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

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

増田 峰義

平成11年2月12日

内容梗概

我々は「コンピュータ・コロニー」を提唱し、分散システムの持つ計算機資源を有効に利用し、並列アプリケーションの効率的な実行を指向する計算機環境の構築を目指している。

コンピュータ・コロニーは、主に共有メモリをベースとした通信を行い、アプリケーションの細粒度並列実行を指向した計算機環境である。コンピュータ・コロニーでは、分散システムの欠点である通信時のレイテンシを軽減するため、専用の通信ハードウェアにより、高速・低遅延な通信を実現する。

分散OS Coloniaは、コンピュータ・コロニー上で動作し、分散している計算資源を統括するとともに、並列アプリケーションに適した共有メモリベースの計算機環境を提供する。Coloniaは、並列アプリケーションに適したプログラミングモデルである「ミッション・ユニットモデル」を提供する。ユーザがシステムに投入するジョブの単位はミッションといい、ミッションは並列実行可能な複数のユニットからなる。ユニットは独立なアドレス空間を持つアクティビティであり、同一ミッション内のユニットはアドレス空間を一部共有している。

本稿は、Coloniaにおける大域的ネーム・サービスについて述べたものである。大域的ネーム・サービスは、システム全域にわたる共有メモリを持つユーザレベルのミッションとして実装される。大域的ネーム・サービスはシステム内に存在するほぼ全てのオブジェクトの位置、およびその移動を管理する。

コンピュータ・コロニーの共有メモリ管理は、大域的ネーム・サービスが保持するホーム位置およびキャッシュディレクトリの情報を利用して、通信ボードが動作することで行われる。

本稿では、共有メモリを利用した大域的ネーム・サービスの実装方法を示すわけだが、具体的な実装を示す前に、まず大域的ネーム・サービスを共有メモリを利用して実装する有効性を示す。大域的ネーム・サービスで扱うデータは、共有メモリで扱うことに適した性質を持つ。すなわち、更新頻度が低く、また更新規模が小さいためキャッシュのヒット率が高く、たいていの場合ハードウェアによる動作のみでキャッシュを管理できる。したがって、分散システムに不可欠なシステム構成の柔軟性を保つためのソフトウェア部分があるにもかかわらず、大域的ネーム・サービスは高いパフォーマンスを発揮できる。

ただし、大域的ネーム・サービスが提供する機能の中でも、共有メモリ管理の情報を提供する大域的ネーム・サービス(以下GDS)は、共有メモリを利用して実装することはできない。なぜなら、共有メモリを実現するためのオブジェクトが、共有メモリを利用しているという自己矛盾に陥るからである。我々は、この問題を解決するために、GDSを共有メモリを利用して実装するためには特別な仕組を持つミッションとして実装する。すなわち、オリジナルのデータが存在するノードを静的に決定し、キャッシュディレクトリをそれぞれのノードのユニットが独自に管理することで、共有メモリのセマンティクスを実現する。これにより、共有メモリ管理に必要な高いパフォーマンスを持つオブジェクトを実現することができる。

本稿ではさらに、GDSの具体的な機能である、ミッションおよびユニットの識別子と、その位置情報の管理機能の実装について述べる。現在、この機能のプロトタイプ

を並列計算機 Cenju-4 上で実装中である。

Abstract

We propose "Computer Colony", a distributed system effectively use the computation machine resources which the distributed system has and aim to build the computer environment which points the efficient execution of parallel applications. Computer Colony mainly does the communication based on shared memory. Computer Colony is a computer environment where applications can run in fine-grained parallel. Computer Colony realize high speed and low latency communication network by the communication hardware of the exclusive use to reduce latency in communication which is the fault of the ditributed system.

"Colonia" is a distributed operating system, which work on computer colony hardware, unifies power of distributed resources, and realizes a suitable computer environment for parallel applications. Colonia suggests "mission-unit model", a programing model suited to parallel execution. "Mission" is a task thrown by users into a system, and "mission" is composed of several "unit" which is executable in parallel. "Unit" is an activity which has its own virtual memory space, and units in the same mission share a part of virtual memory space.

