分散スケジューリングの方式

——知的分散システムのスケジューリング機構——

正 員 田 村 信 介 (東 芝)
 非会員 岡 宅 泰 邦 (東 芝)
 非会員 関 俊 文 (東 芝)
 非会員 古 澤 均 (東 芝)

1. ま え が き

ニーズの多様化と高度化に伴って、固定的な機能を果たすだけでなく、運用目的の変更や機器故障などの環境変化に適応するシステムが必要になっている。また大規模、複雑なシステムを容易でかつ短期間に開発する方法が求められている。自律形の分散システムは、このような要求を満足する有力な手段の一つである⁽¹⁾。つまり自律形の分散システムでは集中管理機構は存在せず、個々のシステム要素自身がその動作を管理するので、要素の追加や変更が他の部分に与える影響が減りシステムの拡張性が向上する。また各システム要素が集中管理機構に束縛されることなく、その能力を最大限に発揮できるので適応性が向上する。システムの部分的故障が全体故障につながることもない。これらの性質は、更にシステム開発労力や運転負荷の特定部分への集中を防ぎ、システムの開発と運用を容易にする。

知的分散システムはこのような性質を実現する分散アーキテクチャである⁽²⁾。システム全体の動作は、個々のシステム要素が互いに協力、協調することによって管理され、管理に集中機構が介入することはない。ここで協力とは、同一目的のための要素間の結合であり、具体的には単一の要素で処理できないジョブを単

一の要素が処理可能なプロセスの系列に分解することである。また協調は異なる目的間の利害の調整機能であり、具体的には協力で求めたプロセス間での資源に関する競合の解消を意味する。本論文では、知的分散システムにおけるこの要素間の協調機能を実現する分散スケジューリング法について述べる。既に述べたように、このとき何らかの集中機構がスケジューリングを実施することは許されない。

スケジューリング問題の解法としては、従来から分枝限定法を用いる方法が多く提案されている⁽³⁾⁽⁴⁾。しかしこれらの手法は、その実行負荷が膨大でしかもそれが集中管理機構に集中するので大規模システムの動的スケジューリングには適さなかった。そのため大規模、動的スケジューリング法としては、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が⁽⁵⁾⁽⁶⁾、ルールがあまりにも経験的、発見的であるため効率的なスケジュールの生成は期待できなかった。ここでは従来方法のこのような欠点を取り除き、大規模で動的なスケジューリングに適用でき、かつ、効率的なスケジュールを生成する方式を提案する。提案方式では、個々の要素が互いにその状態を交換することにより、自身の動作を自律的かつ並列的にスケジュールする。従って、集中管理機構を排した高い拡張性と適応性、信頼性をもつシステムの実現が容易になる。

2. 問題の定式化

知的分散システムに投入されるジョブ J は、協力機構によってプロセスの系列 $\{p_j(1), p_j(2), \dots, p_j(n_j) | n_j \geq 1\}$ に分解される。 J は系列中のプロセスがこの順に処理されたとき終了する。このとき、ある時刻で最も早く処理される J のプロセスを、その時刻にお

A Method for Distributed Scheduling—Scheduling Mechanisms of Intellectual Distributed Processing System—, By Shinsuke Tamura, Member, Yasukuni Okataku, Non-member, Toshitsumi Seki, Non-member & Hitoshi Furusawa, Non-member (Systems & Software Engineering Laboratory, Toshiba Corporation).

田村信介: 正員, (株)東芝 システム・ソフトウェア技術研究所
 岡宅泰邦: 非会員, (株)東芝 システム・ソフトウェア技術研究所
 関 俊文: 非会員, (株)東芝 システム・ソフトウェア技術研究所
 古澤 均: 非会員, (株)東芝 システム・ソフトウェア技術研究所

ける J の先頭プロセス、 $p_j(n_j)$ を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。個々のプロセスはシステム中の資源と呼ばれる要素が処理するが、資源は同時に二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はできない。また、個々のプロセス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。更に、ここでは代替資源のない場合、つまりプロセスを処理する資源が固定的に決まっています、かつ一つしかない場合を考える。代替資源のある場合の考察は第5章で行う。

