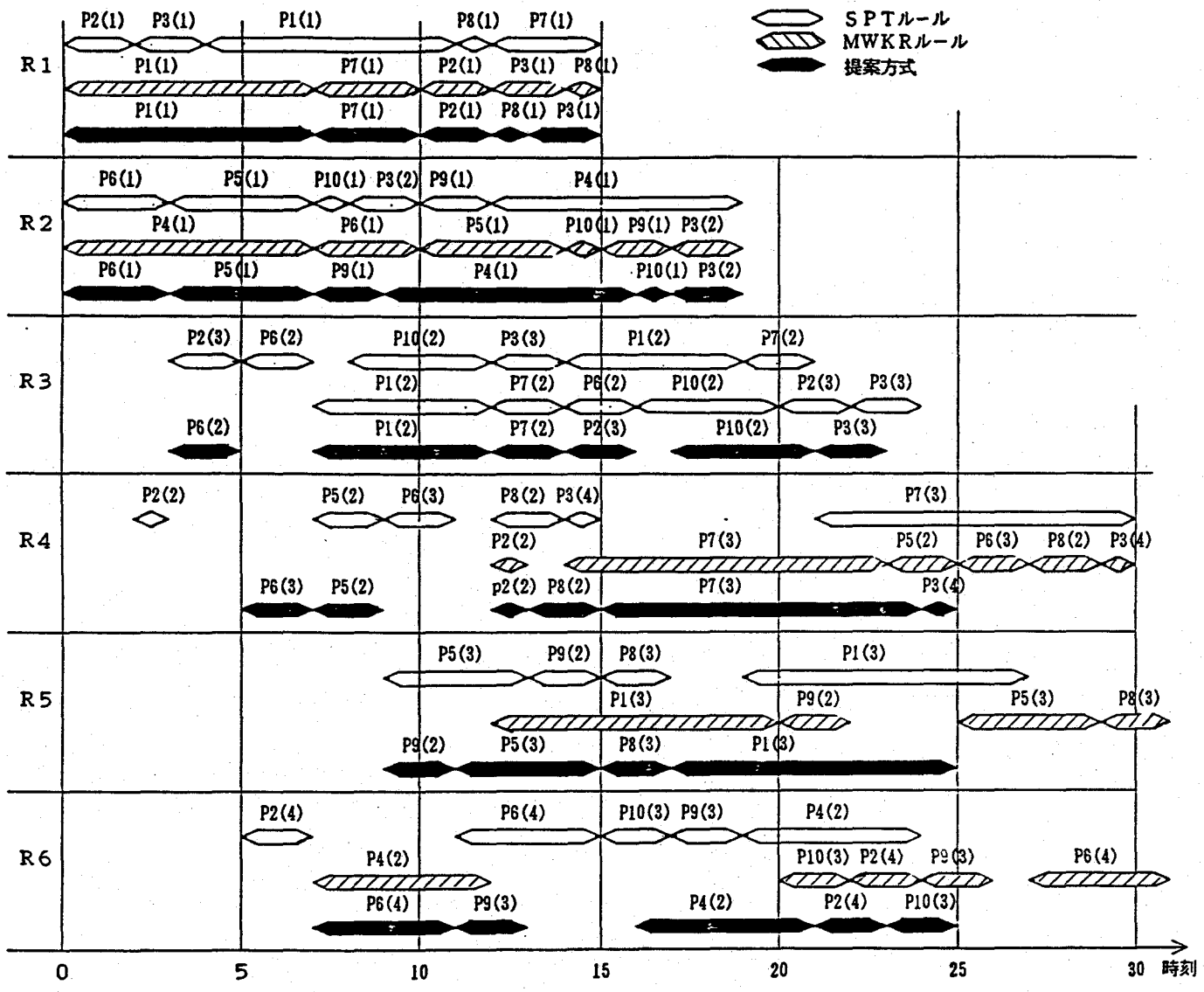


図4.4 スケジューリング結果　（J1 ～J10を投入）

第４節　考　察

第２節で述べた単段スケジューリングでは、プロセスがある程度の間隔をおいて１つづつ投入される場合には、通常のビディング法でも良好な資源割当てを実現する。しかし、ほぼ同時に複数のプロセスが投入される場合、例えば生産システム等でよく見られるようにいくつかの入力装置が各々ある期間中にたまった仕事をまとめて投入するような場合には特定資源への負荷集中を招く。両方向ビディング法はこのような場合にも良好な資源割当てを実現するが、これは通常のビディング法が個々のプロセス要素と資源要素間の互いに独立した情報の交換しか行ってないのに対して、両方向ビディング法がプロセスの資源に対する要求度の算出に際して等価的に資源要素間での情報交換を行うからである。つまり、各資源要素はプロセス要素から要求度を受取ることによって間接的に他の資源群の負荷状態を知ることができる。そこで両方向ビディング法を、資源要素群が直接互いに情報を交換する形態に変形することが考えられる。この方法は第６章３節に示す電力系統における電圧の一定化制御で適用するが、各資源が互いに他の資源の状態を直接知ることができるので、より良好な資源割当てを実現する。しかし、他の資源の状態を完全に把握してしまうとディマンド・マネージメントやマネージメント・バイ・コンセンサスと同様に集中管理的な性質が強くなり［２２］、宣言形システムの実現には適さなくなる。従って資源群が情報を直接交換する場合も、交換すべき情報やその結果個々の資源が獲得する他の資源群の状態を集約、簡略化する方式が重要になる。

第３節では多段スケジューリング法を、代替資源の存在しない場合、つまり個々のプロセスを実行可能な資源がシステム中に１つしか無い場合について考えたが、提案方法は次のようにして代替資源の存在する場合の方法ににも拡張できる。即ちスケジューリングを資源へのプロセス割当てを行う過程（割当て過程）と、資源が割当

てられたプロセス群から次に処理すべきプロセスを選択する過程（選択過程）に分離し、選択過程では提案方法をそのまま使う。新たにジョブが投入されると、それを形成するプロセス系列の各プロセスが最早開始時刻で処理可能になるとして、先頭プロセスから順にその最早開始時刻でのシステム中の資源の負荷が平準化されるように資源を割当てる。この時割当て済みのプロセスの処理順序は第3．3節の手順(3)で$e_3(t,R',P)$を算出した時の順序に仮定する。負荷を平準化する資源割当てには第2節で述べた両方向ビディング法を用いることができる。

尚第3節では、個々の資源上で並列に進行するスケジューリング処理間での整合性を保証する方法については述べなかった。各資源はその上でのスケジューリング処理中に別の資源の状態を参照するが、参照中の資源の状態がその資源上でのスケジューリング処理の開始によって変化して、処理の間に矛盾の生じることがある。しかしこれらの矛盾は、時刻印法［30］を使うことによって、集中機構を設けることなく簡単に防止することができる。

第５節　結　言

本章では知的分散システムの要素間協調機構として、単段スケジューリングと多段スケジューリングの分散方式を提案した。提案した方法はプロセス要素群と資源要素群間での情報交換によって実現され、各要素が互いに他の要素の存在や位置を前もって知る必要は無い。また他の要素群の状態を常時監視する必要も無く、スケジューリング処理のためにシステム全体が同期して動作することもないので、個々のシステム要素の独立性が保証され、宣言形システムの実現が容易になる。

多段スケジューリングにおいては、ジョブを構成するプロセスの系列に分岐や合流の無いことを仮定し、さらにジョブの納期制約や各資源の持つバッファの容量制約を考慮しなかったが、今後はプロセス系列に分岐・合流の存在する場合を扱ったり、各種の制約を考慮する方法への拡張を考えたい。また単段スケジューリングにおける両方向ビディング法では、資源間での直接の情報交換の機構を許していないが、第６章３節に示すような資源間での情報交換をも取り入れた形の改良も検討したい。

第５章　知的分散ＯＳのアルゴリズム

第１節　序言

知的分散システムは、第２章から４章で示したように、各々が独立に宣言された要素群の協力と協調によって集中機構を介することなく動作するが、これら要素が互いに矛盾なく動作するには協力と協調の機構だけでは不十分である。即ち、複数の目的がこれらの要素を共有する時、ある目的のために設定した要素群の状態が別の目的によって破壊されたり、あるいは誤って利用されるのを防ぐには同時実行制御の機能が必要である。また要素間で交換されるメッセージの、全ての関連要素群への正しくかつ矛盾のない順序での伝送を保証する機能が必要である。さらに第３章、４章では全ての要素は正しく動作すると仮定したが、高い信頼性が要求されるシステムでは、部分的な故障発生時でも正常時と全く同じ動作が可能になるような機能が重要である。

システムから集中機構を排するには上述のような機能も分散機構で実現しなければならない。本章ではこれら機能の分散化アルゴリズムを提案する。知的分散システムでは、これらアルゴリズムは第２章４節に示したように、個々の要素（オブジェクト）に分散した共通知識上に実装される。従って上述の機能も、関連要素群の結合によって実現され、集中機構が介入する必要は全く無い。第２節では同時実行制御の分散化方式を、デッドロックの検出と防止を中心に述べる。既にデッドロックを検出あるいは防止する多くの分散アルゴリズムが提案されているが［３２］［３３］［３４］［３５］、それらのアルゴリズムではシステム中の要素の総数をｎとすると、n^2～n^3オーダの計算量とn^2オーダ以上の通信回数が必要であった。ここではn^3オーダの計算量とｎオーダの通信回数のアルゴリズムを提案する。提案アルゴリズムの全計算量は従来アルゴリズムと同程

度であり、１つのメッセージの含む情報が多くなるので通信の総量も変化しないが、要素間の通信回数をｎオーダにまで下げたことにより、異なる要素上での並列処理の効果が高まるとともに、通信オーバヘッドの大幅な削減が可能になり、実質的には従来よりも優れた分散アルゴリズムになる。さらにここでは、デッドロックの検出と防止の併用が可能なロック規約を提案する。デッドロックの検出と防止には各々長所と短所があり、目的によって使い分けのできることが望ましいが、従来は非常に強い制約の下でしか２つの方式を併用することができなかった［３６］。提案するロック規約はこの制約を大幅に緩和する。

第３節では要素間での正しいメッセージ交換を保証する機構を考える。従来の計算機システムでは主に１対１の通信が用いられ、正しいメッセージ交換の保証も、単にメッセージを受信した要素が送信要素に確認信号を返送するだけで十分であった。つまり送信側要素が確認信号を受取ることによって、受信要素へのメッセージ到着が確認できた。しかし宣言形のシステムでは放送通信が多用されるので、確認信号の交換は現実的ではない。１回の放送通信に対して多数の確認信号の返答が必要になるので通信オーバヘッドが高くなるし、また放送は不特定多数の要素が対象であるのでどの要素から確認信号を受取ればよいのかも一般にはわからない。さらに確認信号の交換では、確認信号が受信できない場合に送受どちらの要素に問題があるのかが判定できない。受信要素に問題があれば送信要素はもちろん確認信号を受取ることはできないが、送信要素に問題があっても確認信号は受信できない。このような問題を解決するため既に放送通信の到着の確認を行ないいくつかのアルゴリズムが提案されているが、それらは集中機構を仮定していた［３７］。そこでここでは集中機構を設けない方式として、確認信号を交換する方式に代って、各要素が通信路上のメッセージを計数するフェイル・ストップ放送通信方式を提案する。提案方式によって、システム中の各要素は確認信号を交換することなく他の要素が正しくメッセージ

を受信したかどうかを確認することができ、さらにメッセージの送受信に異常のあった要素の悪影響が他に波及するのを防ぐことができるようになる。

一部に異常が発生しても、システムが全体として異常の無い場合と同様に振舞うためには、要素の多重化が必要である。しかし宣言形のシステムを実現するには、個々の要素は多重化された要素の位置や多重度から独立でなければならない。例えば要素の動作が他の要素の多重度に依存すると、各要素を他から独立に定義したり追加することはできない。そこで第４節では、各要素が放送通信によって多重化された要素群を同時に呼出す放送並列多重化方式を提案する。提案方式が正しく動作するためには、多重化要素群の全てが同一のメッセージを同一の順序で、また正しいメッセージだけを受取る性質の実現が必要であるが、既に提案されている類似の方式ではこれらの性質が仮定されていたり、多重化要素間での同期と情報交換が必要でありオーバヘッドが高かった［４１］［４２］［４３］。提案方式は第３節のフェイル・ストップ放送通信等の機能を使って、これらの性質を低オーバヘッドでかつ分散的に実現する。

第２節　同時実行制御

同時並行に進行するジョブ群が資源に無秩序にアクセスすると、あるジョブが使用中の資源の状態を他のジョブが変更したりして処理結果に矛盾が発生する。複数のジョブを矛盾なく進行させるには、ジョブの資源へのアクセス順序が資源間で逆転しないようにしなければならない。例えばジョブＡとＢが共に資源１と２をアクセスする場合、資源１がジョブＡ，Ｂの順でアクセスされるなら、資源２上でもジョブＡのアクセスはＢのアクセスに先行しなければならない。このような目的でのジョブの資源アクセス動作の制御は同時実行制御と呼ばれ、二相ロック法や時刻印法が広く使われている［３０］。時刻印法は個々の資源の単独の判断でデッドロックの防止が可能であり、形式的には宣言形のシステムに最適の方法であるが、資源アクセスが正常であるのに処理の中止、再実行を行なうことがあり処理の並列性が落ちる。そのため知的分散システムでは二相ロック法を採用する。本節では、二相ロック法を用いる際に発生するデッドロックを検出あるいは防止する分散機構を提案する。

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無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処理を中止することによってデッドロックから脱出する検出法とがある。これらの方法は従来各々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提案する。防止法を用いるには、ジョブ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なく、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ような場合には検出法が有効であり、制御システムのように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。提

案するロック規約により、１つのジョブ内で目的に応じてこれらの方法が自由に使い分けられるようになる。

デッドロックの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［３１］。即ち、デッドロック状態である必要十分条件は、図５．１に示すジョブによる資源ロックとロック許可待ちの関係を表わすHold-Wait グラフ上に有向枝の閉路が存在することであり、またジョブ群が将来どのようなロック要求を出しても、デッドロックを発生せずに全てのジョブを終了させる方法が存在するための必要十分条件は、図５．２に示すHold-Claimグラフ上に有向枝の閉路が存在しないことである。ここで図５．１上では、ジョブＡが資源１ をロックしている時に資源１からジョブＡに向う有向枝が、逆にジョブＢが資源１のアンロックされるのを待っている時、ジョブＢから資源１に向う有向枝が定義される。また図５．２上では、ジョブＡが資源１をロックしている時に資源１からジョブＡに向う有向枝が定義され、ジョブＢが資源１のロック予約をしている時、ジョブＢから資源１に向う有向枝が定義される。