This paper descrives the global name service of Colonia. The global name service is a user-level mission which has a shared memory space over the system whole area. The global name service manages the location and migration of almost all objects in the system.

Sharing memory management of Computer Colony is done in communication board's working using the home position and cache directory which the global name service maintains.

This paper shows a way of implementation of the global name service which uses a shared memory, but before concrete implementation is shown we show first the effectiveness to the implementation of the global name service using shared memory. The data to handle by the global name service has the nature which suited to handle with the shared memory. That is to say, the hit percentage of the cache is high because the cache update frequency is low and also update scale is small, therefore we can manage cache data only by the operation by the the hardware in the most case. Therefore as for the global name service, high performance can be shown, though there is a software part to keep the flexibility of the system composition indispensable for distributed system.

But in the function which the global name service provides, the name service which provides the information of the shared mamory management(hereinafter, are GDS)can not be implemented using shared memory. Because it falls into the self contradiction that object realize a shared memory uses shared memory. We implement GDS as the mission which has special construction to solve this problem. That is to say, the original data fixes on a node statically, and the unit of each node manages cache directory personally, and we realizes semantics of the shared memory. By this, object which has high performance which is necessary to manage shared memory can be realized.

This paper also shows implementation of ID management of mission and unit and position information of them,which is a concrete function of GDS. Present we are implementing the prototype of this function on Cenju-4,which is a parallel computer.

目 次

1 はじめに

近年、ワークステーションやパーソナルコンピュータの高速化、小型化、低価格化により、それらの計算機をネットワークで結合した分散システムが広く普及している。しかし、現行の分散システムでは、これらの分散システムが持つ、トータルとしての膨大な計算能力を有効に活用できていない。そこで我々は、分散システムの持つ潜在的な計算能力を引き出すための計算機環境として、「コンピュータ・コロニー」を提唱する。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計算機環境であり、OSも含めた総合的なシステムである。コンピュータ・コロニーでは、従来の分散システムが、ジョブの効率的な並列実行を行えない原因であるマシン間の通信コストの高さを、通信専用のハードウェアであるColonia NICによって解決し、細粒度並列処理を行うことを可能にする。

「Colonia」は、コンピュータ・コロニー上で動作する分散オペレーティングシステムであり、コンピュータ・コロニーを構成する個々の計算機を有機的に協調動作させ、その計算資源を有効に利用することを目的とする。本稿は、分散OS Coloniaにおける大域的ネーム・サービスの役割、およびその設計について述べたものである。大域的ネーム・サービスは、システム内に存在するほぼ全てのオブジェクトの位置と、その移送の管理を行う。

従来の分散システムにおける大域的ネーム・サービスの役割は、ファイルシステムなどの分散資源の位置透過性を実現することにあった。Coloniaにおける大域的ネーム・サービスはこれに加えて、共有メモリ管理のためのホームの位置やキャッシュディレクトリなどの情報や、アクティビティの位置情報なども提供する。これらの管理情報はシステム全域にわたり、システムの規模が大きくなるにつれて巨大なデータベースとなる。Coloniaでは、この巨大な管理情報を配置しておく領域として、システム全体にわたる共有領域を構築する。

コンピュータ・コロニーでは、アプリケーションは基本的に共有メモリを利用した並列処理を行うため、共有メモリの管理コストは、システム全体の処理能力に関わる。したがって、大域的ネーム・サービスには、高いパフォーマンスが要求される。大域な管理情報を共有領域に配置して管理することは、データの性質とコンピュータ・コロニーのハードウェア環境を考えた上で、パフォーマンス面で有効である。

以下では、2章においてコンピュータ・コロニーのハードウェア構成、およびその上で動作する分散OS Coloniaの概略を述べ、さらに、その共有メモリ管理について述べる。3章では、大域的ネーム・サービスが提供する機能について述べ、その中でも共有メモリ管理を行うGDSについて述べる。さらに、大域的ネーム・サービスをユーザレベル・アクティビティとして実装し、その共有領域を利用することが有効であることを示す。次に、GDSの具体的な実装方法について述べる。

