社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－DPS－122（54）
2005－CSEC－28（54）
2005／3／23

システム縮退動作を可能とする分散ミドルウェアの設計

村 山 和 宏† 落 合 真 一†

近年、センサデータ処理システムなど、高いリアルタイム性能を必要とするシステムに、システムの一部に故障が発生しても性能を維持しつつ処理継続を実現することが求められている。このようなシステムでは、故障を素早く検出し、性能を高速に復旧することによって性能縮退時間を極力短くすることが重要である。我々は、大規模センサ信号処理システムに対し、システムの一部に故障が発生した場合には、重要度の低い計算資源を重要度の高い処理に割り当てることにより、重要度の高い処理のリアルタイム処理継続を可能とするミドルウェアを設計した。本ミドルウェアでは、システム構成情報伝達の効率化による短時間での CPU の移行、故障により発生する演算データの紛失の防止、通信 API の提供による高信頼アプリケーション作成の容易化を実現する。

Design of fault-tolerant distributed real-time processing middleware for signal processing systems

Kazuhiro Murayama† and Shinichi Ochiai†

Recently, real-time processing systems, such as sensor information processing systems, are required to keep high performance computation in case of system failure. To archive this demand, it is needed to reduce down time by finding system failure and recovering systems as quickly as possible. For this reason, we have been developing a fault-tolerant middleware which have three special features about recovery of computing power; (1) migrate CPUs from a low-important process to a high-important process, (2) resume by restoring past data, (3) provide APIs to make applications easy. In this paper, we describe the design of our middleware.

1. はじめに

近年、センサデータ処理システムなど、高いリアルタイム性能を必要とするシステムに、システムの一部に故障が発生しても性能を維持しつつ運転継続を実現することが求められている。このようなシステムでは、故障を素早く検出し、性能を高速に復旧することによって性能縮退時間を極力短くすることが重要である。

我々は、複数のセンサ信号処理アプリケーションで構成された大規模分散リアルタイムシステムに対して、システムの一部に故障が発生しても重要な処理の性能を復旧し、リアルタイム処理を優先して継続させることを可能とするミドルウェアの設計を行った。本稿では、ミドルウェアの設計内容について述べる。

2. 背　　景

2.1 ターゲットシステムの特徴

本研究のターゲットシステムの概要を**図 1** に示す。本システムは以下の特徴を持つ。

複数の CPU ボードによる分散処理システム： 本システムは、N 台の CPU ボードを搭載可能な Compact PCI(C-PCI) ユニットを十数台使用し、これらの間をネットワーク接続することにより構成される。

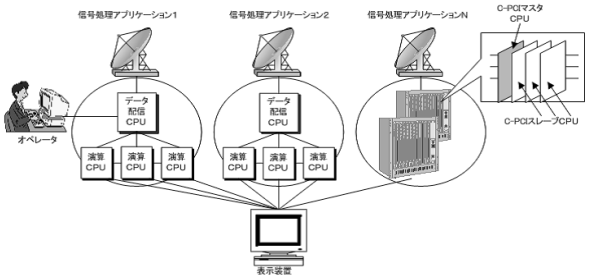

図 1 複数のアプリケーションによるシステム構成

複数の信号処理アプリケーションの組み合わせ： 本システムの処理は、独立した信号処理アプリケーションを複数組み合わせることにより実現している。アプリケーションでは、センサからの入力を「データ配信 CPU」が受信し、「演算 CPU」に配信する。演算 CPU は、センサデータの送信周期と演算時間に応じて 30 ～ 50 台程度使用される。

分散ハードリアルタイム処理の実現： システムを構成している各アプリケーションの処理パターンを**図 2** に示す。各アプリケーションでは、センサからのデータ送信周期が演算に要する時間よりも短く、複数の CPU で同一の処理を行うことにより全体の処理を実現している。演算結果は繰り返し使用しており、各 CPU が処理を行

