# 動的リソース管理ミドルウェアの実装

村山和宏† 佐藤裕幸†
目黒正之† 落合真一†

センサデータ処理システムなどの分散システムにおいて，システムに負荷の変動があってもリアルタイム処理を継続することが求められている．本要求の実現に向け，デッドラインミス発生時において，その原因となったプロセスをシステム内の任意の余剰リソースを持つ計算機に再配置することにより，システムのリアルタイム処理継続を実現する「動的リソース管理 (DRM：Dynamic Resource Management)」技術の研究が行われている．我々は，DRM 技術の代表的な実装である DeSiDeRaTa ミドルウェアをベースとし，プロセス再配置前後での演算結果の継続を実現するための拡張開発を行った．本開発の DRM ミドルウェアでは，Passive Replication 多重化モデルを導入する．そして，将来再配置先と想定される計算機に予め待機系プロセスを起動し，デッドラインミス発生を予測して直前に処理を再配置することによりデッドラインミスの発生を防止する．また，任意の計算機で動作する待機系プロセスに対して常用系プロセスが演算結果を送信することにより，プロセスの系切替時の演算結果を継続する．さらに，アプリケーションでの演算結果の等価化処理を実現するための通信機能や API を提供することにより，Passive Replication 多重化モデルの適用を容易にする．本稿では，DRM ミドルウェアの実装内容について述べる．

# Implementation of the Dynamic Resource Management Middleware

Kazuhiro Murayama,† Hiroyuki Sato,† Masayuki Meguro† and Shinichi Ochiai†

Dynamic real-time systems, such as sensor data processing systems, should satisfy deadlines in case of a varying external environment. To achieve this demand, we have been developing the "Dynamic Resource Management (DRM)" middleware which provides monitoring resource status and utilization, controlling how resources are allocated to processes, and reasoning about effective ways to allocate resources. Our DRM middleware, based on the DeSiDeRaTa middleware which is the standard middleware of DRM technology, has following features, (1) prediction of deadline miss based on past behaviour, (2) allocation of CPU resources before occuring QoS failure, (3) provision of data consistency mechanism between replica processes based on Passive Replication fault tolerant model, (4) provision of group communication functionality and APIs which facilitates creating applications. In this paper, we describe the design of our middleware.

## 1. はじめに

センサデータ処理システムなどの大規模分散システムにおいて，システムを構成するプロセスに負荷の変動があってもシステム全体のリアルタイム処理を継続することが求められている．従来は，各プロセスの最大負荷を想定してシステムを構築するとともに，各プロセスの処理量が最大となっても制限時間内に処理が終わるよう各プロセスをあらかじめ並列化することにより本課題を解決してきた．

現在，高精度な物体識別の実現に向けて，センサから大容量データがシステムに入力されるようになっており，各プロセスの処理量は増加傾向にある．そのため，現状の手法でシステムを構築した場合には膨大な台数の計算機が必要となる．

一方，各プロセスの処理量は，従来手法で構築したシステムで想定する最大数の物体を同時検出した場合にピークに達するが，そのようなケースはまれである．また，システム内の全てのプロセスの処理量が同時に増加する頻度は非常に低い．そのため，多くの計算機リソースが使用されないままとなっている．

近年，計算機リソースの有効活用に向けて，デッドラインミス発生時において，システム内に存在する任意の余剰計算機リソースを，デッドラインミス発生の原因となったプロセスに動的に割り当てることにより，システム全体のリアルタイム処理継続を実現する「動的リソース管理 (DRM：Dynamic Resource Management)」技術の研究が行われている[3)4)]．本研究により，より少ない計算機リソースでのリアルタイム処理実現が可能となる．

DRM 技術の代表的な研究成果として，DeSiDeRaTa[3)] や QARMA[4)] などのミドルウェアがある．これらのミドルウェアは，分散アプリケーションの処理時間を監視し，デッドラインミスが発生した場合には，処理時間が加したプロセスをリソースに余裕のある計算機に再配置する機能を持つ．今日，DeSiDeRaTa, QARMA ミドルウェアのコンポーネント構成が DRM 技術における事実上の標準アーキテクチャとなっている．

† 三菱電機株式会社
Mitsubishi Electric Corporation

我々は, DeSiDeRaTa ミドルウェアをベースとし, プロセス再配置前後の演算結果の継続を実現するための拡張開発を行っている[1]. 本開発では, これらの課題を解決するためのアプローチとして Passive Replication 多重化モデル[2] を導入している. 本モデルの待機系プロセスをデッドラインミスの発生に先駆けて配置することにより, プロセス再配置時間を短縮することができる. また, 常用系プロセスと待機系プロセスの間で演算結果を等価とし, デッドラインミス発生時には待機系を常用系に切り替えることにより, プロセス再配置前後での演算結果の継続を実現することができる.