2 コンピュータ・コロニーの概要

2.1 コンピュータ・コロニー

「コロニー」とは群体を意味し、同種の生物個体が多数集まって互いに協調して生活する共同体のことである。コンピュータ・コロニーの実行形態は、この生物活動になぞらえている。すなわち、複数の独立した計算機がネットワークで結ばれ、互いに協

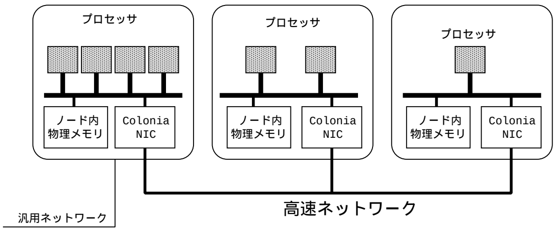

図 1: コンピュータ・コロニーのネットワーク構成

調動作することで、全体であたかも一つの計算機のように振る舞うわけである。コンピュータ・コロニーとは、このような計算機の共同体が提供する計算機環境であり、OS も含めた総合的なシステムである。

2.1.1 コンピュータ・コロニーの目標

コンピュータ・コロニーは、以下の目標を実現する計算機環境である。

- 並列アプリケーションの効率的な実行

  効率的な並列アプリケーション実行のための高速・低遅延なネットワークを構築し、さらに並列アプリケーションに適したプログラミングモデルを提供する。

- 計算機システム全体が持つ資源の有効利用

  特定のマシンへの負荷の偏りをなくし、負荷をシステム全体に均等に配分することで計算資源を有効に利用する。

- 低価格での実現可能性

  汎用マシンによる分散システム構築を可能にすることで、導入のためのコストを低く抑えることができる。

2.2 Colonia NIC

コンピュータ・コロニーのネットワーク構成の例を図 1 に示す。個々のノードは、独立で動作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。ネットワークを構成する各要素マシンは、通信用専用ハードウェアを介して、高速・低遅延な通信を行う。この通信用専用ハードウェアはそれぞれのノードのシステムバスに接続されており、これにより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。

本稿ではこの通信用専用ハードウェアを Colonia NIC と呼ぶ。

Colonia NIC 上には、プロトコルプロセッサが搭載されており、ハードウェアでは実装困難な機能を、ソフトウェアにより実現する。また、Colonia NIC 上にはプロセッサが利用するための 2 次キャッシュも搭載されており、自ノードのメインメモリの内容をキャッシングするのに使用される。

Colonia NIC が提供する 2 つの重要な機能は、高速・低遅延な通信機構、およびキャッシュコヒーレンス制御である。

2.2.1 高速・低遅延な通信機構

一般に、従来の分散システムにおいて、マシン間通信のコストが高くなる要因として、以下の 2 つが大きい。

① 通信資源保護のためのシステムコールによるオーバーヘッド

② プロトコル処理にかかるオーバーヘッド

具体的には以下のようにして、これを実現する。

- **MMU のメモリ保護機能を利用**

  通常ではオペレーティングシステムが行なう通信資源の保護を、MMU(Memory Management Unit:メモリ保護ユニット) のメモリ保護機能を利用して実現する。Colonia NIC はシステムバスに接続されており、ローカルマシンの物理アドレスとリモートマシンの主記憶を対応づける。アクセスは仮想アドレスで行われ、MMU により物理アドレスに変換される。Colonia NIC は、このメモリアクセス命令発生時にバスに流れる物理アドレスを監視し、必要に応じて通信を行う。通信には、MMU による仮想アドレスから物理アドレスへの変換が含まれるため、Colonia NIC は、通常の主記憶と同じ形でハードウェアのみによって保護される。

  したがって、システムコールを発行することなく、高速かつ安全な通信を行うことができる。

- オーバーヘッドの小さいプロトコルの開発

  TCP/IP のような汎用プロトコルではなく、コンピュータ・コロニーにおける通信が持つ特徴に適したプロトコルを開発することで、オーバーヘッドの小さいプロトコルを実現する。