† 三菱電機（株）情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation

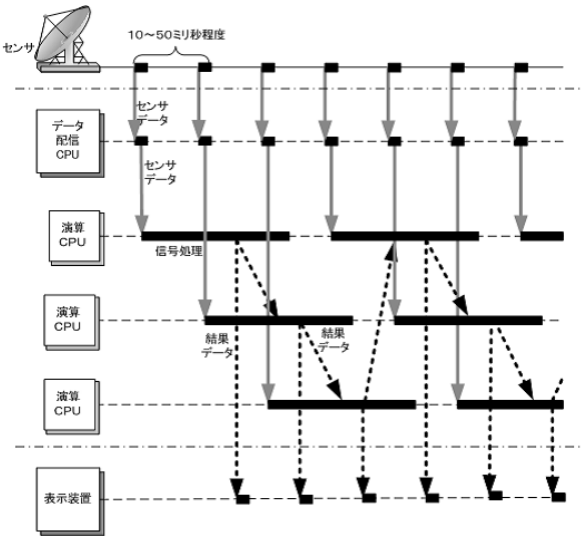

図 2　各アプリケーションの処理パターン

うためにはセンサからのデータと直前の演算結果が必要となる。演算処理に遅延が発生すると後続の処理にも遅延が波及し、処理継続が困難になるため、各 CPU 上では時間制約を厳守した処理（ハードリアルタイム処理）が行われる。

2.2　ターゲットシステムの要求

本研究のターゲットは我々の生活を支えるシステムであり、システムに故障が発生しても処理を継続することが求められている。さらに、故障時に処理を継続する場合において、以下の点を実現する必要がある。

高重要度処理の性能維持：　システムを構成する各信号処理アプリケーションは「重要度」を持ち、障害が発生した場合には、より重要度の高いアプリケーションは優先させて動作を継続させる必要がある。アプリケーションの重要度はオペレータの操作によって動的に変化するため、アプリケーションの重要度を常に監視し、状況に応じて適切な対処を行う必要がある。

演算データの継続：　本システムのアプリケーションでは、過去の演算結果とセンサデータを使用して演算を行い、演算結果の精度を徐々に向上させていくことを特徴とする。CPU 故障などによって演算結果が送信されないと、過去の演算は全て無駄になってしまう。このように、処理を継続するためには性能を回復するだけでなく、次に処理を行う CPU に確実に演算結果を送信することにより、データの継続も実現しなければならない。

2.3　これまでの開発成果

システムの高信頼化に関し、我々は高信頼並列処理を実現するための研究開発を行っており、これまでにシステムの高信頼化を実現する通信ミドルウェア：HAM(HA Middleware)[1] を開発している。

HAM の特徴は以下の通り：

- 故障の検出・故障からの復旧実現：HAM では、CPU、アプリケーション、ネットワークなど、システム構成要素の故障や各故障の重度を判別することができる。そして、アプリケーション停止などの復旧可能な故障を検出した場合には、自動的に CPU を再起動し、処理を復旧させることができる。HAM を用いることによって、数ミリ秒オーダーでの故障検出が可能である。
- システムの階層化管理：HAM ではシステム全体をグループに分割し、各グループに属する CPU から「グループマスタ」となる CPU を 1 台ずつ選択する。さらに、全グループマスタの中から 1 台「システムマスタ」を選択する。システムマスタが各グループマスタの障害検出を行い、グループマスタがグループに属する全 CPU に対して障害監視を行う「階層化管理」を行うことにより、障害の検出を効率よく実現できる。また、障害などにより他のグループの CPU 構成が変更された場合には、この階層化構造を利用することによりシステムマスタ、グループマスタを経由して全 CPU に通知される。