上記アプローチに基づいて我々が実装した DRM ミドルウェアでは, デッドラインミスの解消, 演算結果の継続に加え, デッドラインミス発生予測に基づく待機系プロセスの事前配置, 事前の系切替によるデッドラインミスの未然防止, 演算結果等価化処理の隠蔽を特徴とする.

以下, 2節ではシステム要求実現に向けた基本構想, 3節では DRM ミドルウェアの概要, 4節では実装詳細についてそれぞれ述べる.

## 2. 基本構想

### 2.1 センサデータ処理システム概要

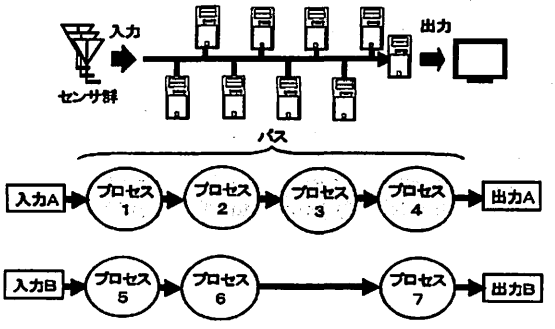


図1 センサデータ処理システムの構成概要

図1は, 我々が研究成果の適用対象としているセンサデータ処理システムの構成概要を示したものである.

本システムは複数のセンサ, 出力装置, 計算機をネットワーク接続した分散システムにて構成され, 外界の物体の検出や識別, 位置の特定といった処理を行う.

本システムでは処理形態に特徴があり, 図1に示すように, 1つの入力に対してプロセスをパイプライン状に連ねた「パス」を処理単位とし, 複数のパスが独立して処理を行う.

パスの演算時間は, センサが検出する物体の数など周囲の状況によって変動する. しかし, パスを構成する全プロセスの処理時間が一律に変動するのではなく, センサが検出した物体の種別によって, 演算時間が変動するプロセスは異なる.

本システムでは, 各センサより数百ミリ秒から秒オーダーの間隔で周期的にパスに対してデータが入力されており, システムに十分なリソースがある限り各パスの終端の出力装置には次の周期までに演算結果が届けられる必要がある. パスを構成する各プロセスでは1周期前の演算結果が必要である.

### 2.2 Passive Replication 多重化モデルの導入

図1のセンサデータ処理システムにおいて, 従来は, 各プロセスの処理量が最大となっても制限時間内に処理が終わるよう, プロセスを N 個のサブプロセスに分割（並列化）し, 各サブプロセスが計算機を占有できるようにシステムを構築してきた（図2上）.

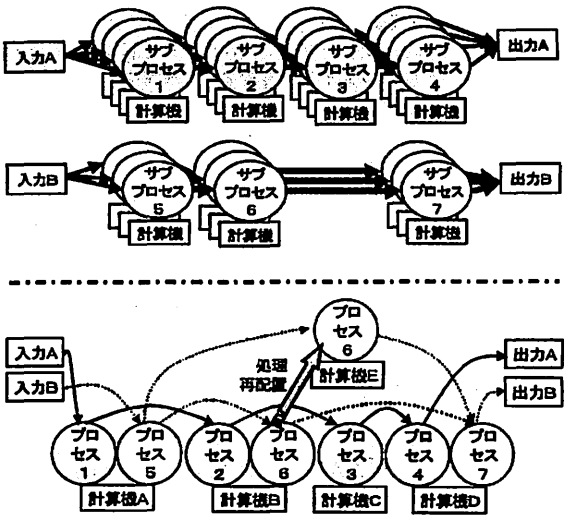


図2 従来のプロセス配置 (上) と DRM 技術適用後のプロセス配置 (下)

このようなシステムに対し, より少ない計算機でのリアルタイム処理を実現するために, DRM 技術の適用が検討されている. DRM 技術を適用した場合のプロセス配置例を図2下に示す. DRM 技術の導入により, 複数のプロセスが計算機リソースを共有し, あるパスにデッドラインミスが発生した場合にはプロセスを他の計算機に再配置することによりデッドラインミスを解消することができる. 図2ではデッドラインミスが発生した場合に, プロセス6を計算機Bから計算機Eに再配置する例を示している.

