分散 OS Colonia における共有メモリを利用した大域的ネーム ・サービス

増田峰義[†]　杉野秀和[††]　鶴崎剛大[††]
伊達新哉[†]　Damien Le Moal[†]　山添博史[†]
五島正裕[†]　森　眞一郎[†]　富田眞治[†]

分散 OS Colonia は、我々が提案してるコンピュータコロニーのための OS である．Colonia は，並列アプリケーションの高速実行環境として共有メモリ環境を提供するとともに，システムが提供する各種サービスも共有メモリ上に実装する．そのため，複数の計算機からなる分散システム上に，高速かつ柔軟な分散共有メモリ管理を如何に効率よく実現するがシステムの性能を左右する．本稿では，このような共有メモリ環境の実現に必要な大域ネームサービスの実装について報告する．

The shared-memory based global name service for distributed operating system Colonia

MINEYOSHI MASUDA,[†] HIDEKAZU SUGINO,[††] TAKEHIRO TURUSAKI,[††]
SHINYA DATE,[†] DAMIEN LE MOAL,[†] HIROSHI YAMAZOE,[†]
MASAHIRO GOSHIMA,[†] SHIN-ICHIRO MORI[†] and SHINJI TOMITA[†]

Colonia is an distributed operating system designed for the Computer Colony we have proposed. Colonia provides user application with shared memory environment as an flexible and efficient parallel execution environment. Various system services are also provided using the shared memory environment. Thus, the implementation of the shared memory management is one of the important factor of Colonia design. This paper reports our implementation of a global name service required to realize shared memory environment.

[†] 京都大学大学院情報学研究科通信情報システム専攻
Department of Communications and Computer Engineering, Graduate School of Infomatics, Kyoto Univ.
[††] 京都大学工学部情報工学科
Department of Information Science, Faculty of Engineering, Kyoto Univ.

1. はじめに

標題の分散OS Coloniaは、我々が提唱している計算機環境である「コンピュータ・コロニー」上で動作する分散OSである。コンピュータ・コロニーとは、並列アプリケーションの高速実行を指向した、共有メモリベースの分散システムである。1) 本稿では、Coloniaが動作するハードウェア環境として、コンピュータ・コロニーの実現例の一つである、共有メモリベースのワークステーションクラスタを想定する。

コンピュータ・コロニーでは、システムの拡張性および管理方式の柔軟性を損なわず、高速な分散共有メモリの実現を目指す。そのために、分散共有メモリの管理コントローラにおける複雑な処理はソフトウェアにより実装し、その他の単純な処理ついてはハードウェアにより実装する方式を採用する。この方式による分散共有メモリ管理が、実用可能な高速性を実現するためには、ソフトウェアにより実装した部分が、十分な性能を持つことが必要となる。

Coloniaの大域的ネーム・サービスにはいくつかの種類があるが、本稿では、特にGDSと呼ばれるサービスについて述べる。Coloniaでは、負荷を分散させるために、プロセスの位置が動的に変化する。GDSが管理する大域的な情報は、動的に変化する、システム内に存在している全てのプロセスの位置等である。これらの情報は、分散共有メモリの管理コントローラが必要とする。したがって、GDSの性能は、分散共有メモリ管理全体の性能に大きく影響する。そこで、本稿ではGDSを共有メモリ上に実装することにより高速化を図る。

以下、2章では分散OS Coloniaの概要について述べる。3章では、GDSが持つ機能、およびColoniaの分散共有メモリ管理方式の中で、GDSが果たす役割について述べる。4章では、GDSの構造と、共有メモリ上への実装方法について述べる。また、その有効性についての考察を行う。

2. 分散OS Colonia

我々が提唱している計算機環境である「コンピュータ・コロニー」は、現行の分散システムが有効に活用できていない計算資源を最大限に利用することで、集中型の並列計算機に匹敵する計算能力を持つ計算環境を低価格で実現することを目標とする。分散OS Coloniaは、コンピュータ・コロニー上で動作するOSであり、物理的に分散している計算資源を有機的に統合するためのソフトウェアである。