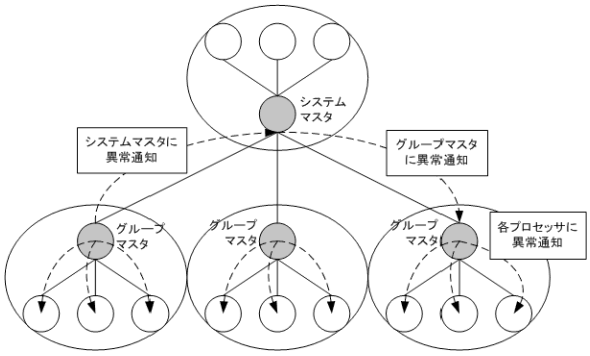

図 3　HAM のシステム階層化管理

- システムの動的構成変更のサポート：HAM では、グループ内の CPU の一覧とその障害箇所を「システム構成テーブル」(**表 1**) に示し、アプリケーションに提供している。アプリケーションは、通信時に本テーブルを参照して通信相手の状態（正常・異常）を確認してから処理を行うことにより、故障 CPU への通信防止（縮退）や再参入 CPU への通信開始（復旧）が可能となる。

表 1　システム構成テーブル

CPU	状			態		
No.	CPU	通信路 1	…	通信路 N	アプリ	結果
0	正常	正常	…	正常	正常	正常
1	正常	異常	…	正常	正常	正常
2	正常	異常	…	正常	異常	異常
…	…	…	…	…	…	…
N	異常	異常	…	正常	異常	異常

2.4　本研究の課題と解決方法

このように、HAM を用いることによって処理の縮退・復旧を行うことが可能となる。しかし、今回のターゲットシステムには 2.2節に示すシステム要求があり、現状の HAM をそのまま使用した場合、次のような問題が生じる。

- 性能縮退中は、全センサデータを処理できない · · · 1 台 CPU 故障が発生すると、処理性能は $1/N$(N: アプリ

ケーションで使用する演算 CPU の数) だけ低下する。つまり、 N 回に 1 回の頻度でセンサデータの取りこぼしが発生してしまう。性能縮退時間が長くなるほどセンサデータの取りこぼしが多くなるため、演算結果の精度が劣化していく。

- 故障発生後、演算結果を継続できない · · · 演算 CPU が処理実行中に故障した場合、最新の演算結果を次の CPU に送信できない。そのため、処理は故障 CPU で途切れてしまい、最初からやり直しとなってしまう。

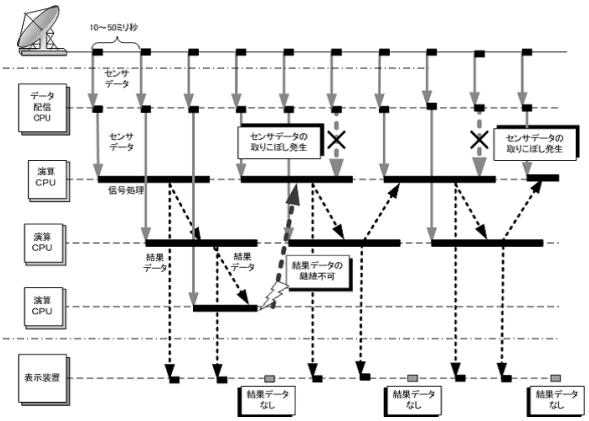

図 4 現行の HAM 適用時の問題点

今回のターゲットシステムでは、 2.2節に示すように、故障が発生した場合には、少なくとも重要な処理のリアルタイム性能を維持できればよく、重要度の低い処理は性能が縮退しても問題はない。つまり、低重要度処理を行っている計算資源を高重要度処理に動的に割り当てることができれば、高重要度処理の性能は保つことができる。

また、 CPU の故障によって最新の演算結果を紛失した場合、過去の演算結果を復元して使用することができれば、結果の精度は低下するが、演算結果の継続は可能となる。

本研究では、 HAM を拡張するとともに、 HAM の階層管理機能と連携して動作する分散ミドルウェアを設計する。これらのミドルウェアでは計算資源を高重要度処理に動的に割り当てるとともに、過去の演算結果を復元し、高重要度処理の性能維持および演算データの継続を実現する。

