協調分散制御システムアーキテクチャ：CODA の概念モデル

神 余 浩 夫*・竹 垣 盛 一*

The Concept of the Coordinative Decentralized Control System Architecture : CODA

Hiroo KANAMARU* and Morikazu TAKEGAKI*

In this paper, we propose the COordinative Decentralized control system Architecture: CODA, as a fault-tolerant industrial plant control system. The system can be reconfigured to adapt to its conditions and environment by the coordination of its controllers. It consists of symmetically coupling autonomous controllers, and has neither supervisory controller nor reserved one. These controllers communicate with each other and decide their own functions. In order to make the communicating time short enough to keep a real-time constraints, they communicate and cooperate locally. Hence, the global system must repeat these local cooperations till getting to a coordinating state; we call this behavior the coordinative dynamics. For example, when a controller is broken, normal controllers backup its functions locally and gracefully degradate under overloaded condition. And this changing is influences to far controllers gradually. To guarantee such a coordination, we designed Coordination Protocol as the rules of cooperations among the controllers. This Protocol consists of three function layers; the Primitive Communication Layers, the Distributive Scheduling Layer and the Autonomous Scheduling Layer. Using the Coordination Protocol, it is easy to apply other control systems. We analyzed the ability of this Protocol, and made its prototype to confirm the ability of this framework.

Key Words: decentralized system, distributed control, autonomous, adaptability, coordination

1. 序 論

近年、大規模かつ複雑化する産業用制御システムに

対して、信頼性と耐故障性の向上が要求されている。

この要求に従い、制御システムのアーキテクチャは全体を一括管理する集中型から「自らの規範に従って行動する部分が他の部分との協調によって自らの目的を達成するとともに、全体の目的を達成する」自律分散システムを指向している^{1),2)}。これまでの自律分散システムは部分が全体に影響を与えない点に注目していたが、故障などにより失われた機能を残されたシステムで維持するためには部分相互の積極的な協調動作が重要となる。このような部分の協調動作により機能と構造を動的再構成してシステムの状態や環境変化に自己組織的に適応するシステムを筆者らは協調分散システムと呼んでいる³⁾。

協調分散アーキテクチャ CODA (COordinative Decentralized control Architecture) は協調分散システム概念の分散制御分野での具体化である。管理系や待機系が存在せず、通信ネットワークに対等かつ平等に接続された自律的な制御装置群から構成される。それぞれの制御装置は毎制御周期に機能分散と機能選択を行って、その状況下で適切な制御機能を動的に構成する。ここで、それぞれの制御装置が多数の装置と通信し協調すると組合せ爆発が起り、制御のリアルタイム性に悪影響が現れるので、CODA の各制御装置は隣接する装置とのみの通信を同時並行して行う。この局所的な協調方式では、部分の相互作用が全体に波及してシステムが適応状態で安定するまで、再構成を何度か繰り返すことになる。すなわち、協調分散システムは状態変化への適応に際してダイナミクスを有するので、この協調ダイナミクスの収束安定性を保証する部分協調機構が必要である。

CODA はシステムの健全な協調動作を保証する協調機構を各制御要素が遵守すべき協調プロトコルとして実現する。協調機構は要素間の相互作用に関する機能と要素の自律的な動作とに分けられて、階層的に実現される。すなわち、協調プロトコルは基本通信機能

* 三菱電機(株)中央研究所 尼崎市塚口本町 8-1-1

* Central Research Laboratory, Mitsubishi Electric Corp., Amagasaki

(Received September 3, 1990)

(Revised December 25, 1990)

の通信プリミティブ層、隣接装置との機能分散を行う分散スケジューリング層および各装置の実行機能選択を行う自律スケジューリング層の階層から構成される。複雑な協調動作を協調プロトコルとしてシステムレベルで実現するため、プラント制御機能群は協調動作を意識しないアプリケーションとして協調分散制御システム上で実行管理される。したがって、協調プロトコルを利用することにより協調分散の応用システムの構築が容易になる。

本論文は協調分散システムの概念と制御システムとしての協調の目的について論じ、その協調アーキテクチャを提案する。またリスク分散と優美縮退を行う協調プロトコルの構成と概要を説明し、CODA の動作シミュレーション結果について報告する。