2.1 ハードウェア環境

2.1.1 ネットワーク構成

コンピュータ・コロニーのネットワーク構成の一例を図1に示す。

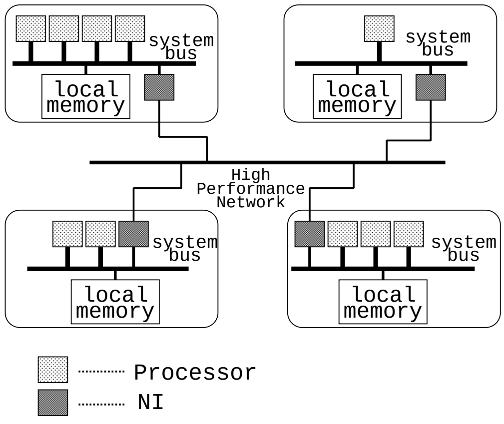

図1 ネットワーク構成例

個々のノードは独立で動作が可能なワークステーションやパーソナルコンピュータである。各ノードのシステムバスには、通信用専用ハードウェアが接続されており、分散共有メモリの管理コントローラとして動作する。各ノードに分散配置された主記憶は、ローカルな主記憶としてだけではなく、リモートノードの主記憶へのアクセスに対するキャッシュとしても使用する。キャッシュ間の一貫性制御は通信用専用ハードウェアが行い、システム全体で共有メモリベースのワークステーションクラスタを構成する。

以下、この通信専用ハードウェアをNI(Network Interface)と呼ぶ。

2.1.2 NI

NIは分散共有メモリの管理コントローラである。分散共有メモリ管理にはソフトウェアで行う処理と、NI上のハードウェアで行う処理とがある。ソフトウェアで行う処理は、NI上に搭載されたプロトコルプロセッサの上で実行される。

NIは、各ノードのシステムバスに接続されており、バス上の物理アドレスを監視し、その物理アドレスから、リモートメモリアクセスの必要性を判定し、必要があるならばノード間の通信処理を行う。

ハードウェアで行う処理には、主記憶上のデータの有効性判定がある。主記憶上のデータの有効性を示すマップを保持し、システムバスを流れる物理アドレスが指すデータの有効性を監視する。他ノードのNIとの通信の必要の有無を判定する。また、従来はソフトウェアで行われていた、通信プロトコルのフロー制御もハードウェアで行う。これにより、プロトコルオーバーヘッドを削減する。

アドレスマッピングに関連する複雑な処理はソフトウェアが行う。この処理には大域的ネーム・サービスが関わるため、詳細は3.3節で述べる。

2.2 Coloniaにおけるメモリ共有

物理的に共有メモリ空間を持たない分散システムで

あるColoniaでは、仮想的な共有メモリ空間を実現する手法として、仮想ページを単位とするShared Virtual Memory[2]を採用している。全ての仮想共有ページには、それぞれホームページと呼ぶ物理ページが存在する。Coloniaでは各仮想ページが全て、当該ホームページにマッピングされることで、仮想的なページ共有を実現する。

ホームページが物理的に離れたノードに存在している場合、アクセスレイテンシ、通信トラフィックの軽減のために、ホームページをローカルノードの主記憶上へキャッシングする。キャッシングされたページをコピーページと呼ぶ。キャッシュ・ディレクトリなどのアドレスマッピングに関わる管理情報は、ページごとに作成される。ただし、ページの内容全てを転送するのではなく、ブロック(32Byte)を単位とした転送を行う。キャッシュの有効ビットなどの一貫性を保つための管理情報は、ブロックごとに付与され、NIが管理を行う。

2.3 ミッション・ユニットモデル

2.3.1 ミッションとユニット

ミッション・ユニットモデルとは、並列プログラムを効率的に実行するために、Coloniaが提供するプログラミングモデルである。

ミッション　ミッションとは、ユーザがシステムに投入するジョブの単位である。ミッションは一つ以上のユニットからなり、これらのユニットが互いに協調動作し、並列に実行されることでジョブが進められる。ミッションには、システム内で一意な識別子であるMission IDが付与される。システムは、このIDでミッションを識別する。