3. 分散ミドルウェアの設計

前述の通り、低重要度の処理を行っている CPU を高重要度処理に割り当てることによって高重要度処理の性能を保持し、さらに、過去の演算結果を再利用することにより故障発生時の演算データの継続を目指す。

これらの機能は、 HAM の拡張およびミドルウェアの新規設計により実現する。設計にあたり、 HAM とミドルウェアの役割分担は以下の通りとする。

- HAM の役割： CPU 移行用制御データの転送を行う。
- ミドルウェアの役割： CPU の離脱・参入を制御する。

2.2節に示すシステム要求を実現するために、以下の設計を行った。

- CPU の動的な移行を可能にするために：
  (1) 制御情報伝達の実現： HAM の階層化管理機能を拡張し、 CPU 故障情報だけでなく、 CPU 移行に必要な情報の伝達も可能にする。
  (2) CPU 縮退・参入の管理： HAM と分散ミドルウェアの連携により、故障発生時における高重要度処理へ CPU の再割付や、信号処理へのスムーズな参入を可能にする。
- 演算データの継続を実現するために：
  (1) 過去のデータの復旧：故障により演算データの継続が途切れてしまった場合、故障する前まで処理を遡ることにより、過去の結果データを利用して処理を再開する。
  (2) API の定義：(1) の処理をアプリケーションで実現すると、アプリケーションの記述が複雑になる。 API を提供することにより、演算データ継続のための処理を分散ミドルウェア内に隠蔽する。

次節以降で設計内容について述べる。

3.1 制御情報伝達の実現

本システムでは、 CPU の移行などに必要な CPU 間の制御データの交換はすべて HAM の階層化管理機能を用いて行うこととする。制御データの交換を効率よく行うために、従来の HAM の機能を拡張した。拡張項目は以下の 2 点：

階層化管理方法の変更： グループマスタ・システムマスタを以下の通り決定する。

- グループマスタ · · · システム全体をアプリケーション単位でグループに分割し、同一アプリケーションの処理を行っている CPU のうち「データ配信 CPU」をグループマスタとする。
- システムマスタ · · · 複数台あるグループマスタのうち、重要度の最も低いアプリケーションのグループマスタをシステムマスタとする。

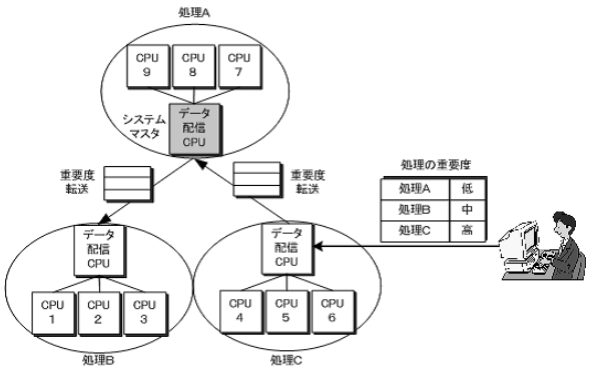

図 5 本システムにおける階層化管理とアプリケーションの重要度の伝達

本システムは、アプリケーションの重要度は動的に変更されるため、システムマスタもそれに伴って変更しなければならない。重要度の変更は、各データ配信 CPU(グループマスタ) に接続されている操作端末によって行われる。そこで、グループマスタは

アプリケーションの重要度を監視し、重要度変更の操作が行われた場合には、HAM の階層化管理機能を利用して全てのグループマスタに伝達する(**図 5**)。これにより、どの CPU がシステムマスタであるかを容易に把握することができる。

CPU 構成の変更： 前述の通り、HAM は、同一グループ内の全ての CPU の状態 (正常・異常) 情報を保持している。CPU 移行時には、グループ中で正常に動作している CPU の中から代替 CPU を選択するため、情報へのアクセスが容易な HAM が CPU を選択するほうが効率がよい。そこで、HAM に以下の機能を追加している。

(1) 移行 CPU の選択・伝達：低優先度処理を行う CPU の中から移行 CPU を選択し、階層化管理機能を使用して移行先のグループマスタに選択 CPU の識別子を通知する。