2.2.2　キャッシュ・コヒーレンス制御

Colonia NIC は単にリモートアクセスを行うだけでなく、ブロック単位 (32byte) でのキャッシュコヒーレンスの一貫性を保つ。Colonia NIC は、メインメモリからキャッシュの管理情報をボード上にキャッシングし、その内容に従い、キャッシュのコヒーレンス制御およびデータの転送を行う。Colonia NIC がコヒーレンス制御を行うことで、キャッシュ制御を高速に行うことができ、同時に通信量も削減することができる。

2.2.3　物理ページの種類

Colonia において、物理ページは以下のような種類がある。

ホームページ 物理メモリが実装されている物理アドレスに位置し、メモリ上にデータの実体が存在する。ノード間共有の際には、他のノードにホームページが存在している。

ホームページが存在しているノードを特に「ホームノード」と呼ぶ。

ダミーページ 物理メモリが実装されていない物理アドレスに位置する。

ダミーページに対するアクセスが発生すると、通信ボードがこれを検出し、対応するホームページにアクセスすべくノード間通信を行う。

コピーページ 物理メモリが実装されている物理アドレスに位置し、ホームページの内容をキャッシングしているページである。無効化されている場合には、通信ボードの動作はダミーページと場合と同じになる。

Colonia NIC は、コピーページまたはダミーページを指す物理アドレスを検出し、ホームページに対してリモートアクセスを行い、データを転送する。また、共有しているページのブロックに対する更新が発生した場合には、そのページの全てのコピーページに対して無効化信号を送る。

2.3　分散 OS Colonia

2.3.1　Colonia の機能

- 共有メモリ実現の支援... 物理的にメモリを共有していないノード間で、共有メモリを実現する。その上で、任意のアクティビティ間でのメモリ空間の共有を可能にする。
- 並列アプリケーションの効率的な実行... 並列アプリケーションを効率的に実行するために、並列実行に適したプログラミングモデルである「ミッション・ユニットモデル」を提供する。また、特定のノードに負荷が集中しないように、アクティビティの移送を行い、動的に負荷を分散させる。
- 位置透過性... 分散システムが持つ物理的に分散した資源を位置透過に管理する機能は、従来の分散 OS においても基本的なものである。

2.3.2　Colonia のシステム構成

Colonia はマルチスレッド・カーネル方式の構成をとり、システムコール処理中の割り込みを可能にすることで、効率的なスケジューリングを行う。Colonia カーネルはごく基本的な機能のみを提供し、それ以外の大半の機能はユーザレベルのミッションにより提供される。そのため、設計、作成および保守が容易になり、また柔軟な構成が可能となる。

2.4　Colonia における共有メモリ

2.4.1　ミッション・ユニットモデル

複数のプロセッサを使用して一つの仕事を処理するためには、仕事を並列実行可能な単位に細分化する必要がある。

ミッション・ユニットモデルは、共有メモリを利用可能な分散システムにおいて、並列プログラムの効率よい実行を実現するためのモデルである。

ミッション ミッションとは、ユーザがシステム内に投入する仕事の単位である。一つのアプリケーションプログラムあるいは一つのコマンドが一つのミッションに相当する。ミッションは並列実行可能な複数個のユニットからなり、それらのユニットが協調動作することでジョブが進められる。

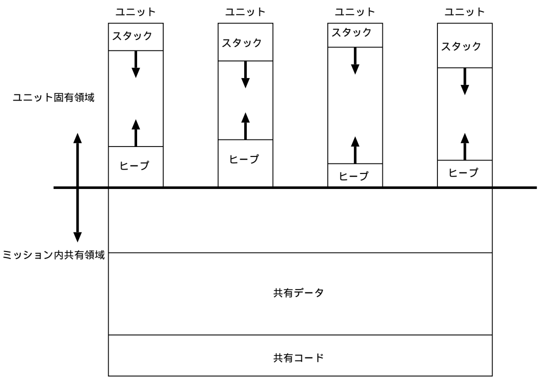

図 2: ユニットのアドレス空間

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、ディスパッチの対象となる単位である。ユニットが別々のプロセッサ上で動作することで、ミッションは並列に実行され、実行効率が向上する。ミッション生成時にまず一つ生成され、その後は必要に応じて追加生成される。