このような前提で、与えられたすべてのジョブが終了する時刻を最小にするスケジュールを考える。つまり、各資源に割当てられたプロセスの処理順序を投入済みのすべてのジョブの終了する時刻が最小になるように決定する。このとき、次の二つの制約を課す。第1は特定機構の巨大化やそこへの負荷の集中を避けるため、スケジュールを要素間で分散的に行うことである。この分散は、更に個々の要素が単にスケジューリングのある定められた部分機能を受けもつような形態ではなく、各要素に関連する情報を集めて自身の動作を自律的に決定する形態でなければならない。第2は、システム運転中のジョブ投入計画の変更や、機器故障にも柔軟に対応できるように、動的スケジューリング機能をもつことである。これらの制約を満たすことによって、初めて拡張性や適応性、更に信頼性の高いシステムの構築が可能になる。

次章では上述のような制約を満たす、つまり個々の資源に関連する情報を集め、与えられたジョブに応じて自身の動作を自律的に管理するスケジューリング法を考えるが、そこで必要になる量あるいは用語を以下に定義する。

〔割当てプロセス集合 $\mathcal{A}_i(R)$ 〕 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 $\mathcal{P}_i(R)$ 〕 資源 R が時刻 t に処理可能なプロセス集合と、これらプロセスの処理時間の最小値を m とするとき、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、次に処理するプロセスとして $\mathcal{P}_i(R)$ 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 $\mathcal{S}(A)$ 〕 プロセス集合 A の後続プロセスを割当てられた資源の集合。

〔最終プロセス処理資源集合 $\mathcal{F}(A)$ 〕 プロセス集合 A の最終プロセスを割当てられた資源の集合。

図1 プロセス群の連続処理区間

Fig. 1. Continuous execution intervals.

〔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 のすべ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 $S_i(P)$ を、時刻 t における P の最早開始時刻と呼ぶ。また $S_i(P)$ から $S_i(P)$ 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。ここでプロセスが最優先で処理されるとは、その直前プロセスが終了すると、待ち時間な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、あるいは時刻 t 以後に R に割当てられている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とき、その結果つながった処理区間を時刻 t における R の連続処理区間と呼ぶ。図1に連続処理区間の例を示す。

3. ジョブ終了時刻最小化ルール

〈3・1〉 プロセスの評価関数 第2章で示したような制約を満たすスケジューリング法として、ディスパッチング・ルールを用いる方法がよく知られている。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、処理候補プロセス群から次に処理すべきプロセスを選択する方法である。SPT では処理時間が最短のプロセスを、また MWKR ではジョブの残り処理時間が最大になるプロセスを選択する。従って、個々の資源は自身に割当てられたプロセスに関する情報だけを用いて、各々独立に次に処理すべきプロセスを決定する。また決定に必要な計算量も少なく動的スケジューリングに適している。しかしこれらの方法では、SPT や MWKR などのルールとジョブ終了時刻の関係に必然性がなく、効率の良いスケジュールを生成するのは困難であった。

本論文では、各資源が処理候補プロセス集合の個々

のプロセスに対して、そ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、計算した評価値が最小になるプロセスを次に処理するプロセスとして選択する方法、ジョブ終了時刻最小化ルールを提案する。提案手法では、各資源がプロセスの評価値を計算するにあたって、資源間で情報を交換することにより他の資源に割当てられたプロセス群の情報も利用する。従って、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・ルールよりも優れたスケジューリングの生成が可能になる。しかもスケジューリングはディスパッチング・ルールと同様に個々の資源の自律動作によって分散、並列的に行われ、また必要な計算量も分枝限定法などと比べて大幅に削減される。