従って、有向グラフ中の閉路を検出することによりデッドロックの検出と防止ができることになるが、既に閉路を検出する分散アルゴリズムとしてプローブ法［３２］や可到達集合法［３３］、その他が提案されている［３４］［３５］。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中の有向枝に沿ってノードからノードへ順次転送し、それがプローブ送出済みのノードに戻った時に閉路を検出する。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、さらに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。そこで知的分散システムでは、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時、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ないので並列処理の効果を引出せるが、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、有向枝の削除に伴う可到達集合の変更は図５．３に示すように複雑であり、従来は有向枝削除時の有効な可到達集合更新方法がなかったので、可到達集合法は二相ロック規約に従う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、デッドロックを検出した時以外に有向枝が削除されるのはジョブが資源をアンロックする場合だけであるが、二相ロック規約に従えばアンロック操作を行なうジョブは必要な全ての資源をロックし終っているので、図５．３のように削除対象有向枝の行先きノードから出る有向枝が存在することはなく、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提案する。必ずしも二相ロック規約に従わなくても処理の整合性を保証できる場合も多いが、本アルゴリズムによって可到達集合法がこのような場合にも、またデッドロックの検出だけではなく防止にも利用できるようになる。有向枝が追加された場合の可到達集合の更新手順は自明であるので、以下では有向枝削除に伴う可到達集合の更新アルゴリズムを考える。本アルゴリズムでは、図５．１，５．２に示したジョブと資源のノード間の関係を、図５．３のようにジョブのノード間だけの関係に変換した有向グラフを扱う。図５．３は、ジョブＡのロックしている資源の少なくとも１つをジョブＣがロック要求あるいはロック予約している時に、ジョブＣからＡに向う枝を定義する規則によって、資源のノードを含むグラフから変換される。次にノードＡからＢに向う有向枝を除く際の可到達集合更新手順を示す。手順では各ノードＸはその可到達集合とともに、逆向可到達、隣接、逆向隣接集合を記憶しており、それらを各々ＦＲ（Ｘ），ＢＲ（Ｘ）ＦＤ（Ｘ），ＢＤ（Ｘ）と表わす。ここでＦＤ（Ｘ）とは、

ノードXから出る有向枝によって直接到達可能なノードの集合であり、ＢＤ（Ｘ）はその隣接集合にＸを含むようなノードの集合である。またＢＲ（Ｘ）は、その可到達集合にＸを含むようなノードの集合である。

［有向枝削除に伴う可到達集合更新手順］

(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）中のノードＣは｛Ｂ，ＦＲ（Ｂ），Ａ，ＢＲ（Ａ）｝を受取ると、ＦＲ（Ｃ）からＢおよび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、Ｓ（Ａ）＝$\cup_i$Ｔ（ｉ）∪ＦＤ（Ａ）のノード群に｛ｉ，Ｔ（ｉ）｝（ｉ∈ＢＲ（Ａ））の列を送る。この時Ａは、ｉの中にＡ自身も含める。ただし、Ｔ（Ａ）＝ＦＤ（Ａ）－Ｂである。

(5) ｛ｉ，Ｔ（ｉ）｝の列を受信したノードＤが、Ｄ自身が列中のＴ（ｊ）に含まれる時、Ａに｛Ｄ，ＦＲ（Ｄ），ｊ｝を返送する。

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

(7) ｛ｋ，ＦＲ（ｋ），ＢＲ（ｋｊ）∪ｋｊ｝の列を受信したノー

ドＥが、ＥがＢＲ（ｋｊ）∪ｋｊに含まれる時、

ＦＲ（Ｅ）＝ＦＲ（Ｅ）∪ＦＲ（ｋ）∪ｋ

とする。

ＢＲ，ＢＤの更新も、(1)～(7)の手順において、ＦＲ，ＦＤとＢＲ，ＢＤの役割を入換えることによって同様に行なうことができる。また複数の更新要求が同時に発生した場合の各ノード上での本手続き実行も、時刻印法を用いることにより簡単に制御できる。

［定理５．１］

手順(1)～(7)は、隣接、可到達集合の更新を正しく行なう。

（証　明）

ノードＡからＢに向う有向枝ａ（Ａ，Ｂ）が削除された場合を考える。ａ（Ａ，Ｂ）につながるノードはＡとＢだけであるので、その削除によって隣接集合が変化するノードはＡだけであり、またＡの隣接集合ＦＤ（Ａ）からはＢだけが減りＢ以外のものが減らないことは明らかである。上述の手順では、(2)においてＦＤ（Ａ）からＢを削除するので、正しい隣接集合の更新が行なわれる。

次にａ（Ａ，Ｂ）の削除によって可到達集合が変化するノードは、ＡおよびＡを可到達集合に持つノード、つまりＢＲ（Ａ）に属するノードＥだけであり、Ｅ∈ＢＲ（Ａ）∪Ａの可到達集合ＦＲ（Ｅ）から削除されるべきノードの集合は、ＦＲ（Ｂ）∪Ｂのノードでかつ Ｅからａ（Ａ，Ｂ）を経由してしか到達できないノード群である。ここでＦＲ（Ｂ）∪Ｂに属し、Ｅからａ（Ａ，Ｂ）を経由せずに到達可能なノードＹは次の関係を満たす。即ちＭ∈ＢＲ（Ａ）∩｛ＦＲ（Ｅ）∪Ｅ｝となるＭが存在して、Ｙ∈ＦＲ（ＦＤ（Ｍ）－ＢＲ（Ａ））∪（ＦＤ（Ｍ）－ＢＲ（Ａ））となるか、またはＹ∈ＦＲ（ＦＤ（Ａ）－Ｂ）∪｛ＦＤ（Ａ）－Ｂ｝でなければならない。上述の手順では、ＢＲ（Ａ）に属する各ノードＥは(3)において

ＦＲ（Ｅ）からＦＲ（Ｂ）ＵＢを削除し、(7)において上の関係を満足するノードＹの集合を加えている。従って(1)～(7)は、隣接、可到達集合の更新を正しく行う。

□

［定理５．２］

ＡからＢに向う有向枝が削除されても、ＢＲ（Ａ）に属する要素の逆向隣接、逆向可到達集合が変化することはない。またＦＲ（Ａ）に属する要素の隣接および可到達集合も変化することはない。

（証　明）

ＢＲ（Ａ）に属する要素をＸとする時、ＢＲ（Ｘ）の要素ＹとＸを結ぶ要素の列にＡがあると、Ａ∈ＢＲ（Ｘ）となるがこれはＸ∈ＢＲ（Ａ）の仮定に矛盾する。デッドロック検出、防止法を用いればグラフ中に確定した閉路が存在することはない（一時的に仮の閉路が存在することはある）。従って、ＡからＢに向う有向枝が削除されてもＹ∈ＢＲ（Ｘ）の関係は変化しない。ＦＲ（Ａ）に属する要素の隣接および可到達集合についても同様に証明できる。

□

［定理５．２］によれば、手順(1)～(7)を実行中にＦＲ（Ａ）に属する要素ＸのＦＲ（Ｘ）やＦＤ（Ｘ）が変化することがなく、また(1)～(7)に対応する逆向隣接、逆向可到達集合の更新手順実行中にＢＲ（Ａ）に属する要素ＹのＢＲ（Ｙ）やＢＤ（Ｙ）が変化することもない。従って各要素は手順(1)～(7)とそれに対応する逆向隣接、逆向可到達集合の更新手順を、それぞれ独立、並列に実行することができる。

知的分散ＯＳでは、個々のジョブが発生する毎に対応するジョブ・オブジェクトが生成され、デッドロックの検出、防止、および可到達集合等の更新アルゴリズムはこれらオブジェクト群上に分散し

て実現される。ここでデッドロックの検出あるいは防止に必要な演算は、追加された有向枝の根元ノードが行先ノードの可到達ノードに含まれるか否かを調べるだけである。また可到達集合の更新についても、ノードの総数をｎとすると、個々のノードに必要な計算量はｎ^{2}のオーダで済む（システム全体ではｎ^{3}のオーダ）。この計算量は既に提案されているアルゴリズムと変わることはないが［３５］、１つの計算機が同時に処理するジョブの最大数を設定すると、手順(1)～(7)の演算はビット・パターンの論理和、論理積、排他的論理和に帰着され、個々のノードの演算が簡単になる。さらに手順(1)～(7)は最大でも（５ｎ＋１）回の通信回数で実行できる。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、これはさらに（２ｎ＋４）回に減る。従来報告されているアルゴリズムで必要な通信回数がｎ^{2}であるので、本アルゴリズムによって大幅な通信回数削減が可能になる。但し１つのメッセージの含む情報が多くなるので、通信の総量は従来と変わらない。しかし通信回数の削減は分散要素によるアルゴリズムの並列実行度の増加を意味し、他のアルゴリズムにおける通信オーバヘッド等も考慮すると、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。

デッドロックの検出と防止は前述したように使い分けの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、そうでない資源をタイプ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。このような条件を満たすロック規約として既に、資源タイプに優先度を付け、ジョブの資源へのロック要求は必ずそれより優先度の高い資源を全てアンロックした後でしか許さない方式が提案されている［３６］。しかし、そこでは１つのジョブが異なるタイプの資源に同時にアクセスすることはできなかった。これはタイプの優先度設定に規準がないためであり、ここではタイプ１の優先度をタイプ２より高くすることによって、１つの

ジョブが同時に異なるタイプの資源にもアクセス可能なロック規約を提案する。次に提案するロック規約を示す。

［規約１］タイプ２資源をロック中にタイプ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］ジョブは自身がタイプ２資源をロックしていない時にのみ、タイプ２資源のロック予約ができる。

［規約２，３］はロック予約に関する通常の規則であり、本規約に特有の規則は［規約１］だけである。また［規約１］は、タイプ２資源をロック中のジョブのタイプ１資源への新たなロック要求は禁止するが、タイプ１資源をロック中でもタイプ２資源へのロック要求は受付ける。従来の規約ではこの両方が禁止されたので、１つのジョブが異なるタイプの資源に同時にアクセスすることはできなかったが、本規約によってそれが可能になる。本規約下でのデッドロックの検出と防止は次のように行なわれる。

つまり、あるジョブのタイプ１資源へのロック要求が他のジョブがロック中のために待たされた時、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ジョブを中止するが、そうでない場合は資源がアンロックされるのを待つ。ジョブがタイプ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資源だけをロック中のジョブを中止してデッ

ドロック状態から脱出する。本ロック規約が正しく動作することは次のように示すことができる。

［定理５．３］

［規約１］～［規約３］に従う時、デッドロック状態からは、タイプ１資源のみをロック中のジョブを中止することによって必ず脱出できる。

（証　明）

まずタイプ１資源へのロック要求によってHold-Wait グラフ中に閉路が生成される場合は、［規約１］によりロック要求を出したジョブはタイプ１資源しかロックしていないので、そのジョブを中止することによってデッドロック状態から脱出できる。またタイプ２資源のロックによってHold-Wait グラフ中に閉路の生成される場合も、閉路中には必ずタイプ２資源をロックしていないジョブが含まれる。なぜなら、タイプ２資源をロックしたことによりHold-Wait グラフ中のノードＡとＢの間に有向枝が追加される場合は、Hold-Claimグラフ中でも必ずノードＡとＢの間に有向枝が存在する。しかしHold-Claimグラフ中に閉路の存在しないことは確認されているので、Hold-Wait グラフ中の閉路には必ずHold-Claimグラフ中には存在しない有向枝、つまりタイプ１資源を要求しているジョブから出る有向枝が存在する。しかし［規約１］によって、このようなジョブはタイプ２資源をロックしていることはない。従って、タイプ２資源のロックによってHold-Wait グラフに閉路が生成される場合も、タイプ１資源しかロックしていないジョブを中止することによってデッドロック状態から脱出することができる。

□

［規約１］～［規約３］では資源のタイプ設定がジョブ間で一致することは必ずしも要求しない。つまりあるジョブがタイプ１に設

定した資源を他のジョブがタイプ２に設定しても構わないが、ジョブ間でタイプ設定が一致する時は次のような性質が成立し、各ジョブがHold-Wait グラフとHold-Claimグラフを別々のグラフとしてではなく、図５．４に示すような１つの混合グラフとして扱うことができる。従ってデッドロックの検出と防止を併用することによるオーバヘッドの増加が無くなる。

［定理５．４］

資源のタイプ設定がジョブ間で一致する場合は、Hold-Wait 関係とHold-Claim関係の混在した閉路が形成されることはない。従ってデッドロック検出あるいは防止のための閉路検出操作は、Hold-Wait とHold-Claimの関係を区別することなく実行できる。

（証　明）

ノードＡからＢに向うHold-Claim関係の存在は、Ａがロック予約中の資源ｉをＢがロックしていることを示すが、資源ｉはＡがロック予約しているのでＡから見ればタイプ２である。従って、資源のタイプ設定がジョブ間で一致する場合は資源ｉはＢから見てもタイプ２であり、Ｂはタイプ２資源をロック中となる。［規約１］によればＢがタイプ２資源をロックするのはタイプ１資源のロックを終えてからであるから、Ｂから出る有向枝は全てHold-Claim関係を表わす。同様にして、Ｂから到達可能な全てのノードに対しても、そのノードから出る有向枝がHold-Claim関係を表わすものであることがわかるが、このことは閉路がHold-Claim関係を含む場合は閉路全体がHold-Claim関係で構成されることを示している。