ユニットは固有のアドレス空間を持ったアクティビティであるが、ひとつのミッションに属するすべてのユニットは、ミッション生成時に指定されたアドレス空間の固定領域を、アドレス位置も含めて共有する。ユニットのアドレス空間については次節で詳しく述べる。

スケジューラは、特定のノードにユニットが集中しないよう、負荷の偏りを是正する。特定のノードの負荷が高くなってしまった場合、スケジューラはいくつかのユニットを他のノードに移送するよう指示を出す。ミッション・ユニットモデルでは、ユニットの移送に伴うオーバーヘッドを小さくし、移送によるコストを下げることを目的とした設計がなされている。

2.4.2 ユニットのアドレス空間

ミッション・ユニットモデルでは、アクティビティであるユニットがそれぞれの固有のアドレス空間を有している。ユニットのアドレス空間は、二つの領域を持つ。ひとつはユニットに固有のデータを配置しておくユニット固有領域であり、もうひとつは同一ミッションに属するすべてのユニットで共有される、ミッション内共有領域である。ユニットのアドレス空間を図 2に示す。

- ミッション内共有空間

  ミッション内共有空間は、ミッションの生成時に指定される。ミッション内で生成されたすべてのユニットは、そのミッション内共有空間を自身のアドレス空間の同じアドレス位置にマッピングする。したがって、このメモリ領域内に対するポインタは、どのユニットにおいても同じ内容を指すことになる。この共有領域は、コードやデータの共有に利用される。

- ユニット固有領域

  ユニット固有領域は、ユニットが完全に自由に利用できる領域であり、スタックや、ユニットに固有なデータ空間として利用される。ユニット固有領域は他ユニットから完全に保護される。

Colonia の提供する共有メモリで最も基本となるのが、ミッション内共有領域によるミッション内ユニット間の共有メモリである。この共有メモリは、コードやデータの共有に利用される。コードを共有するため、ユニットの高速な移送や生成が可能となる。

3 分散 OS Colonia への実装

ネーム・サービスは、オペレーティングシステムの中心的役割の一つである。とりわけ、多数のさまざま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、アクセス可能なオブジェクトに大域的な名前を与えたり、それらを識別したりするための大域的ネーム・サービスは不可欠である。

本章では、Colonia における大域的ネーム・サービスが提供する機能、および共有メモリを利用した、その実装方法について述べる。

3.1 大域的ネーム・サービスの機能

Colonia における大域的ネーム・サービスが管理する情報には、以下に挙げるものがあり、共有メモリ管理に関わるものと、それ以外のものとに分けることができる。

① 共有メモリ管理のための情報を提供する機能

(a) アクティビティのノード位置、およびアクティビティの識別子管理

具体的には、ミッションおよびユニットが持つ、それぞれを一意に特定するための識別子の発行と管理を行う。また、アクティビティの位置管理も行う。つまり、ミッション (ユニット) 識別子を指定することで、そのミッション (ユニット) が存在しているノード位置を特定するためのテーブルを作成し、ユニット間通信などの際にはユニットの位置情報を提供する。