ここで、時刻 t における資源 R の処理候補プロセス P に与える理想的な評価関数 $T_1(t, R, P)$ は、時刻 t に R が P を処理する仮定のもとで、すべ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。つまり個々の資源 R があるプロセスの処理を終了した時刻 t に、次に処理するものとして $T_1(t, R, P)$ を最小にするプロセスを選ぶことにより、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 $T_1(t, R, P)$ を求めるには、 R が最初に P を処理する条件のもとですべての資源上での処理順序の可能性を考える必要があり、膨大な計算量が要求される。そこで、ジョブ終了時刻最小化ルールでは、資源 R は時刻 t にあるプロセスの処理を終えると、 $T_1(t, R, P)$ の近似値である $T_A(t, R, P)$ 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。

〈3・2〉 プロセス評価関数の近似 $T_1(t, R, P)$ を求めるには、資源 R の最初の処理プロセスを P に固定したすべ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。この計算量を削減するため、ここでは $T_1(t, R, P)$ を次のような順で近似する。まず〔近似1〕では、 $T_1(t, R, P)$ の近似値 $T_1(t, R, P)$ の算出に際して、個々の資源 R' は R が最初に P を処理する条件下で、 R' に関するプロセス群の終了時刻だけを最小化する。 $T_1(t, R, P)$ はその結果の R' に関する最大値として求める。

〔近似1〕 個々の資源 R' は R が時刻 t 以後最初に P を処理する条件のもとで、 R' に割当てられたプロセス群 $\mathcal{A}_i(R')$ の最終プロセスを処理する資源群 $\mathcal{F}(\mathcal{A}_i(R'))$ のすべ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。このとき、資源群 $\mathcal{F}(\mathcal{A}_i(R'))$ に割

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、 $\mathcal{A}_i(R)$ と $\mathcal{A}_i(R')$ の後続プロセスの最早開始時刻は、 $e_1(t, R', P)$ 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 上のプロセス P' の処理がその処理順序によって τ 時間遅れたとき、 P' のすべての後続プロセスの最早開始時刻も τ だけ遅らせる。 R' がこのように $e_1(t, R', P)$ を求めたとき、 R は近似値 $T_1(t, R, P)$ を次のように計算する。

$$T_1(t, R, P) = \max_{R' \in \mathcal{S}(\mathcal{A}_i(R)) \cup R} e_1(t, R', P) \quad \dots\dots\dots (1)$$

□

〔近似1〕では個々の資源 R' は、資源群 $\mathcal{F}(\mathcal{A}_i(R'))$ 上だけでの処理終了時刻を、しかも $\mathcal{A}_i(R)$ 、 $\mathcal{A}_i(R')$ の後続プロセス以外のプロセス（この集合を $\mathcal{B}_i(R, R')$ と書く）はすべ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、 $T_1(t, R, P)$ を求める場合に比べて、計算量は大幅に削減できる。資源群 $\mathcal{F}(\mathcal{A}_i(R))$ 上でのジョブ終了時刻だけから $T_1(t, R, P)$ を計算すると、異なる資源が選択した次処理プロセスの後続プロセス処理要求が特定の資源に集中し、全体として処理効率が低下する恐れがある。〔近似1〕は、 $T_1(t, R, P)$ の計算に際して、 R 以外の資源 R' の最終プロセス処理資源群 $\mathcal{F}(\mathcal{A}_i(R'))$ 上でのジョブ終了時刻も取入れることにより、後続プロセスの処理も考慮した全体的な最適化を可能にしている。

更にプロセス群 $\mathcal{B}_i(R, R')$ 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、(2)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。

$$T_1(t, R, P) \leq T_1(t, R, P) \quad \dots\dots\dots (2)$$

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 を処理した後システム中のすべてのジョブが終了するまでには、(3)式で求まる $v(t, P)$ 以上の時間が必要である。従って

$$v(t, P) = \max_{\substack{P' \in \mathcal{F}_1(R) \\ P' \neq P}} u(P') \quad \dots\dots\dots (3)$$

$T_1'(t, R, P) = v(t, P) + t$ を $T_1(t, R, P)$ の近似値と考えることができるが、 R が T_1' 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ぶことに他ならない。つまり MWKR ルールは、 T_1 を T_1' で近似したジョブ終了時刻最小化ルールと等価である。しかし、このとき明らかに $T_1'(t, R, P) \leq T_1$