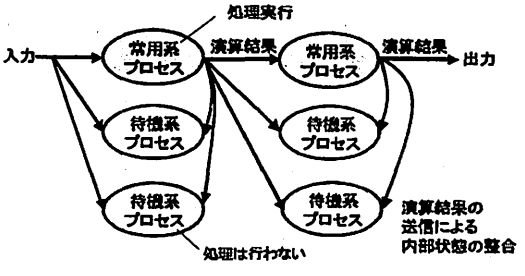


図3 Passive Replication モデルのパイプライン処理への適用

図2に示すように, DRM 技術を導入することによって, より少ない計算機でシステムを構築することが可能となる.

我々は, デッドラインミス発生の防止と演算結果継続を両立させるため, DRM 技術に加え, 従来は分散システムの高信頼化のために用いられる Passive Replication 多重化モデルをシステムに導入する. Passive Replication 多重化モデルとは, 待機系プロセスを任意の計算機にあらかじめ配置し, 常用系が待機系に演算結果を送信しており, 待機系が常用系に切り替えた場合に演算結果を引き継ぐことができるという

ものである（図3）．

本稿ではDRM技術とPassive Replication多重化モデルを組み合わせ，常用系プロセスと待機系プロセスの間で演算結果の等価化を行うとともに，デッドラインミス発生が予想される場合に待機系プロセスに処理を事前に割り当てることにより，デッドラインミス発生の防止と演算結果の継続の両立を実現する．

## 3. DRMミドルウェア概要

### 3.1 DeSiDeRaTaの構成

本稿で開発するDRMミドルウェアのベースとなっているDeSiDeRaTaのコンポーネント構成を図4に，また，以下に概要を示す．

**Software Monitor：** システムに1つ存在し，プロセス，パスの処理時間を収集し，デッドラインミスを検出する役割を持つ．デッドラインミス発生時にはResource Managerに通知する．

**Resource Manager：** システムに1つ存在し，Software Monitorからのパスのデッドラインミス発生通知を受信した場合にプロセスの再配置先計算機を決定する．

**Control：** 各計算機上で動作し，Resource Managerが決定したプロセス再配置方法に従いプロセスの停止，再起動を行う．

**Resource Monitor：** 各計算機上で動作し，ハートビートをResource Managerに送信する．

**Specification：** Software Monitor, Resource Managerが保持する．各プロセスの平常時の処理時間や各パスの処理周期，処理時間増加時の分割数（並列度）などをSoftware Monitor, Resource Managerに提供する．

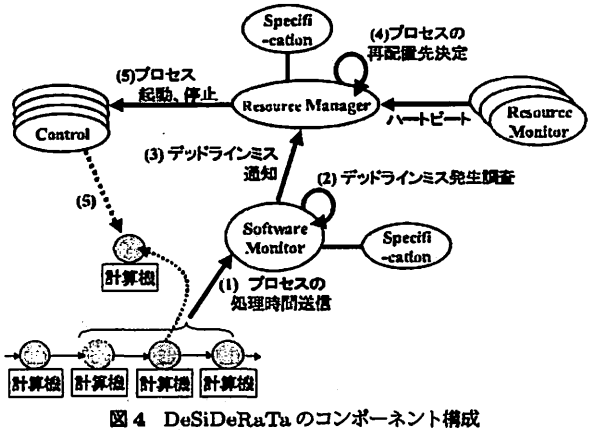


図4 DeSiDeRaTaのコンポーネント構成

パスのデッドラインミス発生時におけるDeSiDeRaTaミドルウェアの処理手順を図4を用いて以下に示す．

各プロセスが処理の開始時刻，終了時刻をSoftware Monitorに送信し(図4中(1))，Software Monitorはこれらの時刻情報からパスのデッドラインミス発生の有無を調査する(同(2))．デッドラインミスを検出すると，Software MonitorはResource Managerにその旨を通知し(同(3))，Resource Managerはデッドラインミス発生の原因となったプロセスを特定するとともに，プロセスの再配置先計算機を決定(同(4))し，Controlがプロセスの起動，停止を行う(同(5))．