協調プロトコルの詳細など CODA の実装に関する具体的な成果は別の機会に発表したい。

2. 協調の概念と目的

2.1 自律分散と協調分散

耐故障性と拡張性に優れた分散制御システムとして、分散された制御装置の自律性を追求した自律分散システムや知的分散システム⁴⁾が提案されている。たとえば、自律分散システムは自律可制御性と自律可協調性の基本概念により、システム要素の故障でシステムが停止しないことを目的としている。しかし、故障により失われた機能を維持する方法についてはあまり議論されていない⁵⁾。冗長系をもたない自律分散システムにおいて喪失機能を回復維持するためには、故障装置の機能を健全な装置間で協調的に分担するので、装置間で密な通信と相互作用が必要となる。このような積極的な相互作用に基づく協調性は従来の自律分散システムの隔離的な自律性だけでは達成できない。すなわち、協調分散システムは自律的な各制御装置が相互に通信し協調動作を行うことにより制御システムとしての性能維持を図ることを特徴とした分散システムである。

2.2 動的再構成による適応

制御システムは制御対象の応答を所定の目標に向けて調整するが、この性能を多様な状況下で発揮できるようにするには自らの特性を状況に適応させねばならない。適応制御やフェジィ自動調整など制御システムが知的・自律的であるほど高度な適応性が要求される。耐故障性も故障という状況下で機能を維持するために制御システムの機能と構造の再構成を行うという意味で適応性とみなすことができる⁶⁾。

従来のフォールトトレラントは故障時に備えて待機

冗長系を用意する方法であったが、主系・待機系とも故障するとシステム全体としての余力があっても機能喪失は免れなかった。協調分散システムは制御装置の協調動作によりそのときの状況に適した機能構成を動的に行いシステムの全能力を最大限活用する。さらに処理能力を上回る機能を要求されたときは協調によって重要機能を優先的に維持する構成を行う優美縮退 (graceful degradation)⁷⁾を行う。たとえば、プラント制御において主反応炉制御は最も重要かつ常時維持されるべき機能であるが、予備タンクなどの周辺装置監視は数分に一度行っても安全上ほとんど問題がない。もし制御装置の多重故障などで過負荷状態となった場合、前者を毎制御周期、後者を数回に一度に縮退して実行して制御システムの致命的機能喪失を回避する。CODA はこのような高度な協調動作をシステム機能として実現する。

2.3 部分協調と全体協調

協調分散システムは中央集権の装置をもたないので、協調に必要な通信と相互作用は個々の要素の自律的な判断により行われる。しかし、各装置が他の全装置と通信することは、制御装置数の増加とともに通信量と処理量が爆発的に増加し、制御システムのリアルタイム性に悪影響を与える。そこで、協調分散アーキテクチャ CODA は隣接する局所的な範囲での通信と協調動作を同時並行して行い、全体の協調を図る。具体例を Fig. 1 に示すと、ある制御装置が故障するとその装置の機能群 (タスク) はまず隣接装置にてバックアップされ、つぎの協調動作で負荷増加した隣接装置のタスクをさらに周辺隣接装置へと分散していく。この部分協調の繰返しにより局所的な変化が全体に波及し、ついにはシステム全体が状況に適応した状態に

Fig. 1 The Dynamics of loads balancing

到達する。この協調ダイナミクスを安定とする部分協調機構が協調分散システムには必要である。

部分は全体の協調の状況を知らずに協調動作を行うので、部分協調の目的が場所によって矛盾したり、全体の協調目的と一致していなかったりすると全体の協調化は達成できない。CODA は重要機能の維持のために負荷均等分散と優先タスクの実行による優美縮退を示すが、この部分協調は隣接装置間の負荷分散スケジューリングと制御装置単体のリアルタイム制約下での優先タスクの実行である。すなわち、部分の協調とシステム全体の目的がリアルタイム充足とリスク分散の点で一致しているため、部分協調に基づく全体の協調化が可能である。

3. 協調分散システムのアーキテクチャ

3.1 システム構成

協調分散制御システムは集中管理装置や待機系をもたない完全な水平分散型システムである。CODA のシステム構成例を Fig. 2 に示す。動的再構成によって機能を制御装置間で移動させるので、各制御装置はすべての機能を処理できる必要がある。このため、対象プラントのセンシング情報をリモート I/O 装置からデータベースにブロードキャストして、制御装置はすべてのプロセスデータを毎制御周期に得ることができる。また、各制御装置は近接するいくつかの装置のプラント制御タスクのソフトウェアを互いに内蔵し、同じタスクを複数装置が実行可能である。すべてのタスクが常時スタンバイ状態にあり、毎制御周期の自律スケジューリングに従い実行される。各制御装置は隣接装置と協調バス (coordination bus) を介して協調動作を行う。隣接装置の故障が検出された場合は一つ向こう側の装置を隣接装置として機能再構成を行う。運転