(t, R, P) となり、(2)式と合すると近似値としては T_1 のほうが T_1' よりも優れていることがわかる。すなわち、 $T_1(t, R, P)$ 、 $T_1'(t, R, P)$ を最小化するプロセスを各々 P_1, P_2 とすると、 $T_1(t, R, P_1) \geq T_1'(t, R, P_1)$ 、 $T_1(t, R, P_2) \geq T_1'(t, R, P_2)$ となるので、 R が P_1 あるいは P_2 を選択した場合、全資源上での処理終了時刻は各々 $w(P_1) = T_1(t, R, P_1)$ 、 $w(P_2) = T_1(t, R, P_2)$ 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 $w(P_1) \leq w(P_2)$ となる。ただし、これは T_1' を用いたスケジューリングでのジョブ終了時刻が T_1 を用いた場合のジョブ終了時刻の下限値よりも必ず大きいことを示すだけで、 T_1 を用いるスケジューリング性能が常に優れていることを示すものではない。

〔近似1〕におけるプロセス群 $\mathcal{B}_i(R, R')$ が最早開始時刻で処理可能になる仮定のもとでは、 R および $S(\mathcal{A}_i(R))$ 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、(1)式で実際に $e_1(t, R', P)$ を計算するのは、 R および資源群 $S(\mathcal{A}_i(R))$ だけで十分である。〔近似1〕では、このように e_1 を計算すべき資源 R' の範囲は限定できるが、 R' は割当てられた全プロセスの処理順序を考慮しなければならない。〔近似2〕では、個々の資源がスケジューリングに際して考慮すべきプロセスも限定する。また〔近似3〕では、これらの近似で必要になる計算量を更に削減する。

個々の資源が考慮すべきプロセスの範囲限定は次のような方針に基づいて行う。つまり、(1)式で R' が $e_1(t, R', P)$ 算出時に仮定したプロセス処理順序は、 R' 以外の資源が仮定する処理順序との競合や、新たなジョブ投入のため、 R' が実際にプロセスを処理する順序とは異なる。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、 R' が算出する(1)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 $\tilde{\mathcal{S}}_i(R)$ が構成する連続処理区間に含まれるプロセスの集合を $\tilde{\mathcal{S}}_i(R)$ 、 $\tilde{\mathcal{S}}_i(R)$ とその後続プロセスから成る集合を $\mathcal{C}_i(R)$ とする。また R 上での P を先頭に固定したプロセス群 $\tilde{\mathcal{S}}_i(R)$ の処理順序 q_P を仮定して、 $\mathcal{C}_i(R)$ のプロセス最早開始時刻を q_P に従って遅らせるとき、 R' に割当てられたプロセスで、その最早開始時刻が $\mathcal{C}_i(R)$ 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 $\mathcal{C}_i^*(R, R', q_P)$ とする。このとき $e_1(t, R', P)$ の近似を、 $\mathcal{C}_i^*(R, R', q_P)$

の最終プロセス群 (これを F と書く) を処理する資源群 $\mathcal{F}(F)$ 上での F のプロセスを含む最も遅い連続処理区間の終了時刻の、 q_P および R' 上でのプロセス群 $\mathcal{C}_i^*(R, R', q_P)$ 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 $\mathcal{F}(F)$ 上では、 $\mathcal{C}_i^*(R, R', q_P)$ とその後続プロセス群以外のプロセスは最早開始時刻で処理可能になると仮定する。 $T_1(t, R, P)$ の近似 $T_2(t, R, P)$ は(4)式によって求める。

$$T_2(t, R, P) = \max_{R' \in S(\tilde{\mathcal{S}}_i(R)) \cup R} e_2(t, R', P) \quad \dots\dots\dots (4)$$

□

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限定され、必要な計算量は更に低下する。また $e_2(t, R', P)$ を算出すべき資源の範囲も〔近似1〕より限定される。更に(5)式が成立するのは自明であり、 T_2 が T_1' より良い近似である事実は変わらない。

$$T_1'(t, R, P) \leq T_2(t, R, P) \leq T_1(t, R, P) \leq T_1(t, R, P) \dots\dots\dots (5)$$