(2) 分散ミドルウェアとの連携：(1) の処理の後、移行 CPU が階層化管理機能によって通知されたり、故障などにより離脱 CPU を検出したりした場合には、分散ミドルウェアに対して参入・離脱する CPU を通知する。その後、分散ミドルウェアは、離脱・参入 CPU の情報を受け取ると、CPU がスムーズに処理への参入・縮退ができるよう処理を行う。

3.2 CPU 縮退・参入の管理

これまでに述べたように、HAM には故障を検出した場合に代替 CPU を選択し、移行先のグループマスタに伝達する役割を持たせている。それに対し、今回設計する縮退ミドルウェアでは、HAM から参入・縮退 CPU の伝達された後の動作を担当し、性能縮退復旧操作をスムーズに行う役割を持たせることとした。

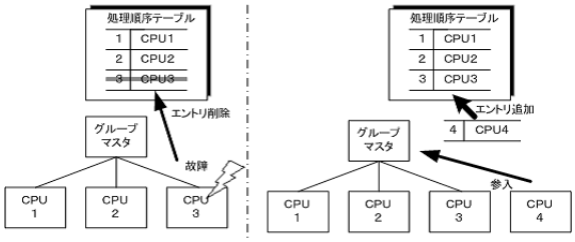

図 6 処理順序テーブルの書き換え (左：縮退時、右：参入時)

縮退ミドルウェアでは、スムーズな処理参入・縮退を「処理順序テーブル」を提供することにより実現する。HAM から参入・離脱 CPU の識別子を受信すると「処理順序テーブル」を書き換え、各 CPU に割り当てるジョブの順序を再定義する (**図 6**)。各演算 CPU は「処理順序テーブル」を参照し、新しく定義された順序に従ってデータ送信を行う。

図 7、**図 8** では、ある CPU(CPU 1) が故障し、新しく CPU (CPU 9) が参入する場合の HAM と分散ミドルウェアの連携動作、データの流れを示している。図中、HAM は、故障を検出してから代替 CPU を選択し、選択した CPU

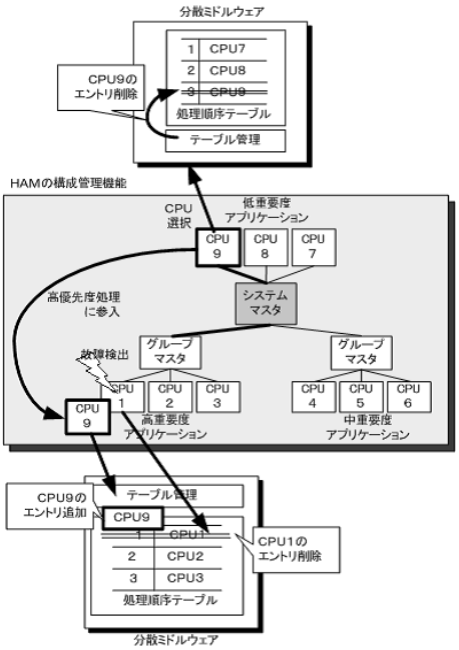

図 7 代替 CPU の参入手順

を移行元・移行先アプリケーションのミドルウェアに通知している。そして、分散ミドルウェアは、HAM から通知を受け取ると、処理順序テーブルを書き換えている。

このように、アプリケーションの CPU 構成が変化するたびに処理順序テーブルが更新されており、各演算 CPU は処理順序テーブルを参照しながらデータを送信することにより、スムーズな処理縮退・参入が実現できる。