なお，DeSiDeRaTaミドルウェアではデッドラインミスが発生してからプロセスの再配置処理を開始するため，再配置処理を行っている間はデッドラインミスが継続発生する．

### 3.2 本稿のDRMミドルウェアの構成

本稿にて実装したDRMミドルウェアを図5に示す．本DRMミドルウェアではPassive Replicationモデルの導入に伴い，DeSiDeRaTaミドルウェアにReplica Manager, Replica Controlコンポーネントを追加した（図5中破線）．

Replica ManagerはResource Managerが保持するコンポーネントである．待機系プロセスと起動ホストの対応を保持する役割を持つ．そして，Replica ControlはConrolが保持するコンポーネントである．Passive Replicationモデルではプロセスは待機系と常用系で処理内容が異なるため，本コンポーネントがプロセスに系の種別を通知する．

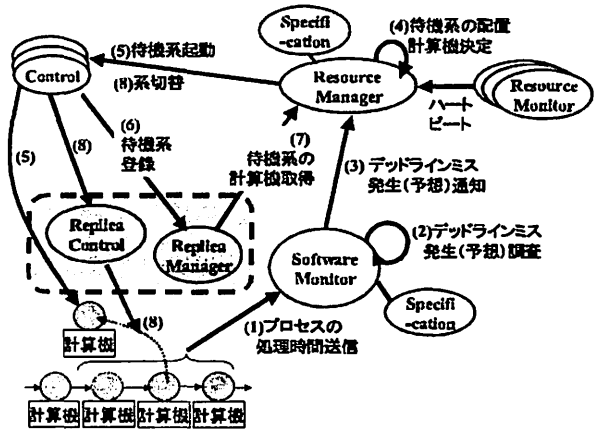


図5 実装したDRMミドルウェアのコンポーネント構成

本稿のDRMミドルウェアの動作を図5を用いて以下に説明する．本DRMミドルウェアは，デッドラインミス発生が予想される場合とデッドラインミス発生間近の場合の2つの状況で動作する．

各プロセスが処理の開始時刻，終了時刻を送信し（図5中(1)），Software Monitorが過去N回の処理時間の傾向をもとに将来のデッドラインミス発生の有無を調査する（同(2)）．デッドラインミス発生が予想される場合，Software MonitorはResource Managerにデッドラインミス発生予想を通知し（同(3)），Resource Managerは原因となったプロセスの再配置先を決定（同(4)）後，Control経由で待機系プロセスの起動，停止を行う（同(5)）．起動された待機系プロセスとそのホストはReplica Managerにより管理される（同(6)）．

そして，デッドラインミス発生間近になると，Software MonitorはResource Managerに再度通知（同(3)）し，Resource ManagerはReplica Managerより待機系プロセスが配置されているホストを取得(同(7))し，Control, Replica Control経由で待機系プロセスと常用系プロセスの

切替を行う（同(8)）.

## 4. 実装詳細

3.2節のDRMミドルウェアがシステム要求を満たすためには, (1) デッドラインミス発生を予測すること, そして, (2) 待機系プロセスをデッドラインミスの発生しない計算機に事前に配置すること, そして, (3) 任意の計算機で動的に起動する待機系プロセスと常用系プロセスとの間で演算結果の等価化が行えることが必要となる. 以下, (1) ～ (3) の実現方法について述べる.

### 4.1 デッドラインミス発生の予測

パスのデッドラインミス発生の予測は, 時系列データ解析において一般的な手法である最小二乗法を用い, 時刻と処理時間に関する一次近似式 ($y = ax + b \pm \Delta$($\Delta$ は変動の誤差)) を求めることにより行う.

DRMミドルウェアにて直近N回のパスの処理終了時刻および処理時間を保持する. $x_i$ をi番目のパスの処理終了時刻, $y_i$ をその終了時刻におけるパスの処理時間とした場合, a, bは

$$a = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

$$b = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i y_i \sum_{i=1}^{n} x_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

として求めることができる[5]. そして, 求めた近似式の傾きaの値が正であればパスの処理時間が増加傾向であり, デッドラインミスが将来発生する可能性がある. その際には4.2節に示す方法で待機系プロセスを起動する.