□

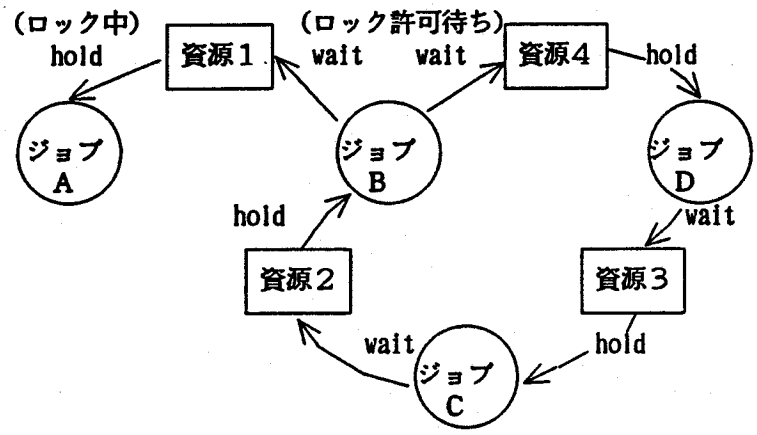


図 5.1 Ｈｏｌｄ－Ｗａｉｔグラフ

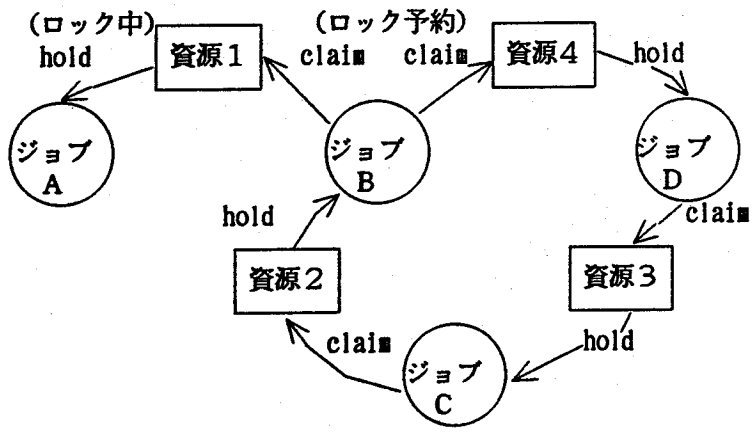


図 5.2 Ｈｏｌｄ－Ｃｌａｉｍグラフ

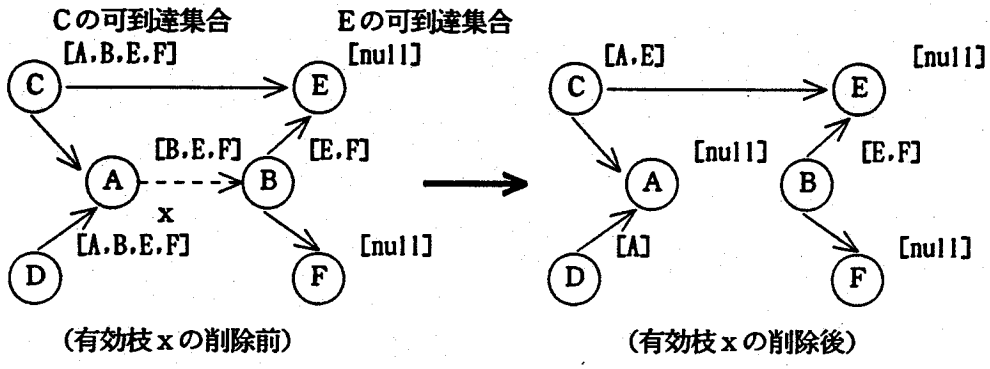


図 5.3　有効枝削除時の可到達集合更新

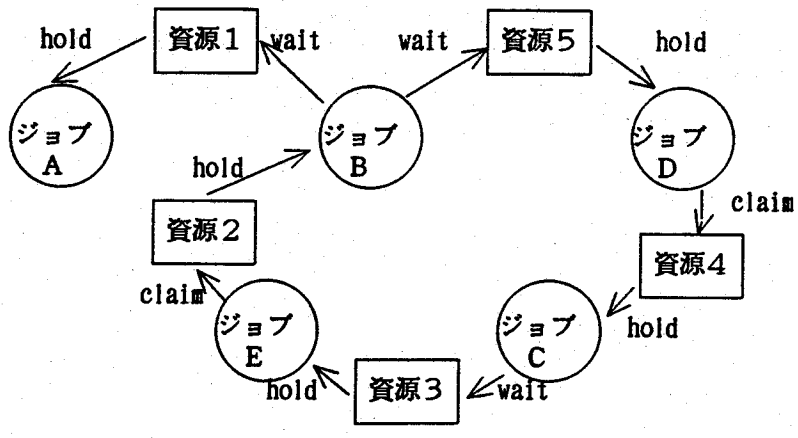


図 5.4 Hold-WaitとHold-Claimの混合グラフ

第３節　フェイル・ストップ放送通信

これまでに述べた知的分散システムの構造やアルゴリズムの正しい動作の前提には、要素間での誤りのないメッセージ交換がある。また故障等で正しいメッセージ交換のできない場合には、それをただちに発見し誤りが他の要素へ波及するのを防ぐ機能が必要である。メッセージ交換を確実にするために、従来のシステムではメッセージを受信した要素が送信側に確認信号を返答する方式が採られて来た。つまりメッセージを送出した要素は、確認信号を受取るとメッセージが正しく届いたと認識し、そうでない場合はメッセージを再送するなどの処理をする。しかし知的分散システムでは放送通信を用いたメッセージ交換が多用され、確認信号を交換する方式の採用は次のような意味で現実的ではない。即ち、放送通信は不特定多数の要素にメッセージを送出するものであるから、メッセージ送信要素がどこから確認信号を受取るべきかを判断できない場合が多いし、例え判断が可能な場合であっても、メッセージを送出する毎に多数の要素から確認信号を受取るオーバヘッドは非常に高い。また確認信号を交換する方式では、確認信号を受取れない時、送信側と受信側のどちらの要素に異常があるのかを特定するのは難しい。

本節ではこれらの問題を解決するために、低いオーバヘッドでしかも集中機構を介さずに、放送メッセージ到着を確認するフェイル・ストップ放送通信プロトコルを提案する。提案プロトコルはさらに、メッセージが到着しない場合の異常要素特定を容易にするとともに、通信異常による処理の誤りが他の要素に波及するのを防止し、システム中の関連する全ての要素の同一順序でのメッセージ受信も保証する。

放送通信の高信頼化を達成する方式として、既にトークンを用いるものや［３７］、メッセージ受信後に要素が送信するメッセージを確認信号とみなし二相コミットメントの手順を適用する方法［３８］、さらには各要素が通信路上のメッセージのログを記憶する方

法［３９］［４０］などが提案されているが、集中機構の存在を仮定したり、メッセージ到着確認のために大量の記憶域や処理時間を使うなど、オーバヘッドの低い分散システムの構築には十分ではなかった。また、これらの方法では異常のある要素がメッセージを送出するまで受信エラー等が検出できないので、異常時のメッセージ再送機構の実現なども難しかった。

フェイル・ストップ放送プロトコルは、ＬＡＮまたは内部バスで結合された計算機システムにおける通信を制御するもので、第２章４節で示した各計算機上の知的分散ＯＳ核の中に実現される。要素間でのメッセージ交換異常の原因にはソフトウェアとハードウェアの２つの側面があるが、ここではハードウェアの異常だけを考えソフトウェアは正しいものと仮定する。従って異常発生の単位は計算機やそれらを結合する通信路であって、計算機に搭載された個々のプログラム（オブジェクト）ではない。そこで本節では、以後システム要素を、オブジェクトではなく計算機およびＬＡＮ（通信路）と考える。また通信異常にはメッセージの紛失や重複等の他に、メッセージ内容の破壊があるが、メッセージ内容の異常に関してはパリティの付加等によって検出、回復できるので、以後では考えない。

次にフェイル・ストップ放送プロトコルを示す。本プロトコルでは、ＬＡＮあるいはバス上を同時に流れるメッセージが１つしか無いことを利用して、送受計算機間で確認信号を交換する代りに、システム中の全計算機が通信路中のメッセージを計数し、送受計算機間でこの計数値を比較する。即ち、各計算機上の知的分散ＯＳ核はＬＡＮを通過するメッセージを常に監視しており、受信したメッセージを特徴づける量の累積量を記憶する。そして送信すべきメッセージには自身の記憶している累積量を付加し、メッセージを受信すると受信メッセージに付加された累積量と自身の記憶している累積量を比較する。この時、正常に動作している計算機群は互いに同じメッセージを受信するので、２つの累積量は等しいはずである。そこで受信側計算機は、２つの累積量が異なる場合は送信側計算機が

故障したと判断して、受信メッセージを無視すると共に送信側計算機に対して故障通知を送る。また各計算機は、一定数（故障確定数）以上の計算機から故障通知を受取ると、自身が故障である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の累積量としては、メッセージの個数、長さ、チェックサム値などを用いることができる。

［定理５．５］

各計算機が累積量を正しく計算、記憶、比較する時、フェイル・ストップ放送プロトコルの故障確定数を増加することにより、各計算機は任意の確度で通信異常を起こした計算機だけの動作を、その影響が他に波及する前に停止させることができる。

（証　明）

通信異常は計算機がメッセージを送受信できなかった時、および余分に送受信した時に発生する。いずれの場合も、異常を起こした計算機が計数した累積量は正常な計算機の累積量とは異なる。二重の異常が発生した時、例えばある計算機が送信時に異常を起こして自身の累積量だけが増加して他の計算機の累積量が増加していない状態で同一の計算機が受信異常を起こすと、累積量が単なる通信回数である場合は、異常計算機と正常計算機の累積量が等しくなるが、このような事象の発生は累積量の増加量を計算機や通信内容の関数にすることによって防止することができる。従ってフェイル・ストップ放送プロトコルでは、異常計算機がメッセージを発した時は、正常計算機がメッセージを無視しメッセージ送出計算機に故障通知を返送することにより異常計算機の動作を停止させると共に異常の影響の波及を防ぐことができる。また異常計算機がメッセージを送出しない場合にも、正常計算機の発したメッセージに対して異常計算機の返送する故障通知に付加された累積量を正常計算機が比較することにより、正常計算機が異常計算機に故障通知を送りその動作

を停止させることができる。さらに、異常が恒久的なもので異常計算機がメッセージを受信できない場合はその停止は不可能であるが、異常計算機は新しいメッセージの受信はできないし、それが送出するメッセージも無視されるので、実質的にはその動作も停止したものと考えることができる。ここで計算機数をN，故障確定数をMとする時、異常計算機の動作が停止できなくなるのはM個以上の計算機が同時にまた同じように故障する場合であるので、その確率F（N，M）は次式より小さい。

$$F(N,M) \leqq {}_NC_M \cdot P^M(1-P)^{N-M} + {}_NC_{M+1} \cdot P^{M+1}(1-P)^{N-M-1} + \cdots$$

$$+ {}_NC_{N-1}P^{N-1}(1-P) + {}_NC_N \cdot P^N < N \cdot (P \cdot N)^M / M! \quad \cdots (5.1)$$

(5.1) 式でPは1つの計算機の故障確率であるが、これはNに比べて十分小さくP・N＜1と考えられる。従ってMを大きくすることにより、異常計算機の動作を停止させる確率を高くすることができる。 □

フェイル・ストップ放送プロトコルは、［定理5．5］を使って確認信号を交換することなく、メッセージの到着を確認することができる。つまり要素Aの送出メッセージmを要素Bが受信できなかった時、その原因がメッセージ・オーバラン等による要素の瞬時的な異常によるものであれば、要素Bが次にメッセージを送出する時、あるいはmを正しく受信した他の要素がメッセージを送出するときに、Bの異常としてmのBへの未到着が確認できる。Bの異常発見時には同時に、Bが最後に正しく送信または受信したメッセージに付加された累積量がわかるので、各要素が送信メッセージの記録を保存することによって必要なメッセージのBへの再送が可能であり、Bは正しい処理を継続することができる。この時、各要素の送信メ

ッセージ保存用の記憶域の大きさは回復すべき異常の継続時間に比例するが、本プロトコルではシステム中のどの要素がメッセージを送信しても異常が検出できるので、数メッセージ分の記録で（原理的には１メッセージ分）瞬時的な異常からの回復が可能である。数メッセージ以上の記録を要するのは要素に恒久的な異常がある場合であり、この場合にはどのような方法を用いても要素の交換、あるいは次節で述べる要素の多重化によってしか処理を継続することはできない。

本プロトコルは明らかに集中機構を必要としない。またメッセージ到着確認のために要素間の動作を同期させたり、各要素がシステム中の要素の数や存在位置等を知る必要はない。さらに累積量の付加や比較、送信メッセージの記録などの操作は、各要素に付加される通信用のプロセッサによって処理できるので要素の実際的な処理効率の低下は少ない。［定理５．５］の前提には、各要素が累積量を正しく計算、記憶、比較することがあるが、この前提も個々の要素の通信プロセッサを多重化することにより、他の要素に影響を与えることなく完全に局所的な操作で実現することができる。システム中の要素の同一順序でのメッセージ受信の保証も、ＬＡＮあるいはバス上に同時に流れるメッセージが１つしか存在しないので、各計算機上のＯＳ核が自計算機内へのメッセージ送出をＬＡＮまたはバスへのメッセージ送出を確認してから行なうことにより容易に実現できる。高信頼化の目的で通信路を多重化する場合や、要素の異常によってメッセージを再送する場合のメッセージ到着順序の保存も、メッセージに付加する累積量を利用することによって容易に実現できる。

第４節　放送並列多重化方式

システムの信頼性を高める方法として、同一の処理を複数の要素で並列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、処理実行中の要素が故障した段階で予備の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予備要素が通常時に処理を行なわないので、一つの要素が複数の要素の予備を兼ねたり、予備要素を常時は別の目的に用いるなど、要素の冗長度を全体として低くすることができる。しかし故障時に予備要素が処理を引継ぐのに時間がかかる、あるいは高速で処理を引継ぐには高頻度で予備要素に処理の途中結果を通報するためオーバヘッドが増加するなどの欠点がある。知的分散システムはリアルタイム制御にも用いるため、高信頼化のためのオーバヘッドの増加や故障による処理の停止時間は極力小さく抑える必要があるので並列多重処理方式を用いる。この時、宣言形のシステムの適応性や拡張性を保つには、個々の要素は多重化の有無やその程度、あるいは多重化要素の存在位置などから独立に定義および動作できなければならない。またリアルタイム・システムで要求される処理性能を得るには、多重化された要素間での同期操作などを極力省略しなければならない。

放送並列多重化はこれらの要求を満足するための方式で［４１］［４２］［４３］、図５．５に示すように放送通信によって多重化された要素群を呼出す。従って個々の要素が、関連要素の多重化の有無やその程度を知る必要がなく、多重度から独立したシステムの構築が可能になる。また多重化された要素間での同期や情報交換の必要もなく、高信頼化のためのオーバヘッドも低く抑えることができる。図５．５では、要素Ａが放送通信によって３重化された要素群Ｂ１，Ｂ２，Ｂ３を呼出し、さらにＢ１～Ｂ３の各々が放送通信によって２重化された要素群Ｃ１，Ｃ２を呼出す。各要素は多重化された要素群からメッセージを受信すると、多数決や、最初に到着したメッセージのみを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。しかし放送並列多重方式を実現するには、次の２つの性質を実現する分散機構が必要である。

(1) 正しく動作している多重化要素群は、同一のメッセージを同一の順序で受信する。
(2) 個々の要素は正しいメッセージしか受信しない。

従来提案されている方法では［４１］［４２］［４３］、これらの性質が仮定されていたが、知的分散システムではこれらを満足する分散機構を実現する。このうち性質(1)は前節で述べたフェイル・ストップ放送プロトコルによって容易に実現できるので、本節では性質(2)を実現する先着ＣＮ方式を提案する。