$e_2(t, R', P)$ を計算するには、 P を先頭に固定したプロセス群 $\tilde{\mathcal{S}}_i(R)$ の処理順序 q_P のすべてに対して $\mathcal{C}_i^*(R, R', q_P)$ を求め、更にプロセス集合 $\mathcal{C}_i^*(R, R', q_P)$ のすべ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、各資源 R' が仮定する R 上でのプロセス群 $\tilde{\mathcal{S}}_i(R)$ の処理順序を、資源 R が $e_2(t, R, P)$ を計算するときに仮定する順序に固定する。また集合 $\mathcal{C}_i^*(R, R', q_P)$ 内のプロセス数を n とすると、それらのプロセス処理順序の可能性の数は $n!$ 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 のオーダーに減らす。

〔近似3〕 まず、後述のようにして定まる P が先頭になるプロセス群 $\tilde{\mathcal{S}}_i(R)$ の処理順序 q_P^* を固定する。そのうえで資源 R' は、プロセス群 $\mathcal{C}_i^*(R, R', q_P^*)$ の処理順序を以下のようにして求める。

(1) $\mathcal{C}_i^*(R, R', q_P^*)$ 中の処理順序の決まっているプロセスの集合を X 、処理順序の決まっている集合を 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遅らせる。更に Y 、 $\tilde{\mathcal{S}}_i(R)$ 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 q_P^* に従って遅らせて、他のプロセス群は最早開始時刻に処理可能になると仮定する。これらの仮定のもとで、 R' が X に属する各プロ

セス Q に対して、 $\mathcal{V}_i^*(R, R', g_p^*)$ の最終プロセス集合 F を処理する資源群上での F のプロセス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 $\tau(Q, X)$ 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。更に X が空になるまで(1)項からの処理を繰返す。

(1)~(3)項の手順によって $\mathcal{V}_i^*(R, R', g_p^*)$ のプロセス処理順序が求まったとき、 $\mathcal{F}(\mathcal{V}_i^*(R, R', g_p^*))$ 上で $\mathcal{P}_i(R)$ および $\mathcal{V}_i^*(R, R', g_p^*)$ の最終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)式における $e_2(t, R', P)$ の近似とする。ジョブ終了時刻最小化ルールが最終的に使う $T_i(t, R, P)$ の近似は次のようにして求まる。

$$T_A(t, R, P) = \max_{R' \in (\tilde{\mathcal{P}}_i(R)) \cup R} e_3(t, R', P) \quad \dots\dots\dots (6)$$

ここで、 g_p^* は $\mathcal{V}_i^*(R, R', g_p^*)$ を $\tilde{\mathcal{P}}_i(R)$ で置き換えた集合に対して(1)~(3)項の手順を施して得られる、 $\tilde{\mathcal{P}}_i(R)$ のプロセス処理順序として求まる。

□

〔近似3〕ではプロセス集合 X の処理順序は、 $Q \in X$ の処理順序を最後にして、 X 中の Q 以外のすべてのプロセスが即座に処理できると仮定し、そのときのジョブ終了時刻を最小にするプロセス Q^* を X 中の最後に順序付けるという操作の繰返しによって求まる。従って、 X に属するプロセス数を n とすると、 $n^2/2$ に比例する計算量で処理順序が求まる。しかし、この方法で求めた $T_A(t, R, P)$ は $T_i(t, R, P)$ よりも大きくなる可能性があり、もはや(2)、(5)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 本節では、(6)式の近似を用いたスケジューリング実行手順を示す。本手順はシステム内の個々の資源上で自律的かつ並列的に実行され、個々の資源の処理の同期をとったり処理結果を調整する集中機構の存在は仮定しない。従って、システム運転中にジョブ投入計画やシステム構成要素に変更があっても、個々の資源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。各資源は手順の中で必要となる情報を、以下に示すテーブルの形で記憶する。スケジューリングは各資源がこれらの情報を互いに交換することによって進行する。