ユニット　ユニットとは、ミッションを並列実行可能な単位に細分化したものである。同一ミッションに属するユニットが、異なるノード上で動作することで、ミッションは並列に実行される。ユニットは、ノード間での高速な移送が可能であり[3]、ノード間に計算負荷の偏りがある場合には、その是正のためにユニットの移送が行われる。すなわち、ミッション・ユニットモデルを採用することで、Coloniaは、分散システム全体が持つ計算資源を、効率的にユーザに提供することができる。ユニットには、同一ミッション内で一意な識別子であるUnit IDが付与される。

2.3.2 ユニットのアドレス空間

個々のユニットは、それぞれ独自の仮想アドレス空間を持つ。ユニットのアドレス空間には二つの領域があり、それぞれユニット固有領域、ミッション内共有領域と呼ばれる。ユニット固有領域には、そのユニットのスタックなどが配置される。ミッション内共有領域には、共有データやテキストなどが配置される。ミッション内共有領域は、Coloniaにおいて最も基本となる共有領域である。[4]

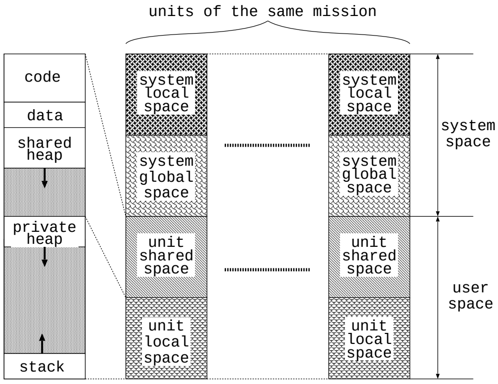

図2　仮想アドレス空間の構造

各仮想共有ページのホームページには、オーナーとなるユニットが存在し、そのページのホームユニットと呼ぶ。各ホームページに対しホームユニットはシステム内で唯一つであり、ホームページとホームユニットの対応関係は、ユーザによる明示的な変更指示がないかぎり変更されない。また、ホームページが物理的に存在するノードをホームノードと呼ぶ。

2.3.3 システムミッション

Coloniaのシステムは、ユーザのジョブと同様、ミッション・ユニットの構成をとる。すなわち、システムはただ1つのシステムミッションを持ち、システムを構成する全てのノードごとに、システムユニットを作成する。システムユニットの位置は固定で移送されない。

Coloniaの仮想アドレス空間は、ユーザ領域とシステム領域に分けられる。ユーザ領域はユーザが生成するミッションが使用する領域であり、システム領域はシステムミッションが使用する領域である。システム領域は、さらにsystem local spaceとsystem global spaceに分けられる。system local spaceには各ノードにおけるシステムユニットに固有なデータを配置し、system global spaceにはシステムユニット全体で共有するデータを配置する。

図2に、Coloniaの仮想アドレス空間の構造を示す。

3. Global Directory Service

3.1 Coloniaにおける大域的ネーム・サービス

ネーム・サービスは、OSの中心的役割の一つである。とりわけ、多数の様々なオブジェクトが物理的に分散して存在している分散システムにおいては、システム全体で共有する大域的な情報の管理を行う大域的ネーム・サービスは不可欠である。

Coloniaにおける大域的ネーム・サービスが管理する主な情報は以下の通りである。

(1)　ユーザ情報
Coloniaでは、ユーザのログイン情報といった、各ユーザに関する固有な情報は大域的に管理さ

れる。

(2) ファイルシステム
Colonia はファイルに大域的な名前を与え、その管理を行う。これにより、ユーザに位置透過なファイルシステムを提供する。

(3) マシン環境情報
コンピュータ・コロニーを構成する各マシンのスペック (CPU クロック数、メモリ容量、HDD の有無、周辺機器の有無など)、各マシン間の物理的な距離などである。