識別子は、それを用いてアクティビティを一意に特定できる必要があるため、大域的に管理する必要がある。

アクティビティの位置情報は、任意のノードからの要求に応えなければならないため、これも大域的に管理することが望ましい。

(b) ホームページおよびキャッシュディレクトリ

異なるノードのユニット間でページを共有するために、ページ同士の対応関係を保持するテーブルを作成し、共有しているページのホームページを記録しておく。

また、ホームページとコピーページの間のコヒーレンスを保つために、コピーページが存在しているノードをキャッシュディレクトリとして記録しておく。

② その他の大域的な情報を提供する機能

(a) ユーザの固有の情報

例えばユーザのログイン情報のような、各ユーザに対する固有な情報も大域的に管理される。

(b) ファイルシステム

ファイルの位置透過性の実現のため、ファイルシステムにおけるネーム･サービスも、大域的ネーム･サービスによって管理される。

(c) マシン環境に関する情報

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。

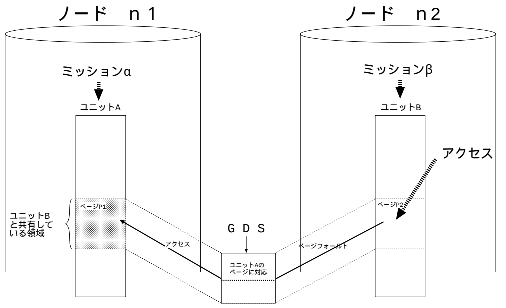

図 3: 共有ページの対応づけ

(d) スケジューラのための情報

並列アプリケーションを効率的に実行するために、各ノードの負荷状況を提供し、動的な負荷分散を行うための支援を行なう。

以下の記述では、特に 1 の共有メモリ管理のための情報を提供する機能の実装について述べる。本稿では、この機構を GDS(Global Directory Service) と呼ぶ。

3.2 GDS

Colonia は、物理的にメモリを共有していないページ間で、共有メモリのセマンティックスを仮想的に実現する。これを実現するためには、システム内に分散して存在するページの位置を管理し、共有しているページ同士の対応づけを行う機構が不可欠である。つまり、たとえば共有しているページへのアクセスで、ページフォールトが発生した場合に、そのページの共有先がどこに存在するかという情報が必要となる。Colonia において、GDS がそのような役割を果たす。

また、Colonia では、異ノード間で共有しているページのキャッシングを行うことで、転送の手間を省き速度の向上を行う。ここで、ページ間のコヒーレンスの一貫性を保つために、ページがキャッシングされているノードを把握しておく必要がある。この、共有ページがキャッシングされているノードの管理も GDS が行う。

3.2.1 共有ページの対応づけ

ページの対応づけ

Colonia におけるメモリの共有は、ユニットのページを単位として行われる。つまり、

図 3 を例にとると、GDS は「ミッションβに属するユニット B のページ P2 のホームページは、ミッションα に属するユニット A のページ P1」と記録することで、ページの共有が実現される。ただ、Colonia におけるページ共有は、同一ミッションに属するユニットが持つ、共有領域でのページ共有が大半であるため、ミッションα =β であり、ページ P1=P2 であることがほとんどである。

実際の共有ページへのアクセスは、以下のように行われる。

- ホームページの特定

  GDS は識別子と参照ページの組から、そのページのホームページを持つユニットと、そのページ番号を一意に決定するテーブルを持つ。

  識別子とページ番号 (図3では、ミッション識別子β とユニット識別子 B とページ番号 P1) ともとにして、このテーブルをひき、ホームページを持つユニット (ミッションα のユニット A) と、そのページ (ページ番号 P2) を特定する。

  このテーブルのエントリは、共有設定されている全てのページに対して存在する。

- ユニットの位置の特定

  上記のテーブルにより特定したユニットが存在しているノードの位置 (図 3では ユニット A が存在するノード n1) を特定する。そのために GDS は、システム内に存在する、全てのユニットおよびミッションのノード位置を管理するテーブルを保持する。

ホームページを持つユニットとそのページ番号、およびそのユニットが存在しているノード位置から、ホームページの物理的な位置がわかる。Colonia NIC は、これらの情報を使いリモートアクセスを行い、実際にデータの転送を行う。

テーブルの内容の変更

GDS が管理するテーブルの内容を変更するユーザレベルでのイベントは、直接 GDS に伝えられず、Colonia カーネルが GDS とユーザの間の仲介役として動作する。この様子を図 4に示す。テーブルの変更を伴うユーザレベルでのイベントには、以下のようなものがある。

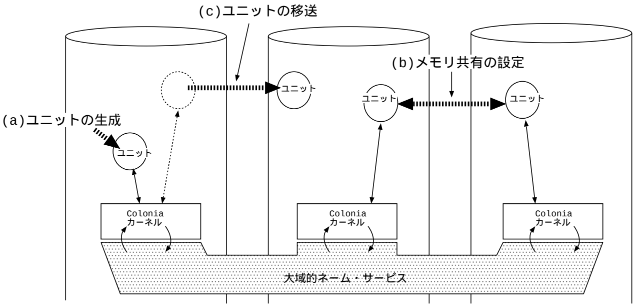

図 4: GDS と Colonia カーネル

(a) ミッション ・ユニットの生成

  ユーザはミッションまたはユニットの生成を Colonia カーネルに要求し、Colonia カーネルは、そのための処理の一環として、GDS に新しいミッション (ユニット) のためのエントリを登録する。