〔処理予定プロセス・テーブル〕 資源に割当てられたプロセス群に関する情報を、プロセスごとに {プロセス名, 所属ジョブ名, 最早処理区間} からなるレコードとして記憶する。ここでプロセス P と Q が

同一の所属ジョブ名をもつのは、 Q が P の先行あるいは後続プロセスであるときのみである。

〔最終プロセス処理資源テーブル〕 資源に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記憶し、最終プロセスを処理する資源ごとに {資源名, 後続処理区間, 一般処理区間, 変更フラグ} からなるレコードが定義される。ここでテーブルを所有している資源を R 、テーブル中の最終プロセス処理資源の一つを 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もつ。すなわち、 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 に属するとき、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 に所属するプロセスが複数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 に属さないプロセス群が構成する連続処理区間の集合である。また変更フラグは、処理の進行や新たなジョブ投入により、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。後続、一般処理区間の内容変更手順については後述する。

以上の情報を用いて、ジョブ終了時刻最小化ルールによるスケジューリングは次のように進む。

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

(2) 資源 R は次に、 $\tilde{\mathcal{P}}_i(R)$ に属する各プロセス P に対して、 P を最初に処理した場合に資源群 $\mathcal{F}(\tilde{\mathcal{P}}_i(R))$ 上の $\mathcal{P}_i(R)$ の後続プロセスを含む最も遅い連続処理区間終了時刻を最小化するプロセス群 $\tilde{\mathcal{P}}_i(R)$ の処理順序 g_p^* と、そのときの終了時刻 $e_3(t, R, P)$ 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 $\tilde{\mathcal{P}}_i(R)$ 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 $\{P, g_p^* | P \in \tilde{\mathcal{P}}_i(R)\}$ と $\tilde{\mathcal{P}}_i(R)$ の各プロセス処理時間を放送する。

(3) 資源 R' は $\{P, g_p^* | P \in \tilde{\mathcal{P}}_i(R)\}$ を受け取ると、自身が $\tilde{\mathcal{P}}_i(R)$ の後続プロセスを処理する場合のみ、各 P に対して $\mathcal{V}_i^*(R, R', g_p^*)$ を求め、(2)項の手順と同じようにして、資源群 $\mathcal{F}(\mathcal{V}_i^*(R, R', g_p^*))$

上での処理終了時刻の最小値 $es(t, R', P)$ を計算する。計算結果は $\{P, es(t, R', P) \mid P \in \tilde{\mathcal{P}}_i(R)\}$ として R に返送する。

(4) R は R' から返答を受け取ると、 R' (R も含む) に関する $es(t, R', P)$ 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。更に R は、 $\tilde{\mathcal{P}}_i(R)$ 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) $\tilde{\mathcal{P}}_i(R)$ の後続プロセスを処理する R' は、 R から P^* の選択通知を受け取ると、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、更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。テーブルの内容に変更があると、 R' は更に変更通知に P^* と R' を添えて他の資源群に放送する。

(6) 変更通知と P^*, R' を受け取った R'' は、既にステップ(5)で P^* に関するテーブル修正を済ませている場合は、変更通知を無視する。そうでない場合は、最終プロセス処理資源・テーブルの資源名に R' をも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・テーブルの変更手続きを示す。各資源 R はステップ(2)あるいは(3)で最終プロセス処理資源・テーブルの資源 R' に関するレコードを参照する場合は必ずレコードの変更フラグを確認する。変更フラグが立っていないときはレコードの内容をそのまま利用するが、変更フラグが立っている場合は、 R' から処理予定プロセス・テーブルの内容を受け取って、以下のようにレコードを修正する。つまり、 R' から受け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。このとき残ったプロセス群の最早処理区間を接続したものが、一般処理区間で除去したプロセス群の最早処理区間が後続処理区間を構成する。