個々の要素が完全な自己診断機能を持てば、要素は正しいメッセージしか出力しないので性質(2)は自動的に満足されるが、高度な自己診断機能の実現は困難であるので、先着ＣＮ方式では多重化された要素からのメッセージを比較する。しかし多重化要素からのメッセージを比較する方式では通常は多数決によって正しいメッセージを抽出するが、多数決をとるには各要素が関連要素群の多重度を知らなければならない。そこで先着ＣＮ方式では、要素が定められた個数（有効個数，以降ＣＮ［Confirmation Number］と記す）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。ここでＣＮは、対応メッセージを送出する要素の多重度を超えてはならないことを除けば、各要素が関連要素群の多重度から独立に設定でき、従ってシステムの適応性や拡張性の低下することがない。また各要素は、要求される信頼度に応じて関連するメッセージのＣＮ値を大きくすることによって、メッセージの信頼度を任意に高くすることができる。さらにフェイル・ストップ放送プロトコルによって、多重化された全要素でのメッセージ受信順序が同一になるので、多重化要素間での意思決定の同一性が保証できる。つまり多重化要素間で同一内容のメッセージが有効化される。先着ＣＮ方式は、全

く同じように誤ったメッセージがＣＮ個先着した場合に誤動作をするが、誤動作確率は(5.1)式と同様の式で決まる値よりも小さく、ＣＮを大きくすることにより任意に小さくすることができる。

先着ＣＮ方式は、従来の多数決法を効率化したものと考えることができる。同様の方法としてｎ重化された要素群から受取る先着ｍ個（ｍ≦ｎ）のメッセージの多数決をとる方式が提案されているが、［４４］、そこでは多重化要素群へのメッセージ到着順序の同一性が保証されていないので、多重化された個々の要素が同一の意思決定をするとは限らない。つまりある要素がｍ個のメッセージに基づいた多数決操作をするが、他の要素がそれとは異なるｍ個のメッセージに基づいた多数決操作をする可能性がある。さらに常にｍ個のメッセージが到着するのを待たなければならない。先着ＣＮ方式で同じ信頼度を得るには、正常時（多重化された要素群に故障の無い場合）にはｍ／２あるいはｍ／２＋１をＣＮとし、ＣＮ個のメッセージ到着を待つだけでよい。

システム中の全てのメッセージのＣＮ値が対応メッセージを送出する要素多重度の１／２を超える場合には、メッセージ到着順序が要素毎に異なっても多重化要素群は同一の意思決定を行なえる。従ってフェイル・ストップ放送プロトコルを用いなくても、先着ＣＮ方式だけで多重化された各要素が正しいメッセージを漏れなく受信し同一の意思決定をすることができるが、フェイル・ストップ放送プロトコルを併用することによって、ＣＮ値が要素多重度の１／２以下の場合にも同一の意思決定ができるようになる。また故障要素が誤ったメッセージを送出する場合には、先着ＣＮ方式ではＣＮ個の同一内容を持つメッセージの到着をあきらめるまでの時間的オーバヘッドが大きくなるが、フェイル・ストップ放送プロトコルによって異常メッセージを即座に排除できるので、無駄な待合わせ時間がなくなる。

先着ＣＮ方式では、各要素は多重化された要素群の送出するメッセージの内容を比較する。従ってこの方法が正しく動作するには、

比較対象となるメッセージの識別機構が必要である。そのために各要素は送出するメッセージに識別子を付加し、多重化要素群が同一の処理過程で送出するメッセージには同一の識別子を発番するようにする。次に、以下の状況で要素（オブジェクト）が動作することを考慮した、メッセージの識別子を発番する分散機構を提案する。

(1) オブジェクト内には複数のメソッドが存在し、それらは同時に動作することができる。
(2) 多重化されたオブジェクトのメッセージ送出順序に非決定性が存在する。多重化されたオブジェクトのメソッド起動順序は、メッセージの到着順序によって決まるので全て同じであるが、複数のメソッドがメッセージを送出する順序は、タイムスライスのタイミングの違いなどによって全ての多重化オブジェクト間で一致するとは限らない。
(3) 各メソッドはリエントラントに動作することができる。

システムの処理が、例えば外部から入力信号を受信することにより１つオブジェクトが１回動作するだけで完了する簡単なものばかりであれば、外部入力に時刻印を付加し、時刻印によってメッセージを識別するような方法で十分である［４４］。しかし外部入力受信後の処理が複数のオブジェクトに分岐して行なわれたり、１つのオブジェクトが複数のメッセージを送出するような場合には、このような識別子の発番法では異なるメッセージに同一の識別子を与えてしまう。また処理順序が決定的であるならば、多重化オブジェクト群にはフェイル・ストップ放送プロトコルによりメッセージは全て同一順序で到着するので、メッセージの識別子としてはメッセージを送出するオブジェクト名とそのオブジェクトからの何回目の送出メッセージであるかを示す数字の対で十分である。しかし上述のような条件下ではそれだけでは不十分であり、ここでは次に示すような識別子の構成を考える。

メッセージ識別子＝オブジェクト名＋メソッド名
＋メソッド起動回数＋送信回数‥‥(5.2)

ここで、オブジェクト名およびメソッド名は、それぞれメッセージを送出するオブジェクトとメソッドの名前であり、メソッド起動回数はメソッドがシステム立上げ後何回目の起動で対応するメッセージを送出したかを示す。また送信回数は、メソッドが1回の起動で複数のメッセージを送出する場合、対応するメッセージが今回のメソッド起動後何回目に送出されたものであるかを示す。

多重化されたオブジェクト群の同一メソッドは、フェイル・ストップ放送プロトコルにおいては全て同じメッセージを同じ順序で受取り、同じ順序で起動されるので、メソッドがいかなるコンテキスト下（具体的には何のジョブを実行中に）で動作したかがメソッド起動回数によって特定できる。また送信回数によってメソッドがコンテキスト内のどのタイミングで送出したメッセージであるかが特定できる。従って(5.2)式によって、(1)～(3)のような状況下でも多重化された複数のオブジェクトが送出する同一のメッセージに対して同一の識別子を与えることができる。また、各オブジェクトは自己の持つ情報だけで識別子を発番できるので、その独立性を失うこともない。

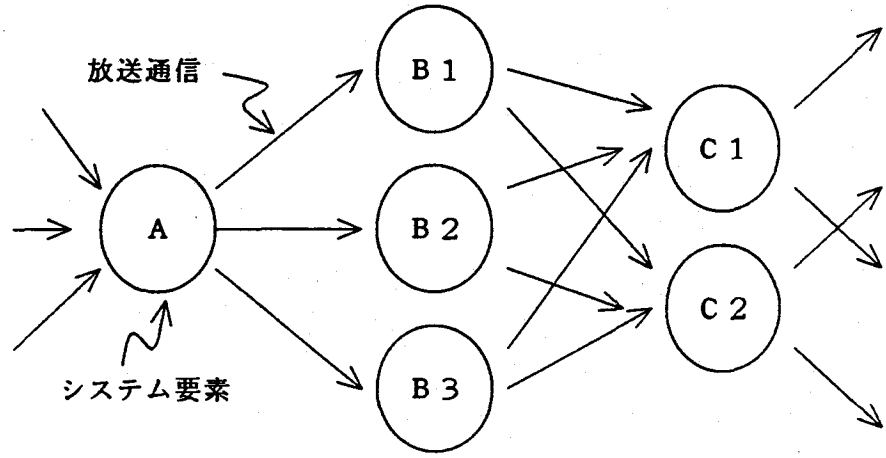


図 5.5　放送結合による並列多重処理

第５節　考察

第２節で提案したデッドロック検出、防止アルゴリズムは、第２章３節で示した宣言形システムの備えるべき３つの条件を満足する。実際個々の要素はアルゴリズムの実行に当って、全要素の存在を知る必要は無く、可到達集合や逆向可到達集合に属する要素群の存在だけを知ればよい。また、ある要素の動作とその可到達集合および逆向可到達集合に属さない要素の動作は独立で、互いに同期させる必要は無い。さらにアルゴリズムの分散形態は機能分散ではなく、各要素の独立な動作の結合から成る。アルゴリズムの性能に関しては、システム中の要素数をｎとすると、個々の要素に必要な計算量はn^2のオーダーであり、プローブ法［３２］等で要求されるｎオーダーの計算量よりも多い。しかしアルゴリズムの実行に必要な通信回数は、プローブ法がn^2オーダであるのに対して、提案アルゴリズムはｎオーダとなる。従って通信に要するオーバヘッドや（実際には個々の要素上での処理時間よりも通信処理に要する時間の方が長い）、要素群での並列処理の効果を考えると、実質的には提案アルゴリズムの方が優れている。プローブ法は、個々の要素がその隣接集合に属する要素群の存在のみを知ればよい方法であり、宣言形システムの備えるべき条件の観点だけからは提案アルゴリズムよりむしろ優れている。しかし実際には要素間での通信回数が多過ぎるため、リアルタイム制御システムのような高速応答の要求されるシステムでは利用できない。提案アルゴリズムは、通信量を抑えるために個々の要素の備えるべき情報を多くした方法と考えることができる。

第３節で述べたフェイル・ストップ放送プロトコルでは、個々の要素が通信路を流れるメッセージの累積量を計数し、送信メッセージにこの累積量を付加する。しかし、この方式をメッセージ再送によって衝突を回避するCSMA/CD 方式のＬＡＮに適用すると、既存の通信プロセッサを用いる場合次のような問題が発生する。つまり、

同時に２つの計算機が各々Ａ，Ｂなるメッセージを送信しようとする時、実際にはＢがＡの受信後に送信されるにもかかわらず、ＢにＡの受信前の累積量が付加されてしまうことである。このため正常な通信であるにもかかわらず累積量の不一致が発生する。これは既存の通信プロセッサを使うのでＯＳ核がＬＡＮへのメッセージ送出を直接制御できないために、メッセージへの累積量の付加タイミングがＬＡＮ上に実際にメッセージを送出するタイミングと異なるからであり、各計算機が送信元計算機毎に累積量を把握することによって回避することができる。

つまり累積量を拡張して、各計算機が図５．６に示すテーブルの形で一定期間保持する。テーブルの各行Ｃｉは、ある時刻ｉに送出されたメッセージの拡張累積量を表わし、各々個別部と合計部とから成る。拡張累積量の個別部ａｉｊは、時刻ｉまでに計算機ｊから送出されたメッセージを特徴付ける量の累積量であり、合計部Ａｉはａｉｊのｊに関する和、つまり時刻ｉまでに送出されたメッセージの拡張前の累積量である。計算機ｊのＯＳ核は、送信時ｉには最新のＣｉをメッセージに付加し、送信完了時ｋ＋１（Ｃｉを付加したメッセージの送信要求からＬＡＮ上への送出終了までの間に、他計算機からメッセージを受信しなければｉ＝ｋであるが、メッセージを受信した場合はｋ＞ｉとなる）にはＣｋ内のａｋｊとＡｋに１を加えたＣｋ+1をテーブルに追加する。またメッセージ受信時には、各計算機が受信メッセージに付加されたＣｍと一致する拡張累積量がテーブルに存在するか否かを調べる。この時メッセージの累積量に通信回数を用いる場合は、Ｃｍ中のＡｍとテーブルに登録された最新の拡張累積量Ｃｑ中のＡｑを比較し、その差（Ａｑ－Ａｍ）だけ離れた行が比較対象Ｃｐとなる。Ｃｐが存在し、かつ全てのｎに対してａｐｎ＝ａｍｎとなる時、メッセージ送出元の計算機が正しく動作しているとみなし、テーブルの最新行Ｃｑ内のａｑｙ（メッセージ送信元計算機をｙとする）とＡｑに１を加えた拡張累積量Ｃｑ+1をテーブルに追加する。

このように送信元計算機毎の拡張累積量の履歴を保持することにより、複数計算機からの通信要求の衝突によって、ＯＳ核の制御と無関係にメッセージの再送が行なわれても、受信メッセージが正常なものであるか否かを正しく判定するとができる。また拡張累積量に合計部を設けることにより、受信メッセージの拡張累積量の比較対象を拡張累積量の履歴からただちに求めることが可能になる。また、拡張累積量の履歴保持量はシステム内の計算機の数程度で十分である。但しこの方法では、システム内の計算機の最大数があらかじめわかっていなければならなず、完全な宣言形システムを実現するには通信プロセッサの制御も直接ＯＳ核が行なう構成が必要である。

フェイル・ストップ放送プロトコルでは、全計算機上でのメッセージ受信順序の同一性を保証するのに、通信路中に同時には一つのメッセージしか流れないことを利用したが、この性質は計算機間を複数の通信路で接結する場合は満たされない。しかしより規模の大きいシステムを構築する際には、例えば複数のＬＡＮを用いて通信能力を強化するようなことが多くなる。このような状況下でのメッセージ到着順序の同一性保証機構の確立が今後の課題である。

第４節では放送並列多重化方式を提案したが、リアルタイム制御等以外の分野では処理速度よりも経済性の重視される場合も多い。従って冗長度の削減が可能な待機冗長形における、位置や多重度から独立なシステム構築を可能にする方式の確立が今後の課題となる。待機冗長形では、主系の異常時に予備系が処理を引継ぐために、事前に定めたチェックポイントにおける主系の状態を予備系に転送する。従って各要素に予備系の存在やチェックポイントに関する知識が必要になるなど多重度から独立したシステムの作成は並列多重化方式の場合よりも困難であるが、チェックポイントにおけるデータの送出や受信を行なう共通のメソッドを個々のオブジェクトの共通知識に組入れる等の手段で位置や多重度からの独立化も可能である。チェックポイントの設定等を、多重化を意識しないプログラムに組

込む効果的な方式が重要になる。

	通信累積量 合計部	通信累積量個別部						
		1	2	3	……	j	……	n
C_1	A_1	a_{11}	a_{12}	a_{13}	……	a_{1j}	……	a_{1n}
C_2	A_2	a_{21}	a_{22}	a_{23}	……	a_{2j}	……	a_{2n}
・	・	・				・		・
・	・	・				・		・
C_i	A_i	a_{i1}	a_{i2}	a_{i3}	……	a_{ij}	……	a_{in}
・	・	・				・		・
・	・	・				・		・
C_m	A_m	a_{m1}	a_{m2}	a_{m3}	……	a_{mj}	……	a_{mn}

図 5.6　通信累積量テーブル

第５節　結言

本章では知的分散システムの個々の要素を、要素間の競合がある場合や要素に異常の存在する場合にも、矛盾無く動作させるための分散機構を提案した。第２節では要素間での競合によって生じるデッドロックの検出、防止に関する分散アルゴリズムを提案した。従来から多くの分散アルゴリズムが提案されているが［３２］［３３］［３４］［３５］、いずれの方法も要素間で必要になる通信の回数が要素数の２乗に比例し、通信オーバヘッドが高くてリアルタイム制御等の高速応答の要求されるシステムでは実際には使えなかった。提案アルゴリズムは要素数に比例する通信回数しか必要とせず、高速応答の要求されるシステムでの利用も可能である。従ってリアルタイム制御システム等では、従来デッドロックの発生しないように設計上の制約や、設計段階における特別の工夫が必要であったが、提案アルゴリズムによりこれらの必要性が削減できる。また従来の方式ではデッドロックの防止と検出の併用にはかなり強い制約があったが［３６］、ここではこれらの制約を緩和することもできた。