また, 求めた式の変数yにパスの制限時間を代入することにより, デッドラインミス発生予想時刻を求めることができる. この時刻をもとに, Software Monitor は Resource Manager にデッドラインミス発生間近であることを伝え, 常用系プロセスと待機系プロセスの切り替えを行う.

### 4.2 待機系プロセスの事前配置

従来のPassive Replicationモデルでは任意の計算機に待機系プロセスを配置している. しかし, これでは必ずしもリアルタイム処理が継続可能な計算機に待機系プロセスが配置されているとは限らない. また, 全ての待機系プロセスを配置するため, プロセス数が多い場合には消費メモリ量も膨大なものとなる. 本稿では, このような問題を避けるため, デッドラインミスが予想される場合に, デッドラインミス発生の原因となるプロセスのみ配置する. そして, 将来の使用リソース量をもとに, それだけの空きリソースを持つ計算機に待機系を配置する.

以下, 待機系プロセス配置計算機の決定方法を示す.

**待機系プロセス配置計算機の決定アルゴリズム：**

4.1節で求めた方法にて将来のデッドラインミス発生が予想されると, パスの中で処理時間が増加傾向にあるプロセスを, 待機系を起動するプロセスとして決定する.

本稿では, 上記プロセスの待機系を配置するにあたり, (1) プロセス再配置を極力行わないようにするため, 将来にわたってデッドラインミスが発生しない計算機に配置し, (2)(1) を実現しつつ, システム内の計算機リソースを極力有効活用するために, 空きリソースの小さい計算機に配置する, の2点の方針に基づき配置計算機を決定する.

待機系プロセスの配置先決定手順を以下に示す.

(i) デッドラインミス発生時刻よりも先のある時刻tにおいて, 待機系プロセスを配置可能な計算機のうち, 空きリソースが最も小さい計算機を待機系プロセスの配置計算機として決定する.

(ii) (i) で再配置計算機が決定しない場合, デッドラインミス発生予想時刻において待機系プロセスが配置可能な計算機が存在することを確認する. 待機系プロセスを配置可能な計算機がなければ (iii) 以降を行う.
待機系プロセスが配置可能な計算機が存在した場合, 初期値として変数vの値をデッドラインミス発生時刻として (ii)-1, (ii)-2 を繰り返す. そして, 時刻tと時刻vの差が$\Delta$ （ユーザが定めた最小値）以下になった場合, その直前に抽出した配置可能な計算機を待機系配置計算機とする. 配置可能な計算機が複数あった場合, 空きリソースが最も小さい計算機を待機系プロセスの配置計算機として決定する.

(ii)-1. 時刻tと時刻vの中間の時刻（uとする）において, 待機系プロセスを配置可能な計算機を抽出する.

(ii)-2a. 時刻uにおいて待機系プロセスを配置可能な計算機が存在する場合, uよりも先の時刻において配置可能な計算機が存在する可能性があるため, 時刻uをvとして (ii)-1 を行う.

(ii)-2b. 時刻uにおいて待機系プロセスを配置可能な計算機が存在しない場合, 配置可能な計算機は時刻vとuの間に存在するので, 時刻uをtとして (ii)-1 を行う.

(iii) ((i), (ii) で再配置計算機が決定しない場合) デッドラインミス発生時刻において, 現在処理中のプロセス処理をN分割し, N個の待機系プロセスが配置可能な計算機群を待機系プロセス配置計算機とする.

(iv) ((i) ～ (iii) で再配置計算機が決定しない場合) デッドラインミス発生時刻において, 他のプロセスを他の計算機に再配置した後に待機系プロセスが配置可能な計算機を待機系プロセス配置計算機とする.

このように, 将来のある時間の使用リソース量を予測し, そのリソース量をもとに待機系プロセスの配置計算機を決定することにより, 他のプロセスの処理時間に変動がなければその時間までデッドラインミスを防止することができる.

また, 本手順において, 待機系配置計算機の候補が複数あった場合には, 空きリソースの最も小さい計算機を選択することにより, 各計算機に大きな空きリソースを生み出すことができ, 処理時間の長いプロセスの再配置が可能となる.