ただし、 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し、それを受け取った資源 R は、自身に $es(t, R, P)$ を計算する必要があるときのみそれを修正し、それ以外の場合は変更フラグを立てる。このことによって R' は一つの資源上で発生したスケジューリング操作に伴う複数の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回行うだけで答えることができる。また、テーブル内容を受け取る資源 R も、必要なときのみテーブル更新を行うのでむだな計算が省略できる。

4. スケジューリング法の評価

本章では、ジョブ終了時刻最小化ルールによるスケ

ジューリング法を実行に必要な通信量、記憶量、処理量、生成スケジュール効率の面から評価する。以下、システム中の資源の総数を N 、一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 $\tilde{\mathcal{P}}_i(R)$ の後続プロセス処理資源群 $S(\tilde{\mathcal{P}}_i(R)) = \mathcal{R}$ に $es(t, R', P)$ の計算を依頼し、各資源群から結果を受け取る。このとき、計算依頼には放送通信が1回、結果の受け取りには $\mathcal{R}$ 中の資源数だけの通信が必要になる。各資源が単位時間にスケジュールを実行する回数は割当てプロセス数 M に比例し、資源数が N 以下であるので、手順実行による単位時間あたり通信回数は $N^2 \cdot M$ のオーダーを超えることはない。次に、資源上の最終プロセス処理資源・テーブルの内容変更に関する通信量を考える。テーブル修正内容の関連資源への要求は、放送により1回の通信で済む。単位時間に一つの資源が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ルを実行する回数より少なく $M \cdot N$ のオーダー以下であるから、システム全体でテーブル修正要求に必要な単位時間あたり通信回数は $N^2 \cdot M$ のオーダーを超えない。テーブル修正内容の転送については、1回の要求に対して最大 N 個の資源が情報転送を行うが、ある資源上のスケジューリング操作に派生した複数の関連資源が発する修正内容要求には、各資源は1回の放送通信で答える。すべての資源がスケジュールを実行する回数が $N \cdot M$ に比例することを考えると、修正内容の転送に必要な通信回数も結局 $N^2 \cdot M$ のオーダーを超えない。システム中の全資源が必要とするテーブル容量は、容易にわかるように $N^2 \cdot M$ のオーダー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 $es(t, R, P)$ の算出に要するものである。これには、最終プロセス処理資源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量自体は M のオーダーであり、結局、各資源はスケジューリング要求があると M^3 オーダの演算をすることになる。(6)式の T を求めるには、処理候補プロセス集合中の各プロセスの処理順序を先頭に固定して手順(2)と(3)を実施する必要があるので、 q を処理候補プロセス集合の要素の数とすると、 M^3 オーダの計算を q 回する必要があるが、 q は M に比べると小さい。

次にジョブ終了時刻最小化ルールのスケジューリング性能を、シミュレーションによって評価する。表1にシミュレーション条件を示す。システムは六つの資源 $R_1 \sim R_6$ からなり、そこへ時刻0にジョブ群 $J_1 \sim J_6$

表 1 システムへの投入ジョブ

Table 1. Jobs entering to a system.

ケース(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
J_1	R_1 (7)	R_3 (5)	R_5 (8)	
J_2	R_1 (2)	R_4 (1)	R_5 (2)	R_6 (2)
J_3	R_1 (2)	R_2 (2)	R_5 (2)	R_4 (1)
J_4	R_2 (7)	R_4 (5)		
J_5	R_2 (4)	R_4 (2)	R_5 (4)	
J_6	R_2 (3)	R_3 (2)	R_4 (2)	R_6 (4)

ケース(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
J_7	R_1 (3)	R_3 (2)	R_4 (9)	
J_8	R_1 (1)	R_2 (2)	R_5 (2)	
J_9	R_2 (1)	R_3 (2)	R_4 (2)	
J_{10}	R_2 (2)	R_3 (4)	R_5 (2)	

が、また時刻5にジョブ群 $J_7 \sim J_{10}$ が投入される。表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に投入されるジョブ J_7 は三つのプロセスから構成され、最初のプロセスは R_1 で処理され、処理には3単位の時間が必要である。第2、第3のプロセスは各々 R_3, R_4 で処理され、処理にはそれぞれ2単位、9単位の時間が必要である。