(4) 計算負荷情報
各ノードの負荷状況を大域的に管理し、スケジューラに提供する。スケジューラは、この情報を利用し、動的な負荷分散を行う。これにより、並列アプリケーションを効率的に実行できる。

(5) 分散共有メモリ管理情報
ホームページ、ホームユニット等の分散共有メモリのための情報管理を大域的に行う。Global Directory Service が行うサービスである。

本稿では特に、分散共有メモリを実現するための大域的な情報の管理を行う (5) の Global Directory Service(以下 GDS) について述べる。

3.2 GDS の機能

GDS は、分散共有メモリを実現するための大域的な情報を管理する。GDS は、NI が分散共有メモリを管理するために必要な情報を提供する。

GDS が持つ機能を以下に示す。

ネーミング
ミッションおよびユニットに大域的な識別子を与えるネーミング機構を持つ。この機能により、各ミッションおよびユニットは、システム内で唯一であることが保証される。

位置情報管理
システム内に分散して存在している、全てのミッションおよびユニットの位置情報を把握する。この情報は、ユニット間で通信を行う場合だけでなく、Colonia の共有メモリ管理機構の実現のために使用される。

ホームユニット情報の管理
各共有ページに対するホームユニットは、システム内で唯一である必要がため大域的に管理される。

以下に GDS が関わる主なイベントを示す。

(1) ミッションの生成/終了
(2) ユニットの生成/終了
(3) ミッションのノード拡張/縮小
(4) ユニットの移送
(5) ホームユニットの特定要求
(6) ユニットのノード位置の特定要求

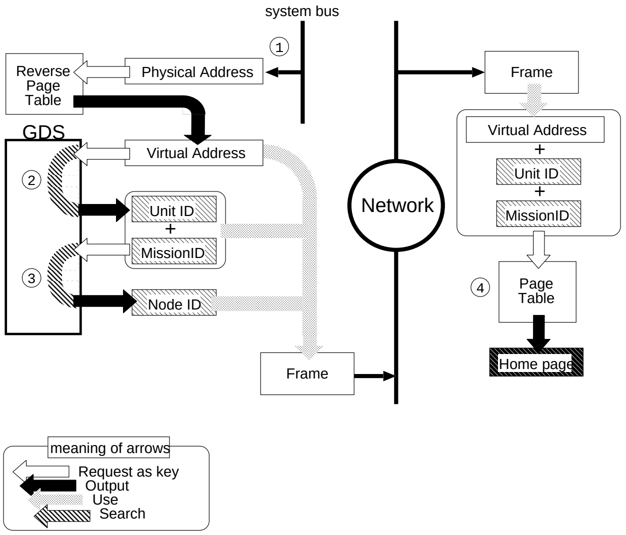

図 3　ページ共有における GDS の役割

(1)(2) は GDS のネーミング機能であり、新しい識別子の発行、使用されなくなった識別子の回収を行う。(3)(4) は位置情報の管理機能を使用する。(5)(6) の要求は共有メモリ管理のための処理である。主に NI 上で動作する分散共有メモリ管理コントローラが要求する。この要求は、NI が動作する度に発生するため、(1)~(4) までの要求と比較して頻度が大きいと考えられる。

3.3 分散共有メモリ管理における GDS の役割

Colonia の分散共有メモリ管理の一連の処理の中で、GDS は以下の二つのサービスを提供する。

- ホームユニットの特定
- ノード位置の特定

図3および以下のチャートで、Colonia におけるネットワークを介したメモリアクセスの手順を示す。GDS は (2) と (3) の処理に関連し、これらの処理は NI 上のプロトコルプロセッサ上で動作するソフトウェアにより実行される。(1) と、(3) のフレーム作成の処理は NI 上に実装されたハードウェアにより高速に実行される。

(1) ネットワークを介したリモートアクセスの必要性判定
NI が、システムバス上を流れる物理アドレスをスヌープする。その物理アドレスが指すデータの有効性を示すマップを参照し、通信の必要性を判定する。