本方式では将来のリソース使用量をもとに再配置を行っており, 必要以上の計算機リソースを配置可能な計算機の有無

を調査している．したがって，(i) で設定した時刻によってはプロセスの再配置先が決定しない場合がある．この場合は (ii) で示したように二分法によって時刻 t を現在時刻に近づけていくことにより，プロセスが必要とするリソースを減らしつつ，かつ現在よりも極力先の時刻までデッドラインミスが発生しない再配置先を決定することができる．

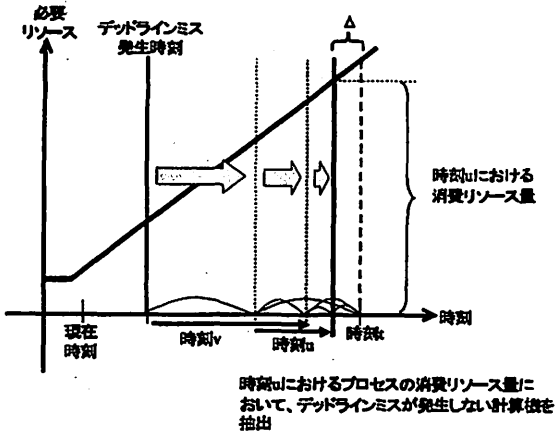


図 6　(ii) の再配置計算機の決定方法の考え方

**待機系プロセス配置計算機の抽出アルゴリズム：**

以下，(i) ～ (iv) で使用する，待機系プロセス再配置可能な計算機の抽出方法について述べる．待機系プロセスが配置可能な計算機とは，(a) 再配置対象のプロセスが制限時間内に処理完了し，かつ，(b)(a) を満たす計算機上で動作する他の全プロセスが制限時間内に処理を完了する計算機である．

本稿では条件 (a), (b) の判定を以下のようにしている．

- (a) の判定方法：リソース割り当て対象プロセスを $p_a$，プロセス $p_a$ が属するパス $P_a$ を構成するプロセス群を $P_a = \{p_1, \cdots, p_{a-1}, p_a, p_{a+1}, \cdots, p_n\}$，時刻 t におけるプロセス $p_i (\in P_a)$ の処理時間を $Exec(p_i, t)$，パス $P_a$ の処理周期を $C(P_a)$ とする．
  時刻 t においてプロセス $p_a$ に与えられる制限時間 $D(p_a, t)$ は以下の式で求められる．

$$D(p_a,t) = D(P_a) - \sum_{i=1}^{a-1} Exec(p_i,t) - \sum_{i=a+1}^{n} Exec(p_i,t) \quad (1)$$

  さらに，計算機 A で動作するプロセスおよび $p_a$ を優先度の低い順に並べたものを，$PE^A = \{p_1^A, \cdots, p_l^A, p_a, p_{l+1}^A, \cdots, p_m^A\}$，プロセス $p_i^A (\in PE^A)$ が時刻 t において CPU リソースを占有する時間を $Cpu(p_i^A, t)$，プロセス $p_i$ の処理周期を $C(p_i)$ とする．以下の式を満たせば，再配置対象のプロセス $p_a$ は時刻 t において制限時間内に処理が可能となる．

$$D(p_a,t) - \sum_{i=l+1}^{m} \lceil \frac{D(p_a,t)}{C(p_i^A)} \times Cpu(p_i^A,t) \rceil - Cpu(p_a,t) > \Delta \quad (2)$$

  式 (1), (2) の判定を全ての計算機に対して行うことにより，プロセス $p_a$ が時刻 t において制限時間内に動作可能な計算機を全て抽出する．
- (b) の判定方法：(a) の判定によって抽出した各計算機について，その計算機上で動作する，プロセス $p_a$ よりも優先度の低いプロセスが全て制限時間内に処理完了可能かどうかを，式 (1), (2) を用いることにより判定する．優先度の低い全てのプロセスが制限時間内に処理完了可能であれば，その計算機は時刻 t においてプロセス $p_a$ に対して CPU リソースを割り当てることができる．