図2、図3にスケジューリング結果を示す。図2は時刻0に $J_1 \sim J_6$ だけが投入された場合に、また図3は、時刻5に更に $J_7 \sim J_{10}$ が投入された場合に対応する。図は $R_1 \sim R_6$ が処理するプロセスの時間変化を示しており、図中の $P_{ij}(j)$ はジョブ i の j 番目のプロセスを処理していることを表す。また白抜き、ハッチング、塗りつぶしの箱は、各々 SPT, MWKR, ジョブ終了時刻最小化ルールを用いた結果である。図2では、ジョブ終了時刻最小化ルールですべての処理が終了する時刻が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3では、ジョブ終了時刻最小化ルールでは時刻25ですべての処理が終了するのに対して、SPT, MWKR ルールでは時刻30, 31にそれぞれ終了する。いずれの場合もジョブ終了時刻最小化ルールのほうが良い成績を上げていることがわかる。図3の場合のほうが成績の差は、絶対値としても相対値としても大きい、これはプロセスの数の増加と共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、ジョブ終了時刻最小化ルールはすべ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3では R_3 上での処理

図2 スケジューリング例 ケース(a)

Fig. 2. A scheduling example case (a).

図 3 スケジューリング例 ケース (b)
Fig. 3. A scheduling example case (b).

は SPT ルールによるほうが、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ときにジョブの終了時刻が最小になることを考えると、ほとんどの場合に、ジョブ終了時刻最小化ルールが、SPT や MWKR ルールよりも良いスケジュールを生成することが期待できる。

5. あとがき

知的分散システムで求められる分散、動的スケジューリング法として、ジョブ終了時刻最小化ルールを提案した。本方法により、個々の資源が必要な情報を他の資源群から集めて、自身に割当てられたプロセスの処理順序を自律的に決定する分散スケジューリングが可能になる。また必要な計算量は分枝限定法などに比べるとはるかに少なく、スケジューリング性能も従来のディスパッチングルールによる方法よりも向上する。

今後、個々のプロセスに対してそれを処理可能な複数の代替資源がある場合や、ジョブに優先度や納期制約のある場合、あるいは資源のパバッファ容量に関する制約が強い場合、ジョブを構成するプロセスの列に分歧や合流のある場合なども扱う方法に拡張したい。

代替資源のある場合への拡張に関しては次のような

れると、それを構成するプロセス群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な資源に割当てる。負荷を平準化する資源は、ビディング (bidding) 法あるいは両方向ビディング法⁽²⁾を用いて、集中機構を介さずに決定することができる。

なお、〈3・3〉節に示した手順では、スケジューリングの処理が個々の資源上で並列して行われるが、これらの処理間での整合性を保つ方法については述べなかった。従って、例えばある資源の状態に関する情報を受け取って利用している間に、その資源の状態が変化し処理全体として矛盾の起こる可能性がある。しかしこれらの矛盾も、時刻印法⁽⁷⁾などを用いることにより、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。

(昭和 63 年 6 月 13 日受付、同 63 年 12 月 7 日再受付)

文 献

- (1) 「分散と協調特集」, 計測と制御, 26 (昭 62)
- (2) 田村, 他: 電学論 C, 108, 393 (昭 63)
- (3) 岩田, 他: 機械学論 C, 47, 709 (昭 56)
- (4) 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, 31, 29 (昭 61-1)
- (5) 室津, 他: 機械学論 C, 47, 803 (昭 56)
- (6) J.H. Blackstone, et al.: Int. J. Prod. Res., 20, 27 (1982)
- (7) 上林: 情報処理, 24, 1020 (昭 58-8)

ことが考えられる。つまり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処理すべきプロセスを選択する過程 (選択過程) とに分離し、選択過程ではジョブ終了最小化ルールをそのまま使う。また割当て過程では、既に割当て済みのプロセスの処理順序を〈3・3〉節の手順で $T_A(t, R, P)$ を算出したときの順序に仮定する。新たにジョブが投入さ