(2) ホームユニットを特定
逆索きページテーブルを索き、物理アドレスから仮想アドレスを取得する。得られた仮想アドレスをキーとして、GDS が管理しているテーブ

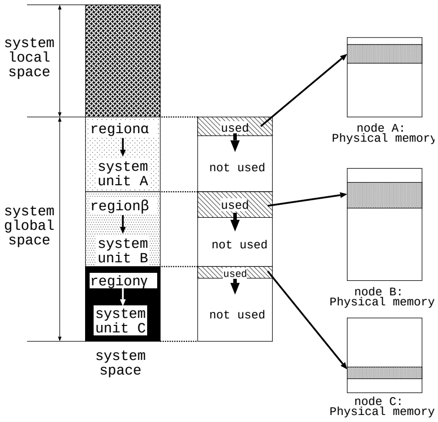

図 4　system global space

ルを検索し、そのページのホームユニットを特定し、その Mission ID と Unit ID を取得する。

(3) ホームノードの特定
ホームユニットの Unit ID、Mission ID をキーとして、GDS が管理しているテーブルを索き、そのページのホームノードの node ID を取得する。
NI が、送信用フレームを作成し、先程得られたホームノードに向けて送信する。

(4) ホームページへのアクセス
ホームノードでフレームを受信し、Mission ID と仮想ページ番号から、ホームノード内での物理アドレスを求めホームページにアクセス。

4. 共有メモリを利用した GDS の実装

本章では GDS を、システムミッションのノード間共有領域である、system global space に配置する実装について述べる。まず、system global space の共有管理手法について述べ、次に、GDS の構造と、その分散処理について述べる。続いて、Colonia において GDS を、共有メモリを利用して実装することの有効性について考察する。

4.1 system global space

system global space は、システムミッションのノード間共有領域であり、ユーザミッションの共有領域と同様の方式で、NI により管理される。つまり、system global space の各共有ページごとに、ホームシステムユニットが唯一つ設定されており、リモートアクセスの際にはこの情報が必要となる。ホームシステムユニットの位置は固定なので、位置情報は静的に決定できる。

共有メモリ管理のための情報を提供する GDS は system global space に配置される。

system global space の共有管理に必要な情報は、system global space の各共有ページのホームユニットを静的に割り振ることで得ることができる。すなわち、system global space を適当な大きさの領域に区分けし、それぞれの領域に対し、ホームを担当するシステムユニットを静的に割り振る。図 4にこの様子を示す。図では system global space を 3 つの領域α、β、γ に区分けし、各領域にシステムユニット A、B、C を割り振っている。

ホームシステムユニットを静的に割り振ることで、system global space に属するページのホームページは、仮想アドレスから一意に決定できる。領域とシステムユニットの静的に割り振られた対応関係は、システムの起動時にコンフィギュレーションされ、システムが拡張/縮小されない限り変更されない。この情報は、各ノードのシステムユニットが、system local space に同一のものを持つ。

4.2 GDS の構成

4.2.1 GDS の構造

GDS を構成するテーブルの配置例を図 5に示す。この図では、GDS を構成するテーブルが、3 つに分割された system global space に配置されている様子を表している。網かけの四角はポインタ、矢印はポインタによるリンクを表す。最外郭の、一点線で囲まれた領域は、同じシステムユニットをホームユニットとする空間である。

図のように、GDS は複数の小規模なテーブル群がポインタによるリンクで結合された一つの巨大な Directory を構成している。この Directory はミッションとユニットの間の帰属関係を軸としている。

例えば、Mission ID と Unit ID から、当該ユニットの位置情報を検索する場合、ミッション・マスターテーブル からユニット・レコードへのリンクを順に辿ることになる。

4.2.2 各テーブルの機能

各テーブルの機能は以下の通りである。

- mission master table
  ミッション・マスターテーブルとミッション・スレーブテーブルは、二階層構造を成し、共同で Mission ID の管理を行う。ミッション・マスターテーブルはシステム内に唯一つ存在し、システム起動時に作成される。
  ミッション・マスターテーブルはミッション・スレーブテーブルへのポインタの配列からなる。ミッション識別子 (Mission ID) を一定量 (例えば 128 個) のチャンクごとに管理するテーブルである。配列の要素が null である場合、そのチャンクに対応するミッション・スレーブテーブルはまだ生成されていないことを表している。
- mission slave table
  ミッションの生成要求の際に Mission ID の発行