### 4.3　常用系と待機系の内部状態等価化

#### 4.3.1　演算結果等価化のための通信機能

本稿の実装ではプロセスが動作する計算機が頻繁に変更されることから，プロセスのロケーションを意識することなく通信できる必要がある．本ミドルウェアでは OMG（Object Management Group）の DDS（Data Distribution Service for real-time systems）[6] 仕様を参考に実装したグループ通信機能を提供する．DDS とは Publish-Subscribe 通信モデルによるプロセス間データ交換技術であり，Publisher（送信側）はトピック（データの種類）の受信を宣言した全ての Subscriber（受信側）に対し，Subscriber のロケーションを意識することなくデータを送信することができる．

本稿のグループ通信機能では，各計算機上で動作するグループ通信デーモンが全トピックの Subscriber/Publisher が存在する計算機を管理するとともに，Publisher が持つデータを，他の計算機のグループ通信機能を経由して Subscriber に送信する．

本稿のグループ通信機能の動作を図 7 を用いて説明する．

待機系プロセスが起動されると，その待機系プロセスが受信したいトピック（図 7 では「演算結果 A」）を各計算機上で動作するグループ通信デーモンに登録する（図 7 中 (1)）．グループ通信デーモンは，他のグループ通信デーモンに，新規に起動した Subscriber/Publisher と送受信するトピックを通知する（同 (2)）．その後，グループ通信デーモンは，通知されたトピックを送信する Publisher が同一計算機上にいる場合，新規起動した待機系プロセスが動作する計算機のグループ通信デーモンにデータ送信を開始する（同 (3)）．

データの送受信は，プロセスとグループ通信デーモンとの間に作成されるデータ受信用共有メモリ領域を介して行われる（同 (4)）．送信プロセスがデーモンとの間に作成した共有メモリにデータを書き込むと，グループ通信デーモンが送信を行い（同 (5)），他のグループ通信デーモンがデータ受信後，受信用共有メモリ領域に書きこむことにより受信プロセスにデータが届けられる．

#### 4.3.2　演算結果等価化処理の隠蔽

演算結果の等価化や系切替の処理をアプリケーションで実現しようとすると，従来と比べてプログラム記述が煩雑となる．本 DRM ミドルウェアでは API を提供し，これらの処理を隠蔽する．