Fig. 2 An example of the CODA

Fig. 3 The layers of the coordination protocol

員の制御盤も協調バスに接続されるが、協調動作には関与しない。データベースと協調バスは制御システムとしてのリアルタイム性が得られるならば共用できる。

3.2 ソフトウェアアーキテクチャ

分散された制御要素が自律的に動作して全体をうまく協調させるためには、部分と全体の目的一致だけでなく通信と協調動作に関する規則をそれぞれが遵守する必要がある。この規則を協調プロトコルといい、Fig. 3 の階層構成をとる⁸⁾。リアルタイム OS と通信ファームウェアの上に基本的な通信機能群である通信プリミティブ層、つぎに隣接装置との機能分散を行う分散スケジューリング層そして単体での優先タスク選択の自律スケジューリング層がある。さらに自律スケジューリング層の上にプラント制御機能の本体である制御タスク群が存在する。アプリケーションタスクは実行時間が通信時間に比して大きく、また他装置に実行を切り換える際に移動しなければならないトラッキングデータなどのタスク内データ量が小さいとする。このアプリケーションタスクにとっては、協調プロトコルの機能は CODA のシステム機能とみなすことができる。要求される協調性が同一であれば、協調プロトコルの上のアプリケーションタスクのみを作成するだけですむので、新しい協調分散システムの構築が容易になる利点がある。

分散環境におけるタスクスケジューリング問題をプロセッサ間分配と単体内での順序決定により解こうとする試みはあったが、両手法を融合するための現実的な最適分散スケジューリング手法などに問題が残されていた⁹⁾。CODA の部分協調で全体を律する基本概念は要素が行うべき相互作用を軽くしているので、複雑な協調機構が単純かつ明確な階層の協調プロトコルにより実現されている。協調プロトコルの各層の機能についてはつぎの章で詳しく説明する。

4. 協調プロトコル

4.1 通信プリミティブ

協調プロトコル最下層の通信プリミティブ層の目的は通信情報バスの低レベルプロトコルの差異を吸収

し、上位の分散・自律スケジューリング層に必要な通信機能を提供することである。CODA の構成は対称な水平分散型であるが、通信もすべての制御装置が同一のアルゴリズムを用いる対称性を行うことを特徴とする。したがって、通信モデルはマスタ・スレーブ型あるいはサーバ・クライアント型でなく双方が対等にメッセージ交換する通信相手を指定して短いメッセージを送受信するデータグラム型通信を用いる¹⁰⁾。非同期で高速応答性が得られること、通信相手とコネクションを生成する必要がないため動的再構成が容易であることおよび比較的単純なプロトコルであり移植性が高いなどの理由による。

CODA の制御装置は相互に故障監視して局所的なバックアップを行う。この故障検出はタイムアウト検出法、すなわち隣接装置からの送信が途絶えると相手の故障と判定する方式を採用する。通信を行う両者の時間ずれが大きくなると正常であっても故障と判定されるので、本判定法は両者間に同期が存在することを前提としている。CODA の各制御装置はプラントとの I/O 動作のために制御周期以内に必ず処理を終了させるため、この I/O 動作一貫性が維持される範囲の緩い同期が存在する。したがって、非同期型のデータグラム通信にもかかわらず本判定法が利用できる。故障が検出されるとその装置をネットワークから切り離し、その装置が担当していたタスクを隣接装置に代替させる。その後、故障装置の両隣接装置は互いを新しい隣接装置として認識し通信を始める。通信プリミティブ層は故障検出から隣接装置の再構成までの機能を含んでいるので、上位のスケジューリング層は制御装置の故障を意識する必要がない。

4.2 分散スケジューリング

分散スケジューリング層は通信プリミティブ層の機能を用いて隣接装置と相互通信を行い、両者間の負荷均等化のための装置間タスク分配を行う。分散スケジューリングのアルゴリズムとそのペトリネット表現を Fig. 4 に示す。各制御装置は現在有するタスク実行権を二つのグループに分け、一つを右側に送る。つぎに残ったグループと左側から受けた実行権を負荷分散となるように分配を決め、左側担当分を左側に送る。自分が担当するのは自分で決めた実行権と右側から送られた実行権の示すタスクである。CODA の制御装置はすべてが同じアルゴリズムにより同時並行的に処理を進めながら協調動作を行うので、マスタ・スレーブなど異なるソフトウェアを複数用意する必要がない。このため、Fig. 4 の点線で囲まれた故障装置を切り離しその両側装置を接続するだけで、分散スケ