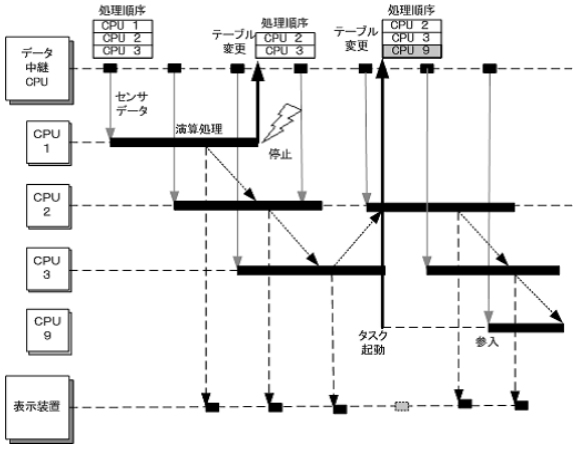

図 8 代替 CPU 参入時のデータの流れ

3.3 過去のデータ復旧の実現

これまでに述べたように、本システムのアプリケーションの処理形態では、センサデータを受け取った CPU が、演算結果を次の CPU に送信するまでの間に離脱した場合、これまでに繰り返し使用してきた演算データが途切れてしまう。その場合、過去に行われてきた演算はすべて無駄なものとなってしまう。

そこで、演算中の CPU が離脱しても過去の演算データを

用いて処理を継続するため、以下の処理を行う。

- 各 CPU は結演算果を次に処理を行う CPU に送信した後、次に処理を行うまで保持する。
- 信号処理が終わった CPU は、次に処理を行う CPU に結果データを送信した後、自 CPU の前に処理を行った CPU に「送信完了通知」を送信する。
- 一定時間内に「送信完了通知」が届かなければ、故障 CPU の次に処理を行う CPU に対し、自 CPU が持つ結果データを送信する。
- 結果データを待つ CPU は、一定時間内に結果データが届かなければ、自 CPU 内で保持していた過去の結果データを取り出し、処理を継続する。

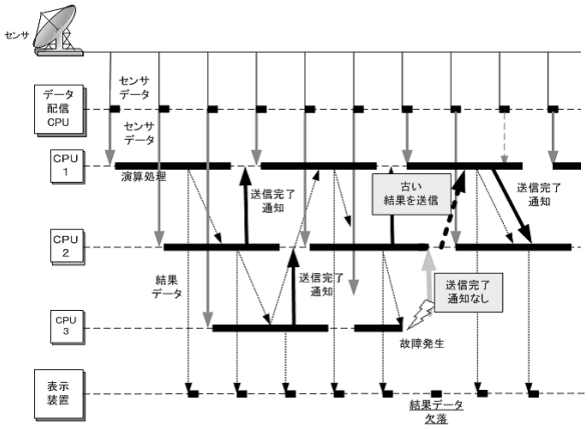

図 9 処理終了通知とデータの復旧

図 9 は、 CPU 3 が CPU 2 から結果データを受信し、CPU 1 に演算結果を送信するまでに故障したケースを示している。この場合、 CPU 2 は CPU 3 から送信完了通知が届かないため、 CPU 2 が結果データを CPU 1 に送信することによりデータを継続している。

本方式では、 CPU がただ１つ故障した場合には、故障 CPU の１つ前の演算結果を使用して処理を継続することが可能である。また、一定時間内にどの CPU からも結果データが届かなければ、自 CPU が持つ演算結果を用いて処理を継続することができる。この場合、結果データは古いものとなってしまうが、過去の演算が無駄になることはない。

3.4 送受信 API の定義

3.3節の処理をアプリケーションで実現する場合、プログラムが複雑となる。そこで、本ミドルウェアでは以下の API を提供し、データ復旧の処理を API 内部に隠蔽する。

データ送信関数 SendToNextCPU()： 本関数は、次に処理を行う CPU に対して結果データを確実に送信する。処理手順を以下に示す。

(1) ユーザより送信するデータを受け取る。
(2) 「処理順序テーブル」を参照し、自 CPU の次に処理を行う CPU にデータを送信する。
(3) 一定時間、データを送信した CPU から「処理完了通知」が届くのを待つ。
(4) (3) の通知が届いた場合には、処理を終了する。通知が届かない場合には、再度「処理順序テーブル」を参照し、 (2) で送信した CPU のさらに次の CPU にデータを送信する。