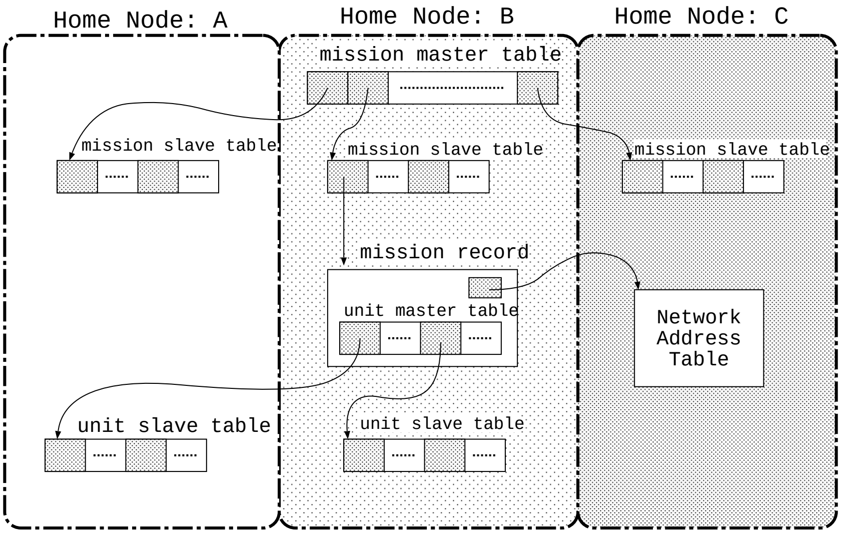

図5　GDSを構成するテーブルの配置例

を行う。ミッション・レコードへのポインタの配列である。ベースとなるID番号を持ち、配列のオフセットと加算してMission IDとなる。例えば、ベースIDが128、要素のオフセットが12であるならば、その要素にはMission IDが140であるミッション・レコードへのポインタが格納される。配列の要素がnullであるなら、その要素が表わすMission IDを持つミッションは作成されていないことを表す。

システムはミッションの生成時に、Mission IDを取得する際、当該システムユニットが動作しているスレーブテーブルを検索し、空きIDがあれば、それを利用する。もし、空きのIDがなければ、ミッション・マスターテーブルに要求を出し、新しいIDのチャンクをもらい、その中からIDの発行を行う。

- mission record
  ミッションに関する情報が格納された構造体である。Network Address Tableへのポインタ、ユニット・マスターテーブル、およびミッションが存在するノード情報などを含む。ミッションが展開されているノードが拡張/縮小する際に更新される。
- unit master table
  ユニット・マスターテーブルとユニット・スレーブテーブルは、二階層構造をなし、共同でUnit IDの管理を行う。各ミッションは唯一つのユニット・マスターテーブルを持つ。機能は基本的にミッション・マスターテーブルと同じである。
- unit slave table
  ユニットのノード番号を示す要素の配列である。ベースとなるID番号を持ち、配列のオフセットと加算してUnit IDとなる。構造および機能は基本的にミッション・スレーブテーブルと同じであり、発行するIDが異なるだけである。
- Network Address Table
  ミッション共有領域の各共有ページごとのホームユニットを格納しているテーブル。Mission IDと仮想アドレスをキーとして検索し、Unit IDを取得する。図5ではノードCに存在しているように描かれているが、実際には各領域に分散配置されている。

4.3　分　散　化

この図5ではsystem global spaceは3つの領域に区分けされており、一点線で囲まれた各領域として表現されている。

大域的な情報の管理を行うのに、集中管理をしていたのでは、性能上のボトルネックになりかねないため、何らかの形で分散化を図る必要がある。GDSの構造は、分散処理を意図して設計されている。