Fig. 4 The Petri-net representation of the distributive scheduling algorithm

ューリング層までのシステム再構成が可能である。

動的再構成によりタスクを実行する制御装置が入れ替わるが、タスクを実行する装置が複数存在したりどの装置も担当しない状況となってはならない。そこでタスク実行権を用いて分散システム上にタスクが一意であることを管理する。各周期にタスク実行権をもつ装置がそのタスクを実行する。装置間のタスクの移動とは実行権の移動にほかならない。タスクの移動に必要なタスク内データは毎回の分散スケジューリングの際に通信する。

毎回の分散スケジューリングにより隣接装置は担当タスクを数え合っているため、故障装置の両隣接装置が失われたタスクを知っている。そこで両装置が故障検出後失われたタスクの実行権を復活させる。

CODA のシステム適応化は隣接装置間の負荷バランスのためのタスク割当を毎制御周期に繰り返して、全体が負荷均等状態で安定することにより行われる。分散スケジューリングは負荷分散によるタスク分配の安定性を保証しているが、この詳細は次章で述べる。

4.3 自律スケジューリング

制御装置の処理余裕が少ないときに故障した隣接装置のタスクを担当しても、すべてをつぎのプロセス I/O 時刻 (deadline) までに処理できない場合がある。このときには優先度の低いタスクを一時的に休止させて重要なタスクを確実に実行する優美縮退を行う。自律スケジューリングは上記の分散スケジューリングにより割り当てられたタスクの中からその制御周期に実行するタスクを決定する機能である。一般に制御タスクは設計時に処理時間が仕様としてわかっているから、自律スケジューリングは周期ごとの静的スケジューリングと同義である。従来のスケジューリング問題

の多くはリアルタイム性を満足するためのタスク実行順序を主題としたが¹¹⁾、自律スケジューリングは順序決定まで行わないため高速である。自律スケジューリングで休止したタスクがあっても、数周期でその制御装置の負荷が分散されるので、すべてのタスクが実行可能となる。協調プロトコルは協調動作をシステム構成要素間動作と要素内動作に分離して、それぞれを分散スケジューリングと自律スケジューリングとして実現している。

応用システムによっては協調動作数回に一度は必ず実行しなければならないタスクが存在したり、低優先度で休止したタスクを適当に実行したりする必要がある。このような場合は優先度を時間的に変化させる自律スケジューリングの拡張が要求されるが、その具体的な機構は応用システムの目的とタスク内容に強く依存する。現時点のタスク優先度管理方式は初期設定値のまま不変の固定方式としているが、将来は可変方式に拡張する予定である。

5. 協調分散システムの特長

5.1 協調動作の安定性

分散制御装置の故障は制御系の構造次数の減少であるから、協調システムはその状態に適応するように機能再構成を行う。このとき、適応化のための協調動作が不安定であると、ある装置に負荷が集中する発散や複数の状態で振動現象に陥る。したがって、安定な協調ダイナミクスを形成する協調機構を設計しなければならない。以下、分散スケジューリングによる負荷分散をダイナミカルシステムモデルに表わし、その安定性に関して論じる。

制御周期 t 回目における制御装置 i, j の負荷（タスク処理時間）をそれぞれ x_{it}, x_{jt} として、その周期の分散スケジューリングによる負荷移動量を $a(x_{it} - x_{jt})$ とする。ここで、 a は設計時に与えられる定数である。毎周期にすべての隣接装置間で分散スケジューリングが並行して行われるので、制御装置 n 台の負荷 $(x_1, x_2, \dots, x_n)$ は次式で表わされる。