効果的な多くの分散アルゴリズムが放送通信の存在を前提にしており、放送通信は分散システムの構築に不可欠な機構であるが、従来は放送メッセージの受信先への到着を確認する効果的な方法が存在せず、実際のシステムでは放送通信が利用されることは少なかった。第３節では、送受要素間で確認信号の交換をすることなく、放送メッセージの到着確認が可能なフェイル・ストップ放送プロトコルを提案した。従来の同種の方法では集中機構の存在を仮定したり、メッセージ到着確認のためのオーバヘッドが高かったが［３７］～［４０］、提案手法により分散機構による低オーバヘッドの高信頼放送通信が可能になる。従って既に提案されている多くの分散アルゴリズムの利用が可能になり、宣言形システムの実現が容易になる。

第４節では宣言形システムの適応性や拡張性を損わず、しかも処理効率の高いシステムの高信頼化方式である放送並列多重化方式の実現基盤を確立した。放送並列多重化方式では、多重化された個々の要素が互いに情報を交換することなく非同期に動作する。従って多重化要素の位置や多重度から独立したシステムの実現が可能であるが、そのためには多重化された各要素が、同一の順序で同一のメッセージをしかも正しいメッセージだけを受取る性質が前提になる。従来提案された方式ではこれらの性質が仮定されていたが、［４１］［４２］［４３］、ここではフェイル・ストップ放送プロトコル等を用いることによりこれらの性質を実現する分散機構を提案した。

第６章　知的分散システムの適用

第１節　序　　言

本章では、鉄道運行管理と電力系統制御の２つの分野における知的分散システムの適用を述べる。これらのシステムは、利用者ニーズの高度化に伴なって、増々大規模かつ複雑化しており、集中的な機構によって監視、制御するのが困難になって来ている。また、これらのシステムの故障が社会に与える影響は大きく、さらに路線や系統の拡大、運用形態の変更などシステムは質的、量的に常に変化しており、信頼性や拡張性、保守性の面からも分散化が求められている。そのため多くの分散化の試みがなされているが［６］［７］［８］［９］、いずれも単なる階層化システムであったり、マネージメント・バイ・コンサンサス［２２］と呼ばれるシステム全体の状態を把握する集中機構を多重に持つような形態であり、完全な分散システムの構築までには致っていない。

ここで鉄道運行管理と電力系統制御とは次のような意味で対照的である。即ち、電力系統が個々の部分の状態が密接に影響し合う本質的に大域的、集中的なシステムであるのに対して、鉄道運行管理システムは局所的、分散的である。従って、システム分散化に当っての課題も異なり、電力系統制御では局所的な情報しか持たない要素群の全体目的達成のための協調動作が中心課題であり、鉄道運行管理システムでは、分散した装置群を様々な観点から動的に統合するシステム構造の実現が問題となる。そこで第２節では、第５章で述べた知的分散ＯＳを用いることによって、鉄道システムの個々の装置に対応するプログラムを分散計算機上の任意の計算機に配置し、必要に応じてそれらを任意の形で統合し、しかもリアルタイム制御システムとして、要求される時間内に確実に応答するようなシステムの構築が可能になることを示す。また第３節では、電力系統における電圧一定化を考え、大域的情報を必要とするシステムであって

も、第４章で述べたビディング法等を用いることによって管理、制御の分散化が可能であることを示す。

第２節で扱う鉄道運行管理システムは、実システムでは無いが、機能的にも規模的にも実システムと変わらないモデルを扱っており、列車走行を模擬するハードウェア・シミュレータを用いた実験によって、実用化までの技術的確認ができた。第３節で扱う電力系統電圧一定化制御は、電力系統管理・制御の中の極く一部であり、モデルも非常に単純化してあるが、ビディング法等の拡張・変形による大域的・集中的なシステムにおける要素群の協力と協調による分散管理・制御の可能性を示すことができた。

第２節　鉄道運行管理システム

鉄道運行管理システムは、軌道走行中の列車群を監視し、それら列車群の種別や行先、あるいは列車間隔に基づいて信号機、転てつ機を制御することにより、列車の安全で正しい運行を保証する。図6.1に運行管理システムの動作を示す。軌道上には数十ｍから数百ｍの長さの軌道回路と呼ばれる電気回路が連続的に配置されており、列車が軌道回路上に存在すると回路の状態はＯＮになり、列車が通過してしまうとＯＦＦになるようになっている。運行管理システムは、これら回路群の状態から個々の列車の走行位置を検出する。しかし、軌道回路の状態からは列車の存在位置を知ることはできるが、個々の列車を識別することはできない。そこで運行管理システムはダイヤ情報を保持しており、列車の位置とダイヤとの対応を常時監視することにより個々の列車を識別する。さらに識別した列車が分岐点等に接近すると、列車の行先きや種別に応じて信号機や転てつ機を制御する。

これらの動作の制御は、初期の鉄道システムでは個々の装置が存在する現地で分散的に行なわれていたが、より高度で複雑な機能を実現するため列車の状態を中央で集中的に監視し、制御する方式が主流になって来た。しかし、鉄道の大規模化と複雑化によって利用者ニーズは益々高度化し、現在では次のような性質を持つシステムが望まれるようになっている。

1．列車密度の増加等に応じてシステムの能力を逐次向上できる。
2．路線の延長、駅の追加や拡大要求に容易に対応できる。
3．部分的な故障によって、システム全体の機能の停止することが無い。
4．必要に応じてシステムの信頼性を容易に向上できる。

しかし、現在の運行管理システムは集中形の計算機システムとし

て構築されているため、これらの要求に柔軟に適応することは難しい。例えば、列車や駅等の状態はそれを表わす情報テーブルにまとめて集中的に管理されるが、テーブル・サイズや負荷の増加に対応する目的で、これらを分割するのはあまり簡単なことではない。そこで本節では、知的分散システムを適用することにより、これらの性質を実現する。つまり、運行管理に必要な機能や情報をオブジェクト・モデルにおけるオブジェクトの集合として宣言的に定義することにより、１～４に示した要求に答える。　図6.2に鉄道運行管理システムの構成を示す。図は従来中央監視室等に設置されていた、計算機システムを LAN で接続した計算機群で構成し、これら計算機上に入出力，ダイヤ，区間，列車，駅の５種類のオブジェクトを分散配置した構成を表す。この時、これらのオブジェクト群は互いに放送通信を介して情報を交換するが、第５章で述べたオブジェクトの位置独立性と放送通信の結合独立性、さらには放送並列多重処理によって各オブジェクトは、どの計算機に存在、移動しても、また何重に多重化されても正しく動作することができる。

ここで、入出力オブジェクトは軌道回路や信号機等の状態を計算機システム内に取込むとともに、これら装置に動作指令を送出するオブジェクトである。単一の入出力オブジェクトがシステム内の全装置をカバーする必要はなく、複数のオブジェクトで監視、制御すべき装置を分担する。また、分担範囲はオブジェクト間で重複していても問題はない。ダイヤ・オブジェクトは１日の標準ダイヤを記憶するオブジェクトで、ディスク中の年間ダイヤ・データから運転日に応じたダイヤを取込み（曜日等によって一般にダイヤは異なる）、列車が始発駅を発車する時刻の一定時間前になると次に述べる列車オブジェクトを生成する。また、操作員の指示に従って、ダイヤの修正を行なう。ダイヤ・オブジェクトも入出力オブジェクトと同様に、単一のオブジェクトが全ダイヤを記憶する必要はなく、複数のオブジェクトが路線毎等で範囲を分担する。また、列車オブジェクトの生成は、列車テンプレート（クラス）・オブジェクトのインス

タンスを作ることによって行なうが、インスタンスの生成される計算機はビディング法によって最も負荷の低いｎ個（ｎ重化する場合）に決まる。

次に列車オブジェクトは個々の列車に対応しており、列車の位置と列車ダイヤを記憶している。列車ダイヤは列車の種別と行先、および通過あるいは停車すべき駅，番線とその時刻の列であり、この情報によって各列車オブジェクトは信号機や転てつ機の制御指令を作ることができる。区間オブジェクトは、軌道上の適当に定めた分岐の無い連続部分に対応し、区間内に存在する軌道回路の情報とその上の在線列車、および隣接する区間を記憶している。ここで区間の持つ軌道回路情報には装置制御コードを付加することができ、例えば、列車が信号機コードの付加された軌道回路を通過する時に、対応する列車オブジェクトに信号機を制御するタイミングを知らせる等のことが可能になる。最後に駅オブジェクトは個々の駅に対応しており、駅が管理する信号機や転てつ機、案内装置等の状態を記憶している。これらのオブジェクトの内、列車オブジェクトだけは列車運行に応じて動的に発生・消滅し（列車オブジェクトは終着駅に到着すると消滅する）、その配置場所は各計算機の負荷が均等になるように決められるが、他のオブジェクトは固定的なものでその配置もオブジェクト定義時にオペレータが設定する。

これらオブジェクト群の動作を図6.3に示す。まず１日の始発時刻前にオペレータの指定によって、ダイヤ・オブジェクトがディスク中のダイヤ・ファイルからその日のダイヤの担当部分を読み込み、読込んだダイヤ中の最も早い列車の始発時刻より少し前の時刻を指定して、自オブジェクトに列車生成を依頼するメッセージを送出する。さらにダイヤ・オブジェクトは指定時刻にメッセージを受け取ると、列車テンプレート・オブジェクトに最も早く始発駅を出発する列車のダイヤ・データを送り、対応する列車を自身のダイヤから削除するとともに、残ったダイヤ上で最も早い列車の始発時刻より少し前の時刻を指定して自身に新たな列車生成を依頼するメッセー

ジを送出して処理を終了する。列車テンプレート・オブジェクトは、ダイヤ・オブジェクトから列車のダイヤ・データを受取ると、ビディング過程を経て、最も負荷の低いｎ個の計算機上に、受取ったダイヤ・データを持つ列車オブジェクトを生成する。

列車オブジェクトは、このように対応する列車の始発時刻直前に逐次生成されるが、生成されると自身のダイヤ・データを基に現在位置（軌道回路の名前）を対応する区間オブジェクトに通知する。次に列車が発車して軌道上を移動すると、入出力オブジェクトがそれに応じた軌道回路の状態変化を検出し、その軌道回路を含む区間オブジェクトに状態変化を知らせる。すると区間オブジェクトは、区間内に在線する列車群の位置関係から状態変化を起した列車を特定し、自身が記憶している列車位置を更新する。状態変化を起した列車は次のように特定することができる。つまり、新たにＯＮ状態になった軌道回路ａ上に存在する列車は、状態が変化する直前にａから列車進行方向の反対側にあり、かつａに最も近い軌道回路上に在線した列車である。また、このような列車が存在しない場合は、新たな列車が区間に進入したことを示している。そのため区間オブジェクトは、隣接する区間毎に進入予定列車の列を記憶するテーブルを保持する。このテーブルの内容はこの区間の直前に位置する区間オブジェクトが、列車進行方向に隣接区間が１つしかない（分岐の無い）場合には、新しい列車の進入時にその列車がいずれ隣接区間に進入することを通知することによって更新される（分岐のある場合については後で述べる）。

このようにして位置変化を発生した列車を特定すると、区間オブジェクトはその位置と、そこへ指定された制御装置コードを対応する列車に通知する。そして列車位置と制御コードを受取った列車オブジェクトが、自身の現在位置を更新し、制御コードが意味のあるものであれば、列車ダイヤ上の次到着駅オブジェクトに制御コードと列車の進路を、さらに必要ならば遅れ時間などの駅での案内に必要な情報を通知する。そして最後に制御コードを受取った駅オブジ

エクトが、制御コードに対応する装置への制御指令を、入出力オブジェクトを経て駅装置へ送る。また、制御指令が転てつ機の転換命令である場合は、列車の進路に対応する区間オブジェクトに対して列車の進入予告を通知する。

これらの動作からもわかるように、本システムは列車オブジェクトの分散配置等の機構も含めて、集中管理部は全く存在しない。従って、負荷ネックの排除とともに信頼性ネックの排除も可能になる。つまり、システムのどの部分が故障しても残りの部分は処理を継続できる。またオブジェクト群は、放送通信によって結合するので、各オブジェクトはどの計算機上に配置、移動されても全く同じように動作でき、さらに第５章で述べた放送並列多重処理によって、これらオブジェクトの多重度は任意に変更することができる。実際、この節の最初に示した性質１～４の実現を次のように確かめることができた。

１．処理性能の逐次向上

列車運行密度の増加に伴なって、運行管理システムの負荷も増加するが、前述のようなオブジェクト構成によって、単なる計算機の追加によって負荷増大に対応することができた。つまり、区間オブジェクトあるいは列車オブジェクトは互いに並列に処理を実行することができるので、負荷の増加に応じて計算機を単に追加し、オブジェクト群を再配置することにより、システム応答時間の短縮を図ることができた。さらに知的分散ＯＳの持つオブジェクト・マイグレーション機能によって、オブジェクトの再配置はシステムの運転中にも可能である。

２．路線延長、駅追加への適応

路線延長や駅の追加は、対応する区間や駅オブジェクトの追加定義によって実現できた。この時、既存の区間オブジェクトとの接続関係等を定義するため、追加対象以外のオブジェ

クトの部分的な変更も必要になるが、一時的に変更前と後の2つのバージョンのオブジェクトを並列に動作させることにより、実験の範囲ではオブジェクトの追加もシステムの運転を止めずに実現することができた。

3．信頼性に関するネックの排除

システムのどの部分に異常があっても残りの部分が、具体的にはどの計算機が故障しても、他の計算機上のオブジェクト群が正しく動作することを確認できた。もちろん正常な計算機上のオブジェクトであっても、異常が発生したオブジェクトからのメッセージ受信を前提にした動作は不可能である。さらに計算機群を結合するLANは、唯一の例外で信頼性に関するネックになり得るが、LANの2重化とフェイル・ストップ放送プロトコルによって片系のLANに異常があっても正しく処理を継続することができた。

4．システム信頼性の逐次的向上

放送並列多重処理によって、各オブジェクトの多重度設定を任意に変更できるようになった。この多重度の変更はシステム運転中にも可能である。さらにフェイル・ストップ放送プロトコルによって、異常の発生をただちに発見し、その影響が他に影響するのを防ぐことができた。異常を発生したオブジェクトは、一旦動作を止めるが計算機の動作回復後、オブジェクト・マイグレーション機能によって、その状態を正常なオブジェクトと矛盾しないように設定し直して処理を続行することができる。