4.3.1　管理テーブルの階層化

管理テーブルを階層化し、上位階層へのアクセス頻度を軽減するとともに下位階層テーブルの変更が、唯一のシステムユニットによってのみ行われるようにすることで、共有データへの排他アクセスを極力排除する。

Mission IDの発行を例にとると、もし、Mission IDの発行をミッション・マスターテーブルのみで行った場合、ミッション生成の要求はミッション・マスターテーブルに集中することになる。これは非効率であり、システムの規模が大きくなればミッション・マスターテーブルがミッション生成の上でのボトルネックとなりかねない。

GDSではミッション生成時の新しいMission IDの

発行は、たいていの場合ミッション・スレーブテーブルが行い、手持ちのIDが不足したときのみミッション・マスターテーブルに新しいIDのチャンクを要求するので、処理の分散化が行われる。この際、処理がスレーブテーブルのみで完了する場合は、排他制御が不要である。

4.3.2　アクセスの分散化

管理テーブルの階層化により、個々のテーブルへのアクセスは、ある特定のシステムユニットに偏ることになる。このようなテーブルを、それを頻繁にアクセスするシステムユニットが存在するノードがホームノードとなる共有領域に配置することで、アクセスの分散化が図れるとともに、当該テーブルへのアクセスレイテンシを軽減することができる。

4.4　共有メモリを利用した実装に関する考察

4.4.1　共有メモリを利用する利点

GDSの管理テーブル群を、共有メモリ上に配置する実装を採用する有効性について考察する。

なお、ここではGDSが扱うデータについて以下のように前提を設ける。

前提1　ホームユニットは変更されない

あるホームページのホームユニットは、一度設定されると基本的には変更されない。変更されるのは、ユーザがシステムコール等により明示的に変更を指示した場合のみである。

前提2　ユニットの移送頻度は低い

ユニットの移送には、それなりのコストがかかるため、頻繁には発生しない。ユニットの移送は数msに一度程度の頻度であると予想される。

この実装の利点を以下に示す。

- NIのキャッシュの一貫性管理機能の利用
  データのキャッシングを行っている物理ノード等の動的な情報をNIがキャッシュブロック単位で保持しているため、NIのキャッシュ一貫性制御機構を利用することで、管理テーブルの細かい程度でのデータ更新も、低いオーバーヘッドで実現可能である。
- データの性質に起因する有効性の考察
  (1)　扱うデータ構造が比較的巨大
  各ノードで必要となる管理情報には空間的、時間的局所性があると考えられる。したがって、それぞれのノードが図5のテーブル群のフルコピーを持つのでは無駄が多く、ノードの主記憶を圧迫することになる。一方、共有メモリを利用する場合には局所性を利用し、必要な情報のみ主記憶上にキャッシングすることで、主記憶の利用効率を向上することができる。また、キャッシュヒット率を向上させるためにGDSが管理しているデータのキャッシュをディスクキャッシュ等と区別し、キャッシュの破棄などの点で優遇することが考えられる。
  (2)　更新される可能性はあるが、その頻度は小さく、かつ更新の規模は小さい
  大規模なテーブルの変更が発生するのはミッションやユニットの生成/終了時点に、ユニット・スレーブテーブルやNetwork Address Tableが変更されるくらいである。通常、ミッションが動作している間の変更は、ユニットに位置情報が変更される程度であり、わずかにユニット・スレーブテーブルまわりを数バイト程度変更するくらいである。
  (3)　参照頻度が更新頻度に比べて高い
  GDSの管理情報を最も頻繁に利用するのはNIであると考えられる。NIは、キャッシュブロックのミスヒットといったキャッシュのブロック単位の操作にGDSの情報を必要とするが、GDSの更新はミッション/ユニットの生成/終了/移送やページフォールト時であり、この時の参照範囲は、高々数キャッシュブロック程度であるため、一度キャッシュを行うと高い確率でそのデータが残っており、GDSへの参照がノード内で簡潔する可能性が高くなる。