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{t+1} = \begin{bmatrix} -a & a & 0 \\ a & -a & \\ & 0 & \ddots \\ 0 & & 0 \end{bmatrix} \begin{bmatrix} 0 & -a & a & 0 \\ a & -a & & \\ & 0 & & 0 \\ 0 & & & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_t + \begin{bmatrix} a & a \\ 0 & 0 \\ 0 & 0 \\ a & 0 & -a \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_t + \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_t$$

$$= a \begin{bmatrix} -2 & 1 & & 1 \\ 1 & -2 & 1 & \\ & \ddots & \ddots & \ddots \\ 0 & & 1 & -2 & 1 \\ 1 & & & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_t + \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_t$$

$$\therefore X_{t+1} = aA_n X_t + X_t = [aA_n + I] X_t$$

ただし、 $A_n(n \times n)$, $X(n)$ であり、 I は単位行列である。いま、 $[aA_n + I]$ の特性方程式を考える。

$$F_n = \det[\lambda I - [aA_n + I]]$$

A_n の固有値が、 $2 \cos(k\pi/n) - 2$, $k=0, 1, \dots, n-1$ であることから¹⁴⁾、特性方程式の根は次式となる。

$$\lambda = 2a(\cos(k\pi/n) - 1) + 1, \quad k=0, 1, \dots, n-1$$

ただし、 $k=0$ は $\sum x_i$; ($i=1, n$) = const に対応する固有値である。有限回の分散スケジューリングで X_t がある状態に収束し安定となるのは、 $[aA_n + I]$ の $k=0$ 以外の固有値が単位円内に存在するとき ($|\lambda| < 1$) であるから、 $(0 < a < 1/2)$ のときである。すなわち、システム全体が安定収束するためには、隣接装置間の負荷移動量は両者の負荷の差の半分以下としなければならない。

分散スケジューリングはタスク単位で行われるため、負荷移動量は常に $a(x_{it} - x_{jt})$ とはならず、移動するタスクに応じて異なる値となる。上のモデルでは隣接装置間に偏差があれば負荷移動が行われようとするが、移動するタスクが一つだけでそのタスクの負荷が両者の偏差より大きければ安定条件を満足しない。そこで、CODA の分散スケジューリングはこの場合にタスクの移動を行わない。この境界となる負荷の値を分散スケジューリングのしきい値という。このような負荷移動量の不連続性など実際の分散スケジューリングとダイナミカルモデルの間に違いはあるが、この条件は十分効果的な適応安定性の判断の目安として利用できる。

5.2 同期問題

分散システムの装置間の時間ずれが大きくなると、装置間の相互通信やプロセス I/O 動作において同時刻性が崩れてデータの一貫性が保たれなくなる。そこで、装置間の時間ずれを監視・補正する同期機構が必要になる。協調分散システムはアーキテクチャ上、強力な集中型の同期機構をもてないので隣接装置間の通信で同期をとりながら全体の時間ずれを抑制しなければならない。そこで、協調プロトコルの局所的な同期機構の性能を評価する¹²⁾。

Fig. 5 は Fig. 4 の分散スケジューリングを時間軸上に表わしたチャート図であり、左右制御装置に起こる送受信の事象の時刻をそれぞれ、 $T_{L1}, T_{L2}, \dots, T_{R1}$,

Fig. 5 The synchronous chart in the coordination protocol

$T_{R2}, \dots$ とする。同期は受信タイムアウト以内の受信待ちによって行われる。いま、左装置から右への通信 ($T_{L2} \rightarrow T_{R3}$) においてタイムアウトとならない時間ずれは次式の範囲である。ただし、装置間の通信時間を ΔT_{com} 、タイムアウト時間を ΔT_{out} 、送受信処理に要する時間を ΔT_s 、 ΔT_r とする。

$$\Delta T_r < T_{R3} - T_{R2} < \Delta T_{out}$$

$$T_{L1} + \Delta T_s + \Delta T_{com} < T_{R2} + \Delta T_{out}$$

ここで、 $T_{L1} + \Delta T_s = T_{L2}$ 、 $T_{R1} + \Delta T_s = T_{R2}$ であるから、

$$T_{L1} - T_{R1} = T_{L2} - T_{R2} < \Delta T_{out} - \Delta T_{com}$$

となる。スケジューリング結果の通信についても同様にして、

$$T_{L4} - T_{R4} = T_{L5} - T_{R5} > \Delta T_{com} - \Delta T_{out}$$

ここで、両者のスケジューリング時間が等しいとすると ($T_{L4} - T_{L3} = T_{R4} - T_{R3}$)、いずれの制御装置もタイムアウトとならないのは、

$$|T_{L1} - T_{R1}| < |\Delta T_{out} - \Delta T_{com}|$$

の場合である。一般に、通信時間よりタイムアウト時間のほうが長いので、隣接装置間で許される最大時間ずれは $\Delta T_{out} - \Delta T_{com}$ である。