API を用いた場合の常用系，待機系プロセスの処理手順を図 8 を用いて以下に示す．

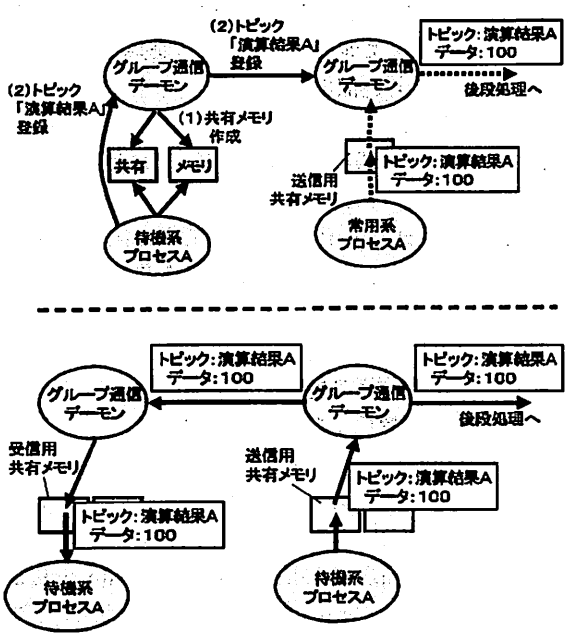


図7 グループ通信機能（上：受信トピック登録時，下：データ送受信時）

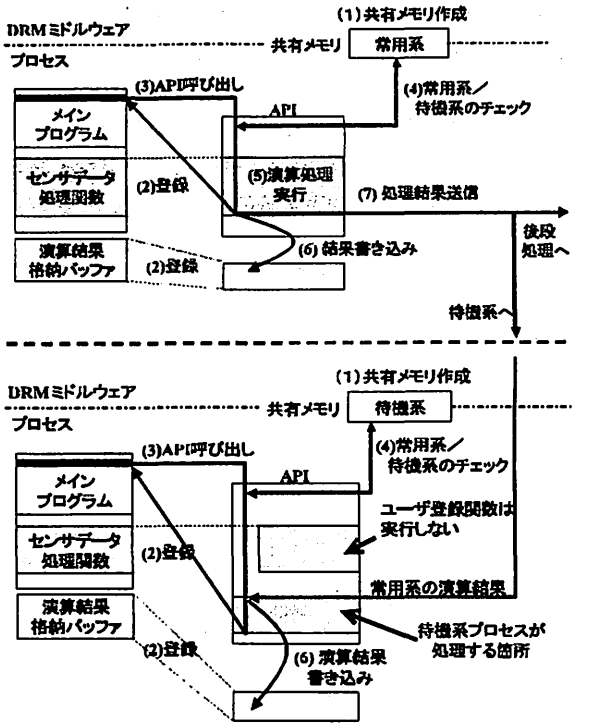


図8 プロセスの動作（上：常用系，下：待機系）

プロセス起動時，DRMミドルウェアとの間に共有メモリを作成する（図8中(1)）．系の切替は，DRMミドルウェアが共有メモリ領域に「常用系」「待機系」の種別を書き込むことにより実現する．次に，ユーザが作成したセンサデータ処理関数のポインタと処理結果を記憶する領域のアドレス，受信するトピック名をAPIに渡し，APIを呼び出す（同(3)）．APIでは，前段プロセスから演算結果が届くのを待つ．演算結果を受信すると，系の種別を調べ（同(4)），共有メモリに「常用系」と示されていた場合には，登録されたセンサデータ処理関数を実行（同(5)）し，バッファに値を書き込み（同(6)），後段プロセスおよび待機系プロセスに処理結果を送信し（同(7)），APIの処理を終了する．

なお，(4)にて共有メモリに「待機系」と示されていた場合には，APIは前段からの処理を受信後，センサデータ処理関数は呼び出さず，常用系プロセスからの演算結果を待つ．常用系プロセスからの演算結果を受信すると，DRMミドルウェアは(2)で登録したバッファに受信結果を書き込み（同(8)），APIの処理を終了する．

本APIにより常用系，待機系ともに同一プログラムで実装することができ，プロセスはAPIを呼び出すだけで系の違いによる処理の切替や演算結果の等価化が実現できる．

## 5. まとめと今後の課題

本稿では，センサデータ処理システムに向け，DeSiDeRaTaミドルウェアをベースとし，リアルタイム処理とプロセス再配置前後での処理結果継続を実現可能とするDRMミドルウェアの開発を行った．開発したミドルウェアではPassive Replication多重化モデルを導入し，プロセスの再配置先と想定される計算機に事前に待機系プロセスを配置するとともに，デッドラインミス発生を予測して事前に処理の再配置を行うことによりデッドラインミス発生の未然防止を実現した．また，任意の計算機で動的に起動する待機系プロセスに演算結果の送信を行うためのグループ通信機能や，演算結果の等価化を行うためのAPIを提供し，演算結果の継続の容易化を実現した．

本ミドルウェアの実装の有効性を確認するためには，DRMミドルウェア導入により削減できる計算機の台数，デッドラインミス削減の度合いなどについて調査を行うことが必要である．そのためには2.1節に示すような実システムが必要であり，現状では評価を行うことができない．今後，システムの模擬環境を構築し，評価を行っていく．

## 参考文献


1) 村山 和宏，大谷 治之，佐藤 裕幸，目黒 正之，宮森 信之，落合 真一：「分散リアルタイムシステムに向けた動的リソース管理ミドルウェアの設計」情報処理学会研究報告．マルチメディアと分散処理研究会報告 Vol.2006, No.26(20060316) pp.257–262.
2) R.Guerraoui & A.Schiper, "Software-based replication for fault tolerance", IEEE Computer, 30(4):pp.68-74, Aplil 1997.
3) L. R. Welch et al., "Adaptive Resource Management for Scalable Dependable Real-Time Systems.", 4th IEEE Real-Time Technology and Applications Symposium, 1998.
4) Kevin Bryan, et al., "Integrated CORBA Scheduling and Resource Management for Distributed Real-Time Embedded Systems", 11th IEEE Real-Time and Embedded Technology and Applications Symposium, 2005.
5) G. ストラング著，山口 昌哉監訳，井上 昭訳，「線形代数とその応用」，産業図書．
6) "Data Distribution Service for real-time systems specification", Object Management Group, 2003.