4.4.2　ソフトウェアによる方式との比較

GDSと同等の機能は、ソフトウェアによるメッセージパッシングを用い、以下に述べるような方式で実現できる。ユニットの位置情報を例にとる。

ユニットが生成されたノードをそのユニットのホームとし、そのユニットの位置に関しては、そのノードが情報を把握する責任を持つ。ユニットが移送されても、ホームであるノードに問合せれば、そのユニットの位置がわかる。取得した情報はローカルノードにキャッシュされ、次回以降のアクセスにはこのキャッシュが使用される。キャッシュ・ディレクトリはホームノードにおいて管理される。

Coloniaでも基本的に、同じ流れでユニットのノード位置を取得する。すなわち、ローカルノードにデータがキャッシングされている場合はそれを使用し、キャッシングされていない場合には、当該ユニットへのリンクを持つミッション・スレーブテーブルにアクセスしノード情報を取得する。このミッション・スレーブテーブルは、基本的に当該ユニットが生成されたノードがホームユニットである領域に存在する。

二つの方式の異なる点は、キャッシュヒットの判定とキャッシュの一貫性制御方式である。

メッセージ・パッシングでは、ハッシュテーブル等を作成し、必要なデータがキャッシングされているかをソフトウェアで判定する。一方、Coloniaではこの判定をNIがハードウェアにより高速に行い、キャッシュ・ディレクトリの管理もNIが行う。この点では、

共有メモリを利用した実装が有利であるといえる。

しかし、Coloniaでは、キャッシュのヒット判定をハードウェアにより高速に行うことができるが、データの管理単位が固定長のブロック単位(32byte)であるため、キャッシュの無効化の際には無関係なキャッシュまで無効化してしまうことがある。逆に、メッセージ・パッシングでは、ソフトウェアによりデータのサイズによらない、柔軟なキャッシュの無効化を行うことができる。

我々は、以上のことを考察した上で共有メモリを利用する実装が有効であると考え、その方式を採用するが、実際にどちらの実装方式が有効であるのかを示すには、シミュレーション等を行う必要がある。それは今後の課題とする。

5. まとめ

本稿では、分散OS Coloniaの大域的ネーム・サービスの一つであるGDSを、共有メモリを利用して実装する手法について述べた。

GDSは、システム内に分散して存在する、全てのミッション、ユニットの位置を動的に把握する。GDSは、システムのノード間共有領域上に実装され、ユーザが使用する共有領域を実現するための情報を管理する。

GDSが管理するデータの性質およびColoniaの共有メモリの実現方式から、GDSを共有領域上に実装することは、有効であると考えられるが、シミュレーションなどにより、その具体的根拠を示す必要がある。これは今後の課題である。

6. 謝辞

メンター・グラフィックス・ジャパン株式会社には，Higher Education Programの一環として製品とサービスをご提供いただいたので，ここに感謝する．

本研究の一部は文部省科学研究費補助金（基盤研究(B)(2)#10558045，および，(C) #09680334）による．

参考文献

1) 山添博史, 他 :並列アプリケーションを指向した分散システムコンピュータ・コロニーの構想, 情処研報,97-OS-76,pp55-60,1997
2) Kai Li, Paul Hudak :Memory Coherence in Shared Virtual Memory Systems, In the ACM Transactions on Computer Systems, Vol7, No.4, November 1989, pp321-359
3) 山添博史, 他 :分散共有メモリ上のOSにおける高速移送可能な並列アクティビティ,(SWoPP下関)1997
4) 青木秀貴, 他 :共有メモリベースのシームレスな並列計算機環境を実現するオペレーティングシステムの構想, 情処研報,97-OS-34,1997
5) Jean Bacon(藤田昭平, 篠田陽一, 今泉貴史 訳) : 並行分散システム, オペレーティングシステム、データベース、分散マルチメディアシステムへの統合的アプローチ, トッパン(1996)
6) Curt Schimmel(岩本信一 訳, 前川守 監訳) : UNIXカーネルの内部解析, キャッシュとマルチプロセッサの管理,SOFT BANK(1996)
7) 前川守, 所真理雄, 清水謙太郎 編 :分散オペレーティングシステム,UNIXの次にくるもの, 共立出版社(1991)