CODA の同期機構は局所的であるから、全体では上式以上の時間ずれが存在する。たとえば、リング型ネットワークに n 個の装置が接続されているとすると、隣接の意味で最遠の装置とは $n/2$ 離れているからその最大時間ずれは、 $T_{max} = n/2(\Delta T_{out} - \Delta T_{com})$ となる。制御周期が T_{max} より小さいとプロセス I/O が時間ずれの間に 2 回行われるので、制御装置間のデータ一貫性が維持できなくなる。したがって、この式を満足するようにタイムアウト時間を設定しなければならない。

6. 実験例

6.1 プロトタイプ概要

以上の協調プロトコルのプロトタイプをワークステーションの通信機能上に作成し、優美縮退などの協調動作の検証を行った。使用したネットワークは IEEE 802.3 の Ethernet の TCP/IP 体系であり、その UDP (User Datagram Protocol) を用いて通信プリミティブ層を作成している¹³⁾。Ethernet は通信応答の上限時間を陽に保証できないので、リアルタイムシステムに本来適していない。プロトタイプでは送信回数を抑えネットワーク競合が起きにくくした上で応答時間を大きめに見積り (UDP の最高応答時間約 50 ms に対して 100 ms)、リアルタイム性が得られるようにした。

この協調プロトコルのスケジューリング結果に従い、制御タスクを処理するプロセスをそれぞれの計算機で実行する。ただし、プロトタイプではタスクは CPU を消費するだけのダミータスクとなっている。制御周期は 5 秒でタスク数は 16 個とし、それぞれのタスクに優先度と処理時間を初期設定で与えている。制御周期に対して通信時間が十分小さいので、今回自律スケジューリングでは協調動作のオーバーヘッド時間を無視している。

6.2 システム状態変化

CODA は制御システムの状況変化に対する機能維持のための協調適応化と、環境変化に対する機能適応化が可能である。はじめに、状態変化の例として制御装置の故障に対する適応化シミュレーションを Fig. 6 に示す。

初期状態 Fig. 6 (a) では四つの制御装置がほぼ均等にタスクを分担しているが、制御装置 C が故障するとタスクアウトによりその両隣の制御装置 B、D が故障を検知し、失われたタスクを復活させる。ただし、Fig. 6 (b) に示すように制御装置 B は過剰負荷になるので自律スケジューリングにより最も優先度の低いタスク 04 の実行が休止する。つぎの周期の分散スケジューリングで制御装置 A、B 間および A、D 間でタスクが Fig. 6 (c) のように移動し、さきに休止したタスクも実行できるようになる。しかし、まだ負荷バランス状態には達していない。さらにつぎの周期で三つの制御装置間の負荷バランスが達成し、Fig. 6 (d) のタスク配分で安定となる。すなわち、制御装置の故障に対して残存する制御装置を協調し、機能維持のための最適な秩序状態へと遷移したことになる。

いま、分散スケジューリングのタスク分配が起きる

	load ratio				0%	100%
A	00	01	02	03		
B	04	05	06	07		
C	08	09	10	11		
D	12	13	14	15		

(a) Initial state

A	00	01	02	03		
B	05	06	07	08	09	
C	04					
D	10	11	12	13	14	15

(b) Controller-C faults

A	15	00	01	02	03	04
B	05	06	07	08	09	
C						
D	10	11	12	13	14	

(c) Load balancing

A	00	01	02	03	04	
B	05	06	07	08	09	
C						
D	10	11	12	13	14	15

(d) Semi-stable state; shuttling a task

Fig. 6 The simulation of a fault of a controller

しきい値を 0.5 秒としたが、これを 0.2 秒にすると Fig. 6(d) で安定とならずに再び負荷分散が行われ、結果的に Fig. 6(c) と Fig. 6(d) のタスク分散状態を交互に繰り返す準安定状態にしなければならない。しきい値を小さくすることによりきめ細かい負荷分散が可能になるが、その反面でこのような準安定状態が発生する場合がある。ただし、CODA のタスクはどの制御装置で処理を行っても制御システム全体としての機能は同一であるから、準安定状態が支障となることはない。