知的分散システムの適用によって、機能的には以上のような利点の確認ができたが、一方では、オブジェクト・モデルや放送通信を採用したことによる処理効率低下の恐れがある。そこで次に、知的分散鉄道運行管理システムの処理性能を考える。表6.1は、i80386(16MHz)をCPUとして持つ計算機群を10M bit／secのEthernetで接続

し、さらに各計算機にi80186とi82586から成るLANプロセッサを付加した場合の知的分散ＯＳの基本性能を表わす。

コンテクスト・スイッチ時間は、0.2 msecである。この数値は動的結合や多種類のメッセージ交換モード、さらには多重メッセージ処理等の機能を持つため、インテル社がi80386用ＯＳとして製造しているiRMXの0.17 msecよりも少し劣るが、ほぼ同等である。実際タイムスライスの間隔を20 msecとした場合のオーバヘッドの差は、1秒当り1.5 msecである。オブジェクトがメッセージを出してから、それが他のオブジェクトに到着するまでの時間は、２つのオブジェクトが同一サイトにある場合は、最高速の通信性能を持つＯＳの１つと言われるＡmoeba［１８］のRPCと同等である。Ｖ［１５］は知的分散ＯＳやＡmoebaの約倍程度の速度を持つが、これはＶが通信機能をRPC専用に特化したためと考えられる。２つのオブジェクトが異なるサイトに存在する場合には、Ａmoebaが圧倒的な速度を示すが、これは内部バスの速度や通信プロセッサの速度差によるものと思われる。実際i80186で行っている通信処理をi80386で実行するだけでも３倍近い速度向上が図れるはずである。ここで、サイト間通信の遅れはサイト内に比較するとかなり大きいが、そのほとんどはLANプロセッサの負荷であり、計算機本体CPUのオーバヘッドはサイト内通信と同じである。オブジェクト生成とオブジェクトのサイト間の移動の時間は、オブジェクトのサイズを40Kbyteとする時、各々18.2msecと2.3 secであるが、オブジェクト移動に際して移動対象オブジェクトの処理が実際に停止する時間は5msecの近辺の値になるはずである（実際には測定できなかった）。

このような基本性能を持つLANと計算機４台から成る鉄道運行管理システム全体の性能を、列車走行シミュレータに接続して、測定した結果を次に示す。列車の同時在線台数を 160台、駅数（但し、制御駅と呼ばれる転てつ機制御等が必要な駅だけ）を60駅とし、かつ全てのオブジェクトを完全に２重化した場合の性能を測定した結果、各計算機の負荷率は50％以下に抑えることができた。この列車走行

シミュレータの規模は、主都圏の地下鉄3路線分を超えるものであり、測定結果はオブジェクト・モデル採用によるオーバヘッドがあっても、知的分散システムが十分実用に耐え得るものであることを示している。実際1つの状態変化に対する計算機内部での処理時間（軌道回路の状態変化を検知してから信号機等へ出力信号を送出するまでの時間）は40msecでありまた複数の状態変化が同時に起こって処理が重なる場合でもそれは平均320msecに納まる。通常の鉄道システムに要求される処理時間は計算機処理および現地装置との通信遅れ等を含めて5sec前後であるので、これらの値は十分なものと考えることができる。また、列車オブジェクトの生成や計算機故障回復時等のオブジェクト移動における処理停止時間も十分小さい。

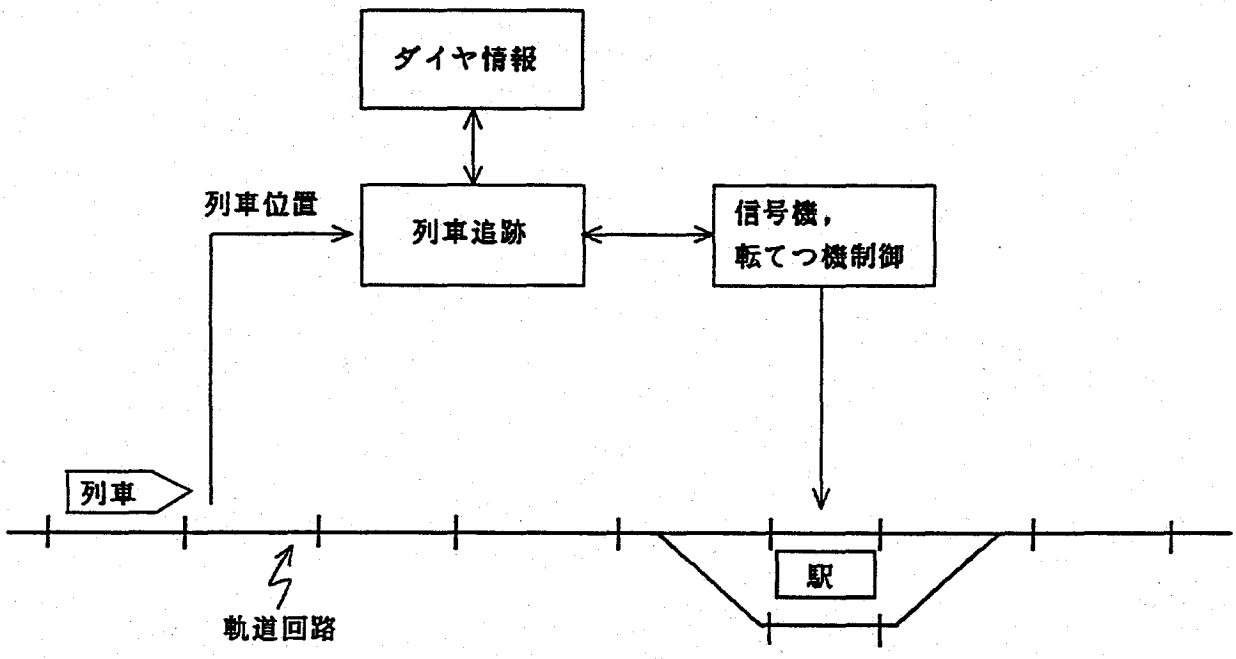


図６．１　鉄道の運行管理

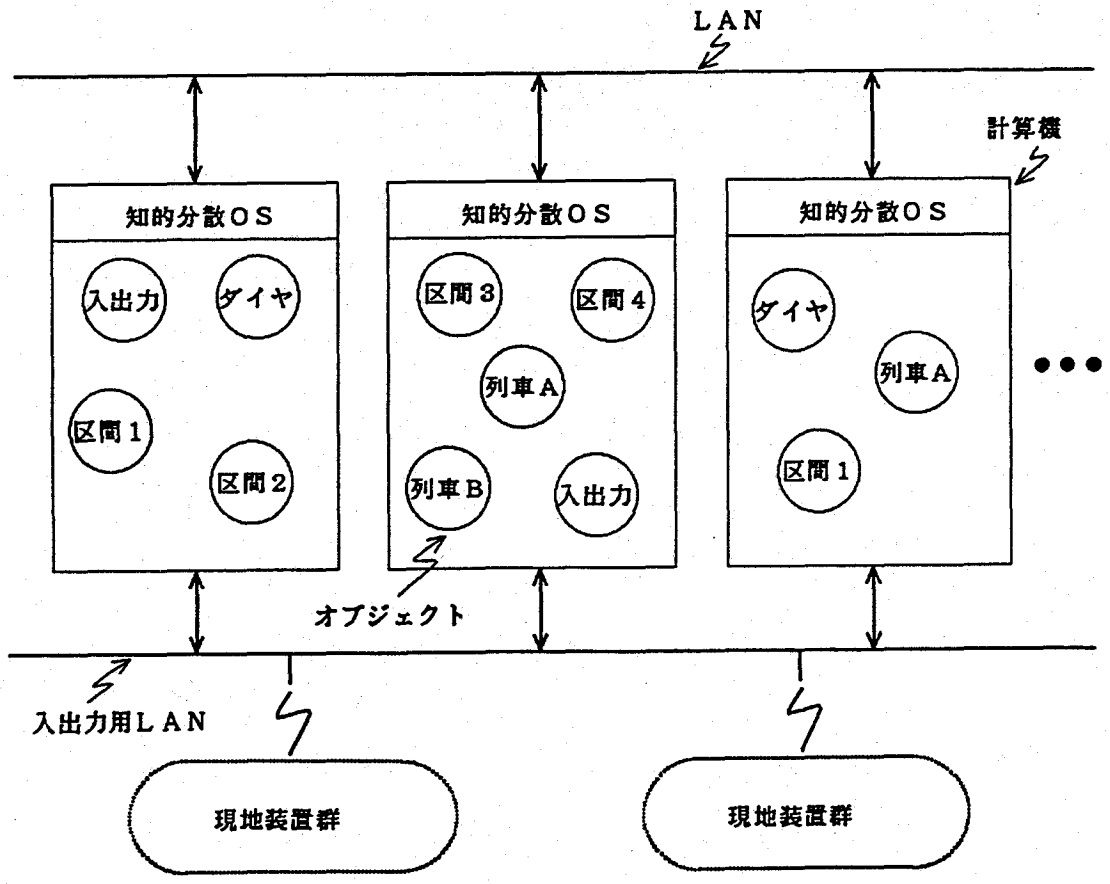


図６．２　知的分散運行管理システムの構成

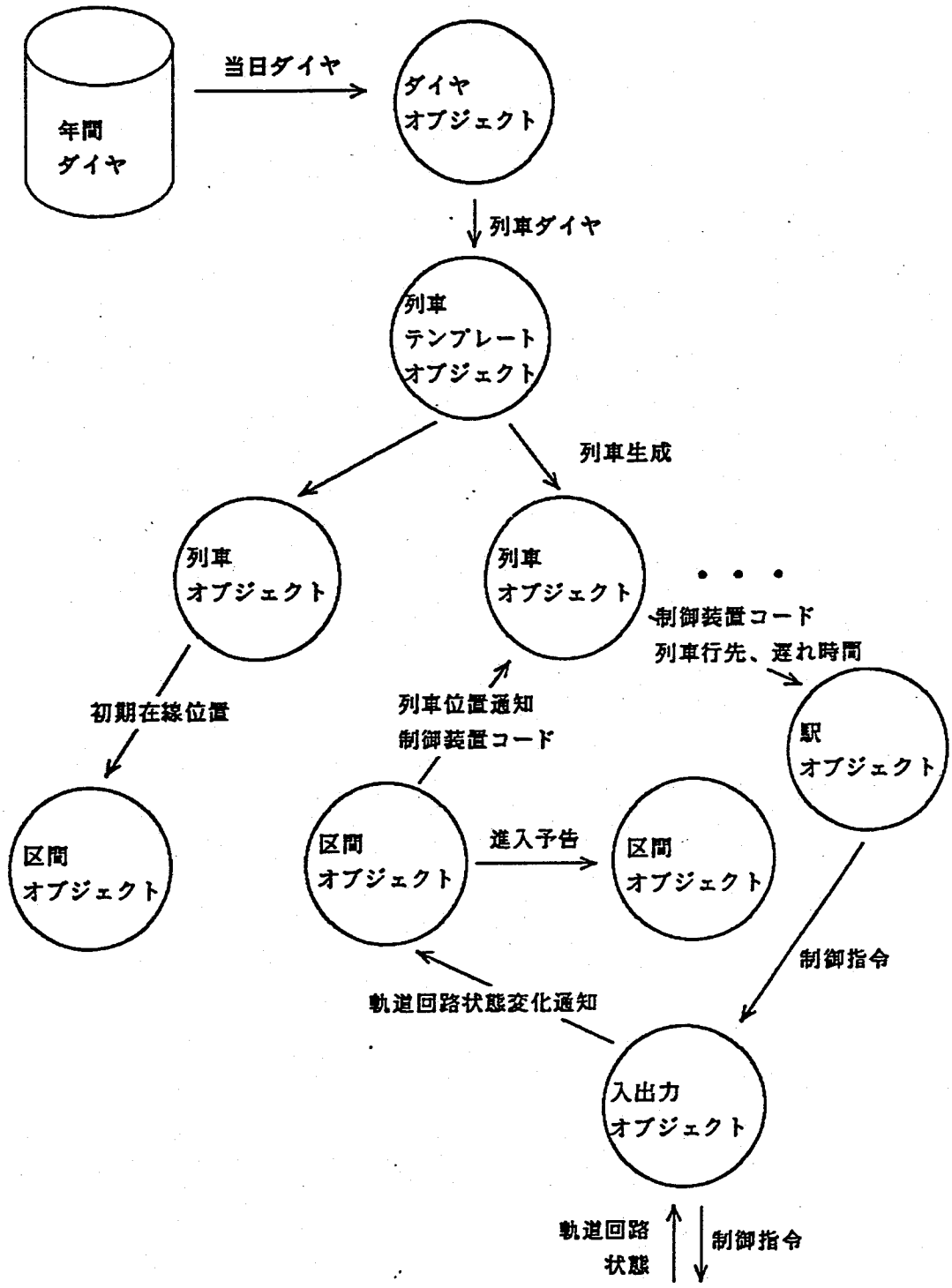


図６．３　鉄道運行管理オブジェクトの動作

表６．１　知的分散ＯＳの基本性能

機能	知的分散ＯＳの処理時間	他のＯＳの処理時間
コンテクスト・スイッチ	0.2msec	iRMX　0.17msec
オブジェクト間通信 （同一サイト内）	0.9msec (4byte)	Amoeba 0.8msec V　0.5msec
オブジェクト間通信 （異サイト間）	3.8msec (4byte)	Amoeba 1.4msec V　2.5msec
オブジェクトの生成	18.2msec (40Kbyte)	
オブジェクト移動	2.3sec (40Kbyte) 停止時間は5msec以下	

ＣＰＵ　：i80386(16MHz)
通信プロセッサ　：i80186
i82586

第３節　電力系統の電圧制御

鉄道運行管理システムへの知的分散適用の主な目的は、ハードウェア構成から独立した柔軟なソフトウェアの実現性、有用性を示すものであり、システム要素間の協力と協調機構の評価ではない。つまり、鉄道運行管理システムは元来局所的、分散的な性質を備えたものであり、列車オブジェクトの生成時に計算機間で負荷を調整する目的以外には要素間の協調動作の必要は無かった。そこで本節では、システム要素間の協力・協調動作の有効性を確認するために、電力系統電圧一定化制御への知的分散システムの適用を考える。

電力系統の電圧は、系統内の各部位に設けられた装置の出力を調整することによって制御されるが、従来はこれらの装置は互いに全く独立に分散して制御されていた。しかし電力系統は大域性、集中性の高いシステムであり、個々の部位の状態が互いに強く影響し合うので、電力系統の大規模、複雑化とともに装置間の協調動作による運用の効率化が望まれている。そのためこれら装置群を集中的に管理する方法が検討されているが、系統全体の拡張性や保守性、信頼性の観点からは分散方式の方が優れている。そこで本節では、第４章で述べたビディング法を用いることにより、これら装置群の協力と協調による電圧の分散制御が可能になることをシミュレーションで示す。電圧一定化制御のシミュレーションに用いた電力系統を図6.4に、また系統中の各種定数値を表6.2に示す。