データ受信関数 ReceiveFromPreviousCPU()： 本関数を使用すると、前に処理を行った CPU から結果データを確実に受信することができる。本関数の処理手順を以下に示す。

(1) 「処理順序テーブル」を参照し、自 CPU の１つ前に処理を行った CPU から結果データが届くのを待つ。
(2) 一定時間内に結果データが届かなかった場合、再度「処理順序テーブル」を参照し、 (1) の CPU よりさらに１つ前に処理を行った CPU からのデータを待つ。
(3) 一定時間内に結果データが届かなかった場合、自 CPU 内に保持していた過去の結果データを取り出してアプリケーションに渡す。

図 10 は、演算後、 CPU2 が、結果データを次に処理を行う CPU(CPU3) に送信する直前に故障した場合の API の動作を示したものである。

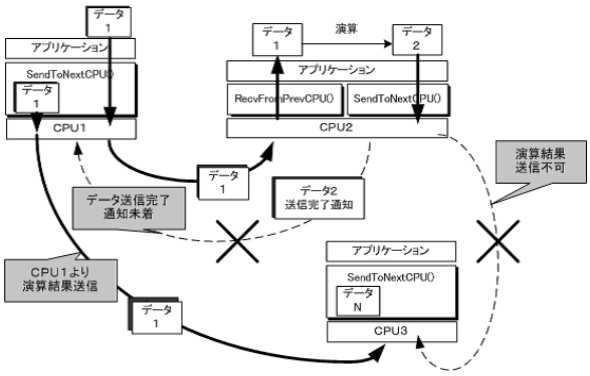

図 10 送受信 API の動作

図中、 CPU1 は結果データ送信後、その結果を保持する。そして、 CPU 2 からの処理完了通知を待つが、届かないため、 CPU 3 に対して、保持していたデータを送信する。一方、 CPU 3 は、 CPU 2 からの結果データ到着を待つが一定時間内に届かないため、 CPU 1 からのデータを待つように切り替える。

このように、 3.3節の動作をミドルウェア上で実現し、API 内に隠蔽することにより、アプリケーションは送信完了通知やデータの保持などといった複雑な処理を行うことは必要なく、「センサデータ受信→過去の結果データ受信→演算→結果データ送信」という処理だけを記述すればよい。

4. 設計内容まとめ

4.1 システム全体の動作

設計したミドルウェアの全体構成を**図 11** に示す。図中、網掛け部分が今回の設計箇所である。

- HAM の追加機能：
  - アプリケーション重要度管理：システムマスタは、

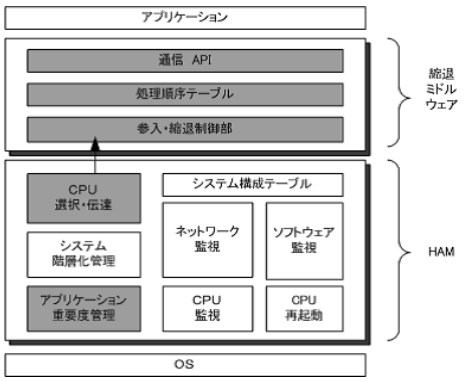

図 11 ミドルウェアのソフトウェア構成

アプリケーションの重要度に応じて動的に変化する。本機能でアプリケーションの重要度を管理することにより、システムマスタとなっている CPU を明確にする。

– CPU 選択・伝達：HAM により代替 CPU を選択し、縮退ミドルウェアに通知する。

- 分散ミドルウェアの機能：
  – 処理順序テーブル：各 CPU へのジョブの割り振り順序を示すテーブル。
  – 参入・縮退制御部：HAM の CPU の故障・復旧検出をシステム構成テーブル経由で認識し、処理順序テーブルを書き換える。
  – 通信 API：演算 CPU 間でのデータ送受信を行う。

故障検出時から参入完了までの各構成要素の動作を**図 12** に示す。故障を検出してから CPU が高重要度アプリケーションの処理を開始するまでの動作は以下の手順になる。