6.3 システム環境変化

つぎに、CODA の環境変化に対する協調適応化の例として、制御対象プラントに異常が発生してタスクの処理時間が変化した場合のシミュレーションを Fig. 7 に示す。発生する異常はタスク 05, 06 に関係し、それらタスクの処理時間が 2 倍および 2.5 倍に増加すると想定する。協調プロトコルはこの時間増加を

	load ratio				0%	100%
A	00	01	02	03		
B	04	05	06	07		
C	08	09	10	11		
D	12	13	14	15		

(a) Initial state

A	00	01	02	03		
B	04	05	06	07		
C	08	09	10	11		
D	12	13	14	15		

(b) Change of loads as the plant abnormal

A	00	01	02	03	04	
B	05	06				
C	07	08	09	10	11	
D	12	13	14	15		

(c) Load balancing from controller-B

A	00	01	02	03	04	
B	05	06				
C	07	08	09	10		
D	11	12	13	14	15	

(d) Load balancing

Fig. 7 The simulation of an anomaly on the plant

異常診断のアプリケーションタスクから知らされる。Fig. 7(a)の初期状態では Fig. 6(a)と同様に四つの制御装置がほぼ均等にタスクを分担して処理を行っている。いま、プラントに故障が発生してタスク 05, 06 の処理時間が増加になると、それらを担当していた制御装置 B は過負荷となるので自律スケジューリングにより優先度の低いタスクが休止状態となる。この一時的な縮退のあと、分散スケジューリングによって制御装置 B のタスクは制御装置 A, C へと分散し、すべてのタスクが処理されるようになる。さらにつぎの制御周期で負荷バランスが進み、Fig. 7(d)の安定状態に到達する。制御装置の故障という制御システム状態変化の例と同様に、プラント異常によるシステム環境変化の場合でも、制御システムは一時的に縮退した後、新しい秩序で機能維持を行っていることが示されている。

7. 結 論

システム状態や環境変化に対し制御要素が協調的に動作することにより状況に適応化する協調分散制御システムアーキテクチャ CODA を提案し、その協調メカニズムを協調プロトコルとして実現した協調分散制御システムのプロトタイプを作成し、そのシステム全体の協調動作性とプロトコルの有効性を確認した。CODA の協調メカニズムは装置間でタスク分配を行う分散スケジューリングという隣接相互作用と自己の実行タスクを決定する自律スケジューリングという自律作用の組合せで構成され、この比較的簡単な部分協調のメカニズムによって全体の協調的適応化が達成される。本手法を実際の制御システムとして実現するためには、タスクの条件や協調機構の詳細について明確にすべき点が残されている。これらの実装に関する課題は現在研究中である。

最後に本研究中にご助言いただいた、三菱電機中央研究所の西口憲一氏に感謝いたします。

参 考 文 献

- 1) 石井, はか: 分散と協調特集, 計測と制御, **26**-1 (1987)
- 2) K. Mori, et al.: Autonomous Decentralized Software Structure and Its Application, Fall Joint Computer Conference, 1056/1063 (1986)
- 3) 神余, 竹垣: 協調分散制御システムの設計手法に関する考察, 第9回自律分散システム研究会講演論文集, 17/20 (1989)
- 4) 田村, 岡宅, 関: 知的分散システムのアーキテクチャ, 電気学会論文誌, **C-108**-6, 393/400 (1988)
- 5) 森, 宮本, 井原: 自律分散概念の提案, 電気学会論文誌, **C-104**-12, 303/310 (1984)
- 6) 竹垣, 石岡: 知的制御システム, p. 152, 海文堂 (1990)
- 7) B. W. Johnson: Fault-Tolerant Microprocessor-Based Systems, IEEE Micro, **4**-12, 6/21 (1984)
- 8) 神余, 竹垣: リアルタイム性を指向した協調分散プロトコル, 第1回自律分散システムシンポジウム論文集, 63/68 (1990)
- 9) J. A. Stankovic, et al.: Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems, IEEE Trans. Computers, **C-34**-12, 1130/1143 (1985)
- 10) A. S. Tanenbaum: Computer Networks 2nd. ed., p. 280, Prentice-Hall (1988)
- 11) C. L. Liu and J. W. Layland: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, J. ACM, **20**-1, 46/61 (1973)
- 12) 神余, 竹垣: 協調分散システムにおける同期機構, 第34回システム制御情報学会研究発表講演会論文集, 65/66 (1990)
- 13) D. Comer: Internetworking with TCP/IP, p. 120, Prentice-Hall (1988)
- 14) R. Bellman: Introduction to Matrix Analysis, p. 66, TATA McGraw Hill (1974)