(0)～(10)のノードから成る系統であり、ノード(7)～(10)には電力源が、またノード(2)～(5)には装置０～装置３の無効電力補償装置が存在する。無効電力補償装置は系統の電圧を一定に保つための装置であり、装置設置位置付近の電圧低下を無効電力を注入することによって防止する。ところが、各無効電力補償装置は異常時だけに動作するものであり、装置コストを抑えるために個々の装置の容量を小さくする要求がある。従って各装置の無効電力注入量には上限があり、その範囲を超えるような電圧低下に対しては、電圧異常

発生位置以外の装置も無効電力を注入しなければならない。そこで、容量の小さい装置群で多重に発生する電圧異常に対処するには、各装置への無効電力注入量割当てを、各装置の注入量の余力がなるべく大きくなるように決める問題が生じる。これは最適化問題の１つであるが、計算量の問題や電圧安定性に関する現象の解明が不十分なこともあって、集中方式においてもまだ十分な性能を持つ機構は実現できていない。ここで提案する方法では、個々の装置がビディングに基づき発見的・経験的な知識を交換することによってこの機構を実現する。次にビディング手順を示す。

［ビディング手順１］

（１）各無効電力補償装置ｉ（以後では単に装置ｉと呼ぶ）は、自装置端の電圧Ｖ（ｉ）が許容値Ｖｒ（ｉ）よりも低くなると、無効電力注入量をｋ｛Ｖｒ（ｉ）－Ｖ（ｉ）｝だけ増加させる。さらに装置ｉは、この操作をＶ（ｉ）がＶｒ（ｉ）以上になるか、あるいはは注入量が制限値に達するまで繰返す。

（２）装置ｉの注入量が制限値に達してもＶ（ｉ）がＶｒ（ｉ）よりも小さい時は、装置ｉは他の装置群に応援依頼を放送する。

（３）装置ｉからの応援依頼を受取った装置ｊは、装置ｉへの貢献度Ａ（ｉ，ｊ）を計算し装置ｉに返答する。

（４）装置ｉは他の装置群から貢献度を受取ると、最大の貢献度を返答した装置ｊ*に電圧低下量｛Ｖｒ（ｉ）－Ｖ（ｉ）｝を通知する。

（５）電圧低下量｛Ｖｒ（ｉ）－Ｖ（ｉ）｝を受取った装置ｊ*は、その無効電力注入量をｋ｛Ｖｒ（ｉ）－Ｖ（ｉ）｝だけ増加させる。

（６）装置ｉは、Ｖ（ｉ）がまだＶｒ（ｉ）より小さい場合は（２）からの操作を繰返す。

上述の手順（３）における装置ｊの装置ｉに対する貢献度Ａ

（ｉ，ｊ）は次式に従って計算する。

$$A(i,j) = \frac{dV(i)}{dq(j)} \{q_M(j) - q(j)\} \quad \cdots\cdots\cdots (6.1)$$

但し、ｑ（ｊ）は装置ｊの現在の無効電力注入量であり、q_M（ｊ）は装置ｊの最大無効電力注入量である。つまり（6.1）式の右辺第１項は、装置ｊの無効電力注入による装置ｉでの電力上昇感度、また第２項は装置ｊの注入量の余力となる。従って、手順（５）において貢献度が最大となる装置ｊ*が無効電力を注入することは、注入量余力が大きくかつ注入効率の最も高い装置に注入負荷を割当てることになるので、個々の装置の注入量の余力をなるべく均等化して大きくする割当てを実現する。

表6.3は図6.4における装置３の電圧が許容値より小さくなった場合の手順実行結果である。表における状態０は各装置の初期状態であり、装置３の電圧が０．９１９６で許容値０．９２２０よりも小さくなっている。また各装置の無効電力注入量は装置０～３が各々０．０４００、０．０２９５、０．０８２２、０．１０００であり、装置３の注入量は上限値０．１０００に達している。装置０～２の注入量の上限値は各々０．５０００であり、現在の注入量はまだ上限に達していない。表における状態１は手順実行後の状態である。まず、状態０において装置３は自身の電圧低下を発見し、無効電力の注入を試みるが、既に注入量が上限に達しているので装置０～２に応援依頼を放送する。放送を受取った各装置はその注入量余力と装置３に対する感度から貢献度を計算して装置３に返答するが、装置３は最大の貢献度０．０１５６０＝０．０３７３３×（０．５－０．０８２２）を持つ装置２を選択する。そして最後に装置２が無効電力注入量を０．１５９７まで増加することによって装置３の電圧が０．９２２５となり許容値を上回る。

ここで個々の装置の電力上昇感度は、装置が動作する毎に各装置が自身に対する影響を観測することによって決定する。表6.3の場合には、装置3は状態0に到達するまでの過程で、装置0～2が無効電力注入量を各々0から0．0400、0．0295、0．0822に変えた時に感度0．02181、0．01368、0．03761を計算している。例えば装置0の状態0から状態1への電圧上昇量は0．9631－0．9612＝0．0019であるが、これは状態0における装置0の装置2に対する電圧上昇感度0．02405に装置2の無効電力注入量0．1597－0．0822＝0．0775をかけた値0．00186とほぼ等しく、このように設定した感度が妥当なものであることがわかる。

ビディング手順1では各装置の貢献度を感度×装置の注入量余力とし、貢献度最大の装置が比較的少量の無効電力注入を繰返すことによって個々の装置の注入余力の均等化を画ったが、感度の精度がこのように比較的高い性質を利用すると、次のようにより正確でかつ高速な注入量割当て方式が実現できる。

［ビディング手順2］

（1）各装置iは、自装置端の電圧V（i）が許容値Vr（i）よりも低くなると他の装置群に{Vr（i）－V（i）}と各装置kの注入量に対する電圧上昇感度を添えて応援依頼を放送する。

（2）応援依頼を受取った装置jは、{Vr（i）－V（i）}と各装置kの注入量に対する装置iの電圧上昇感度V（i，k）を記憶し、自身の注入量余力R（j）を他の装置群に放送する。

（3）他の装置群から注入量余力を受取った装置jは、R（j）よりも大きい注入量余力を放送した各装置kに対して、Vu（j）＝ Σ V（i，k）×{R（k）－R（j）}を計算する。ここでVu（j）＞Vr（i）－V（i）ならば無効

メッセージを、またＶｕ（ｊ）≦Ｖｒ（ｉ）－Ｖ（ｉ）ならば｛Ｖｕ（ｊ），Ｒ（ｊ）｝を他の装置群に放送する。

（４）｛Ｖｕ（ｊ），Ｒ（ｊ）｝を受取った装置ｋは、最大のＶｕ（ｊ）を送った装置ｊ*を見つけ、Ｖｕ（ｋ）≦Ｖｕ（ｊ*）ならば、Ｒ（ｋ）－Ｒ（ｊ*）で決まる量の無効電力を注入する。但し自身がｊ*である時は、｛Ｖｒ（ｉ）－Ｖ（ｉ）－Ｖｕ（ｊ*）｝／Ｖ（ｉ，ｊ*）の無効電力を注入する。

手順２による無効電力注入量の割当てが図6.5のようになることは容易にわかる。つまり無効電力補償装置を注入量余力の大きいものから順に左から並べた時、余力の大きいｊ*番目までの装置の余力が等しくなように、かつｊ*＋１番目の装置の余力よりも小さくならないような注入量割当が求まる。従って感度の精度が正しければ手順１よりも完全な注入余力の均等化が可能になる。また手順１のように微小な無効電力注入を繰返すことが無いので電圧回復までの時間も短縮できる。尚、本節では電圧低下の回復しか考えなかったが、電圧上昇への対応も負の無効電力を注入することにより全く同様に行える。

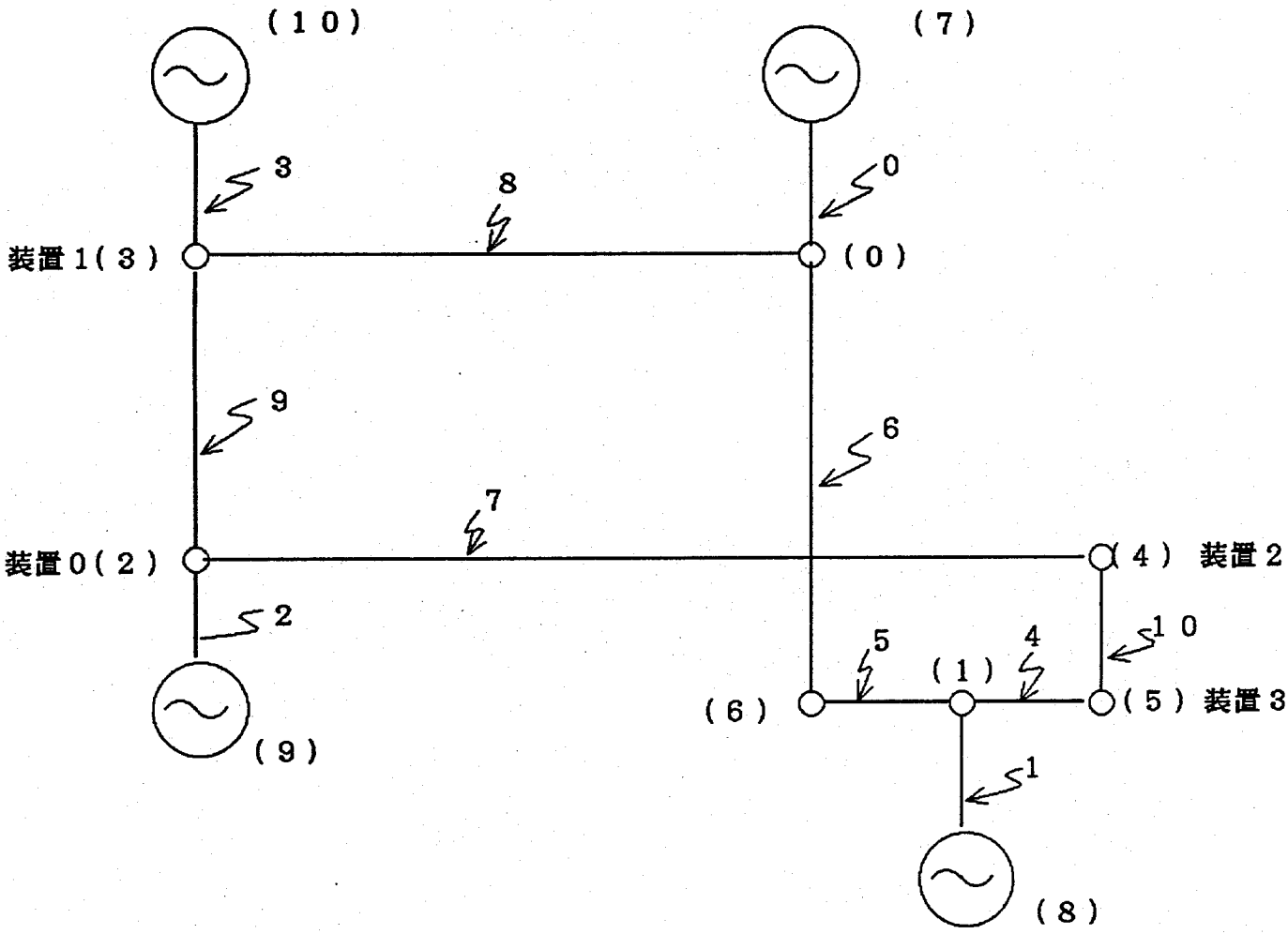


図6．4　シミュレーションに用いた電力系統

表６．２　シミュレーション系統の定数

ブランチ番号	線路リアクタンス(PU)	対地サスセプタンスの1/2(PU)
0	0.038	
1	0.037	
2	0.055	
3	0.085	0.05
4	0.028	0.10
5	0.013	0.04
6	0.040	0.06
7	0.030	0.06
8	0.100	0.05
9	0.045	0.07
10	0.008	0.02

表６．３　無効電力注入量と電圧の変化

装置０

状態番号	V(0)	q(0)	dV(0)/dq(0)	dV(1)/dq(0)	dV(2)/dq(0)	dV(3)/dq(0)
0	0.9612	0.0400	0.03075	0.01764	0.02293	0.02064
1	0.9631	0.0400	0.03075	0.01764	0.02293	0.02064

装置１

状態番号	V(1)	q(1)	dV(0)/dq(1)	dV(1)/dq(1)	dV(2)/dq(1)	dV(3)/dq(1)
0	0.9919	0.0295	0.01713	0.03462	0.01362	0.01256
1	0.9930	0.0295	0.01713	0.03462	0.01362	0.01256

装置２

状態番号	V(2)	q(2)	dV(0)/dq(2)	dV(1)/dq(2)	dV(2)/dq(2)	dV(3)/dq(2)
0	0.9225	0.0822	0.02405	0.01472	0.04172	0.03733
1	0.9257	0.1597	0.02376	0.01453	0.04127	0.03692

装置３

状態番号	V(3)	q(3)	dV(0)/dq(3)	dV(1)/dq(3)	dV(2)/dq(3)	dV(3)/dq(3)
0	0.9196	0.1000	0.02181	0.01368	0.03761	0.04145
1	0.9225	0.1000	0.02181	0.01368	0.03761	0.04145