(1) HAM は故障検出を分散ミドルウェアに通知し、分散ミドルウェアは「処理順序テーブル」を書き換える。
(2) 故障検出後、システムマスタ上の HAM は、低重要度処理を行っている CPU から代替 CPU を 1 台選択し、低優先度処理を停止、高優先度処理を起動する。
(3) HAM は代替 CPU をシステムマスタ上のミドルウェアに通知する。
(4) ミドルウェアは「処理順序テーブル」を書き換え、代替 CPU を処理から離脱させる。
(5) HAM は、代替 CPU を高優先度処理のマスタ CPU に通知する。本通知は、 HAM によってミドルウェアに転送される。
(6) ミドルウェアは「処理順序テーブル」を書き換え、代替 CPU を処理に参入させる。

4.2 今後の課題

今回の設計では、重要度の高い処理に障害が発生してもシステムを停止させることなく処理を継続するための機能、および、演算中に故障が発生した場合の演算データ継続を実現するための機能設計を行った。

今後、センサデータ周期内 (10 ～ 30 ミリ秒) で性能復旧を実現し、センサデータの取りこぼしを発生させることなくリアルタイム処理を継続するために、以下について検討する。

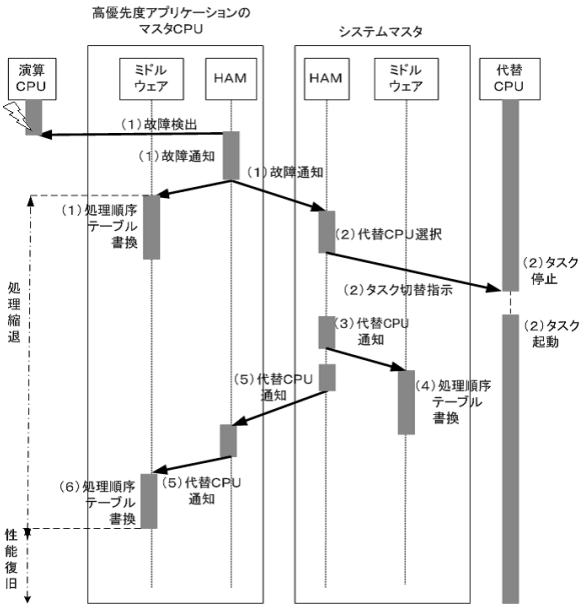

図 12 各構成要素の動作

- レプリカ起動による起動時間の短縮：低重要度アプリケーションの CPU 上では、高重要度処理への移行を想定し、あらかじめ高重要度アプリケーションのタスクも起動し、待機させておく。これにより、タスク起動に要する時間を短縮可能である。
- 構成要素間の連携オーバーヘッド短縮：HAM と分散ミドルウェアの連携を高速に行うため、タスク構成の最適化やタスク間通信の高速化・効率化（ギガビット通信、ゼロコピー通信など）を行う。

今後、これらを適用し、性能評価を行なうことにより、システム要求実現の可能性について検証を行う。

5. おわりに

今回、複数の信号処理アプリケーションによって構成されたしステムにおいて、システム故障発生時に重要度の高いアプリケーションの継続動作を実現するミドルウェアの設計を行った。今後、試作・評価を行うとともに、チューニングを行うことによりオーバーヘッドの低下を図り、実システムへの適用を目指す。

参 考 文 献

1) 村山 和宏, 落合 真一： 大規模分散システムに向けた高信頼化機構の設計, 情報処理学会研究報告 2003-DPS-111, pp191–196.
2) 村山　和宏, 落合　真一, 山口　義一：MPI/SP におけるクラスタ統合方式の設計, マルチメディア通信と分散処理ワークショップ論文集, pp85–90 (2001)．
3) 村山 和宏, 落合 真一： MPI/SPx におけるクラスタ間データ交換の実現, 情報処理学会研究報告 2004-DPS-116, pp1–6.