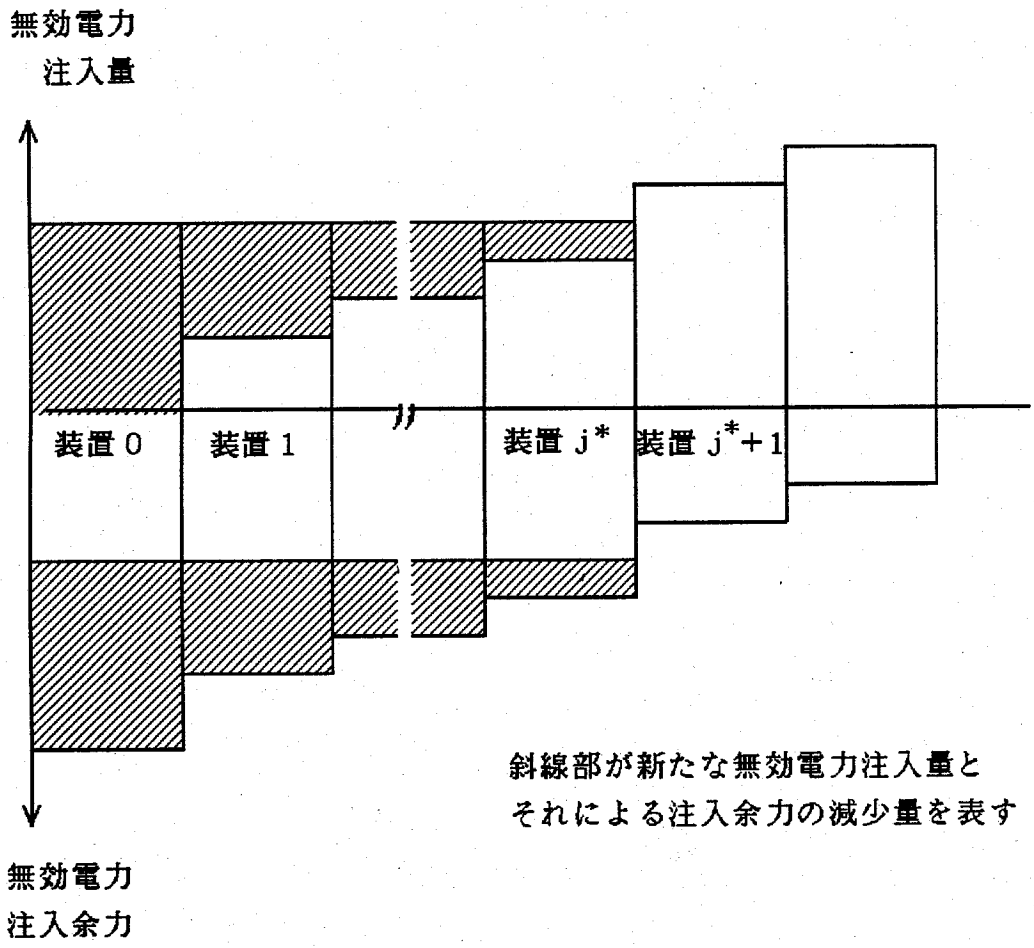


図６．５　ビディング手順２による無効電力注入割当て

第４節　考　察

第３節では電力系統の電圧一定化制御に際して、第４章２節で述べた、ビディング法を用いることによって、系統内に分散した無効電力注入装置群の注入余力平準化を行った。第３節前半では通常のビディング法を用い、後半では両方向ビディング法を第４章４節で示したように資源要素間での直接の情報交換を許す形に変形して用いたが、後半で用いた方法によって図6.5に示したように非常に良好な負荷割当てを実現することができた。これは各無効電力注入装置が、他の装置と直接情報を交換することによって他の全ての装置の状態を知ることができるので当然の結果である。しかし一方では、第３節ビディング手順２の（３）（４）において複数装置上での並列処理の効果を引き出せたものの、各装置が他の全ての装置に対する電圧上昇感度を記録する必要があったり、全装置からのメッセージ到着を待つため大域的な同期が必要になるなど、宣言形システムとしては望ましくない性質もある。またビディング手順２の（３）では全装置が放送によってメッセージを送出するため、通信量の増加が余儀なくされた。

従ってこの方法は宣言形のシステムにとっては一般的には好ましくないが、マルチプロセッサ計算機における個々のプロセッサへの負荷割当て等には効果を発揮する。即ち、第３節では各無効電力注入装置が他の全ての装置に対する電力上昇感度を記憶する必要があったが、プロセッサ群への負荷割当の場合には、各プロセッサは自身の負荷と、自身より大きい最小負荷および自身より小さい最大負荷、さらには自身より小さい負荷を持つプロセッサの個数とそれらプロセッサ群の総負荷だけを記憶すればビディング手順２と等価の処理が実現できる。従って各プロセッサは他のプロセッサ上での負荷の流入や消滅の都度これらの量を更新すればよく、ビディングの度に全プロセッサがその状態を放送で通知し合う必要はない。そのためビディング過程で個々のプロセッサが全プロセッサからのメッ

セージを待つなどの大域的な同期の必要が無くなり、要素間の通信量も小さく抑えることができる。また各プロセッサは、他のプロセッサ群の状態を前述のように、システム中のプロセッサ数等から独立して集約した形で記憶することができ、宣言形システムへの性質が満たされるようになる。

このように、両方向ビディング法は問題に応じて変形、改良が可能であるが、より一般の場合でのこのような方向での両方向ビディング法の変形については今後の課題としたい。

第５節　結　言

第２節では知的分散システムのリアルタイム制御への応用として鉄道運行管理システムを扱かった。その結果、オブジェクト・モデルによるプログラム間通信の多用や制御アルゴリズムの分散化による処理速度の低下が予想されたが、応答速度が１０ｍｓｅｃ程度以上のシステムならば十分対応できることがわかった。またシステムの知的分散化による拡張性や信頼性、処理性が予想通り向上することが確認できた。特に処理性については、鉄道システムでは線路上の区間毎あるいは列車毎に並列に実行できる処理が多く、分散化によって非常に大規模な鉄道システムを扱かうことが可能になった。鉄道運行管理システムは元来、局所的、分散的な要素が多いため、管理・制御アルゴリズムを分散化するために特に工夫するような点は少なかったが、ここでは考えなかった運転整理機能（列車運行に遅れが生じた時に、列車を間引いたり行先きを変更する機能）などは大局的な情報を必要とするため今後検討したい。また将来の鉄道システムとして、列車間で協力・協調しながら状況に応じてその運行形態を決定する無ダイヤ・システム等も考えたい。

第３節では協力・協調による問題解決の例として電力系統の電圧一定化制御を扱かった。電力系統制御の中の非常に限られた範囲だけの検討であり、モデルも単純化してあるが、このような方法による電力系統全体の管理と制御の分散化の可能性を示すことができた。分散化に際しての１つの鍵となるのは発見的・経験的な知識である。第３節でも感度の利用や、ビディング方式の採用など厳密なモデルから導びかれる量や方式以外のものを取入れたが、電力系統を実際に分散管理する場合には個々の要素が予測できないような量がさらに多くなり（理論的には予測可能であっても、センサの量や通信量、計算量等が膨大になる）発見的・経験的な手法の重要性が増す。

第 7 章　結 論

要素群の単なる集合からなる宣言形のシステムとして知的分散システムの構造を提案し、要素群に分散した協力と協調機構およびそれらの機構を実現する知的分散ＯＳの構造と機能を示した。また鉄道運行管理や電力系統の電圧一定化制御に適用することにより、知的分散システムがシステムの処理能力や拡張性、信頼性を高めるのに有効であることを示した。次に今後の課題をまとめる。

まず第一は知的分散システム実現機構の拡張である。現在はLANで接続した計算機群から成るハードウェアを考えているが、広域に分散したシステムを扱うにはWANその他による結合を考えなければならない。またより高速で大量の処理を行うには、個々の計算機のマルチ・プロセッサ化も重要であり、これらに対応した知的分散ＯＳの拡張が必要である。

第二の課題は協力と協調機構の改良である。第６章における電力系統の電圧一定化では単段スケジューリング法を適用したが、ＦＡシステム等では多段スケジューリング法が必須である。しかし第４章で述べた方法ではまだ必要とされる計算量や通信量が十分小さいとは言えず、より効率的な方法を検討したい。また要素群の協力機構も、現在はプロダクション・システムの延長であるため本質的には全数探索に基礎を置いている。従ってその限界は見えており、根本的に異なったアプローチが必要である。協力と協調は学習と並んで情報処理の分野では今後最も重要になる課題の一つであると考えている。

最後に第三の課題は応用分野に関するもので、単段スケジューリングだけでなく多段スケジューリングも含めた協調機構の有効性を実証するには、ＦＡシステム等への本格的な適用が必要である。また協力機構の実証には、自律的なデータ群の単なる集まりからなる分散データベース・システム等への適用が重要である。

謝辞

本研究をまとめるに当たって、直接理解ある御指導と励ましをいただいた情報工学科の宮原秀夫教授に心から感謝致します。また本研究をまとめるに当たり、御助言と御指導をいただいた制御工学科の坂和愛幸教授、情報工学科の都倉信樹教授、橋本昭洋教授、西尾章治郎助教授に感謝致します。

本研究の機会と御指導、励ましをいただいた株式会社東芝、システム・ソフトウェア技術研究所の西島誠一所長、新井政彦部長に感謝致します。さらに本研究の遂行に当たって、有益な助言と討論および励ましをいただいた同研究所の岡宅泰邦主任研究員、西村和夫主任研究員、遠藤経一主任研究員、関俊文主事、長谷川哲夫主事、原嶋秀次主事、永瀬恵子主事、北村麻子主事、古澤均主事に心から感謝致します。

文 献

［１］分散と協調特集，計測自動制御学会学会誌，Vol.26,No.1,1987.

［２］シナジェティクス特集，電気学会論文誌，Vol.C107,No.11,1987.

［３］H.Haken,"Synergetics,An Introduction,Nonequilibrium Phase Transitions and Self-Organization in Physics, Chemistry and Biology,"Springer-Verlag,1977.

［４］D.E.Rumelhart,"Parallel Distributed Processingl,2," MIT Press,1986.

［５］J.J.Hopfield,et al.,"Newral Computation of Decisions in Optimization Problems,"Biol.Cybern.,1985,pp.141-152.

［６］長谷川，他，「列車運行管理の分散化」，第２０回鉄道サイバネ論文集，1983.

［７］長谷川，「トラフィック制御システムにおける分散処理」，計測と制御，Vol.26,No.1,1987,pp.57-61.

［８］D.Barrie,et.al.,"Computer Configuration for Ontario Hydro's New Energy Management System,"IEEE Trans.on Power System,Vol.4,No.3,1989,pp,927-934.

［９］D.J.Chapman,et.al.,"A Case Study in Change from a Centralized Energy Management System to a Decentralized One, "IEEE PICA'89,1989,pp.446-452.

［１０］舟木，他，「プロセス制御用計算機システムの高信頼化への道のり」，電気学会論分誌，Vol.C107,No.4,1987,pp.341-348.

［１１］森，他，「自律分散概念の提案」，電気学会論文誌，Vol.C104,No.12,1984,pp.303-310.

［１２］L.D.Ermann,et al.,"The HEARSAY-II Speech-Understanding System:Integrating Knowledge to Resolve Uncertainty,"Computing Surveys,Vol.12,No.2,1980,pp.213-253.

[１３] R.G.Smith,"A Framework for Distributed Problem Solving,"UMI Research Press,1981.
[１４] R.G.Smith,"The Contract Net Protocol:High-Level Communication and Control in a Distributed Problem Solver,"IEEE Trans.on Com.,Vol,C-29,No.12,1980,pp.1104-1113.
[１５] D.R.Cheriton,"The V Distributed System,"CACM,Vol.31,No.3,1988,pp.314-333.
[１６] J.Ousterhout,et al.,"The Sprite Network Operating system,"IEEE Computer,Vol.21,No.2,1988,pp.23-36.
[１７] M.Accetta,et al.,"Mach:A New kernel Foundation for UNIX Development,"Proc. of USENIX 1986 Summer Conference,1986,pp.93-112.
[１８] S.J.Mullender,et.al.,"Amoeba a Distributed Operating System for the 1990s,"IEEE COMPUTER,May,1990,pp.44-53.
[１９] A.Newell,et al,,"Human Problem Solving,"Prentice-Hall,1972.
[２０] H.W.Kuhn,"The Hungarian Method for the Assignment Problem,"Naval Res_Logist,Quart.,No.2,Vol.83,1955,pp.83-97.
[２１] 伊理，他，「グラフ・ネットワーク・マトロイド」，産業図書，1986.
[２２] M.P.Mariani,"Distributed Data Processing Technology and Critical Issues,"North-Holland,1984.
[２３] T.C.Hu,"Parallel Sequencing and Assembly Line Problems,"Oper.Res.,Vol.9,No.6,1961,pp.841-848.
[２４] E.G.Coffman,et.al.,"Optimal Scheduling for Two Processor Systems,"Acta Informatica,Vol.1,1972,pp.200-213.
[２５] M.J.Gonzalez,"Deterministic Processor Scheduling,"Computing Surveys,Vol.9,No.3,1977,pp.173-204.
[２６] 岩田，他，「作業者および代替機械を考慮したジョブショップ・スケジューリング」，機械学会論文誌，Vol.47,No.417,C,

1981,pp.709-716.
［２７］石井，「スケジューリングの近似解法」，オペレーションズ・リサーチ，Vol.31,No.29,1986,pp.29-34.
［２８］J.H.Blackstone,et.al.,"A State of the Art Survey of Dispatching Rules for Manufacturing Job Shop Operations,"Int. J,Prod,Res,Vol.20,No.1,1982.
［２９］室津，他，「代替機械を考慮した大規模ジョブショップ・スケジューリング問題の一解法」，機械学会論文誌，Vol.47, No.418,C,1981,pp.803-810.
［３０］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,pp.1020-1035.
［３１］R.C.Holt, "Some Deadlock Properties of Computer Systems," Computing Surveys, Vol.4, No.3, 1972,pp.179-196.
［３２］K.M.Chandy,et al., "Distributed Deadlock Detection," ACM Trans. Computer Systems, Vol.1, No.2, 1983,pp.144-156.
［３３］S.S.Isloor,et al., "An Effective ON-LINE Deadlock Detection Technique for Distributed Database Management Systems," Proc. COMPSAC'78, 1978,pp.283-288.
［３４］R.Obermarck, "Distributed Deadlock Detection Algolithm," ACM Trans. Database Systems, Vol.7, No.2, 1982, pp.187-208.
［３５］E.Knapp, "Deadlock Detection in Distributed Databases," Computing Surveys, Vol.19, No.4,1987,pp.303-328.
［３６］J.H.Howard Jr., "Mixed Solution for the Deadlock Problem," Communications of the ACM, Vol.16, No.7, 1973, pp.427-430.
［３７］J.Chang,et al., "Reliable Broadcast Protocols," ACM Trans. Computer System, Vol.2, No.3, 1984,pp.251-273.
［３８］M.Takizawa, "Cluster Control Protocol for Highly Reliable Broadcast Communication," Proc. of the IFIP

Working Conf. on Distributed Processing, 1987,pp.431-445.

[39] P.M.Melliar-Smith,at.al,"Fault-Tolerant Distributed Systems Based on Broadcast Communication,"Proc.of the 9th Intr.Conf.on Distributed Computing System,1989,pp.129-134.

[40] S.Mishra,et.al.,"Implementing Fault-Tolerant Replicated Objects Using Psync,"Technical Report of Arizona University,TR89-3,1989.

[41] E.C.Cooper, "Replicated Distributed Programs, "Proc. of the 10th ACM Symp. on Operating System Principles, Operating System Review, Vol.19, No.5, 1985,pp.63-78.

[42] D.Powell,et al., "The Delta-4 Approach to Dependability in Open Distributed Computing Systems, "Proc. on the 18th International Symp. on Fault Tolerant Computing, 1988,pp.246-251.

[43] R.F.Cmelik,et al., "Falut Tolerant Concurrent C: A Tool for Writing Fault Tolerant Distributed Programs," ibid., 1988,pp.56-61.

[44] K.Mori, et al., "Autonomous Decentralized Software Structure and Its Application, "Fall Joint Compouter Conf., 1986,pp.1056-1063.