  同一ミッション間での共有領域、つまり図 3においてα =β である場合は、共有ページの設定は自動的になされ、ユーザは明示的にページの共有設定を要求する必要はない。

(b) 異ミッション間のページ共有

  異なるミッション間でのページ共有 (つまり図 4でα ≠ β の場合) を行うためには、ユーザが Colonia カーネルに対して明示的にページの共有を要求し、Colonia カーネルは GDS を呼び出し、対応関係の登録処理を行わせる。

(c) ユニットの移送

  ユニットが移送された場合、その位置情報は更新される。しかし、そのユニットが持つホームページは即座に移送されず、そのページへの参照が発生したときに初めて移送先のノードに転送される。

3.2.2 コヒーレンス制御

異なるノード間での共有メモリのセマンティクスを実現するためには、各ノードから常に最新のデータにアクセスできなければならない。そのために、すべてのホームページに対して、そのコピーページが存在しているノードを記録したキャッシュディレクトリを用意する。ページの更新が発生した場合、キャッシュディレクトリの内容にしたがい、Colonia NIC がコピーページが

存在するノードにデータの無効化信号を送り、コヒーレンスを保つ。ただし、Colonia NIC が行うコヒーレンス制御はブロック単位 (32byte) であり、ページ全体の無効化は行わない。

3.3 共有メモリを利用した大域的ネーム・サービス

一般に、大域的ネーム・サービスが扱うデータはシステム全域にわたり、その規模はシステムが巨大であればあるほど大きくなる。さらに、データの性質として外部のインターフェイスに対して、管理するデータが位置透過であることが望まれる場合が多い。そこで、Colonia における大域的ネーム・サービスはミッションとして生成する。扱うデータは、ミッションの共有領域に配置される。

本節では、大域的なデータを共有領域に配置してその管理を行うことが、コンピュータ・コロニーにおいては適当であることを示す。そのために、まず共有メモリを設計する上で、性能におよぼす影響の大きいキャッシュ制御機構の、実装の違いによる特質について述べる。次いで、大域的ネーム・サービスが管理するデータの性質について述べ、その性質がコンピュータ・コロニーにおけるキャッシュ制御機構との親和性が高いことを示す。

3.3.1 キャッシュ制御機構の実装方式による比較

キャッシュ制御機構の役割は、データ転送用バッファ領域間のコヒーレンス制御およびデータ転送である。キャッシュ制御機構には様々な実装が考えられるが、ここでは特に、全てをハードウェアにより実装した場合と、全てをソフトウェアにより実装した場合をとりあげ、それぞれの方式を比較することで、キャッシュ制御機構において着目すべき点を明らかにする。

ソフトウェアによる方式 ソフトウェアによるキャッシュ管理の利点は、柔軟な構成が可能な点である。例えば、キャッシュサイズをデータの種類に応じて差を設けたり、様々なデータの転送方式を実装することが可能である。また、システムの規模の拡張などにも柔軟に適応することができ、その意味では分散システムに適している。逆に、キャッシュを管理するためのコストが大きくなり、パフォーマンスの上で問題が残る。

ハードウェアによる方式 例えば CC-NUMA などように、キャッシュの管理を全てハードウェアにより行う方式である。ホームの位置およびキャッシュディレクトリの管理が自動的に行われるため、オーバーヘッドが低くパフォーマンスが高い。

反面、キャッシュサイズや、データの転送方式などの点で柔軟な構成をとることができない。したがって、システムの拡張性の面で柔軟性に欠ける。

以上の比較により、キャッシュ制御において、パフォーマンスと柔軟性はトレードオフの関係にあるといえる。どちらの方式が優れているかは、システムを使用する環境によるところが大きい。つまり、キャッシュの制御をどのような方式で行うかを決定するための重要な要素となるのは、対象としているデータ自身の性質である、と考える。そこで、次節では大域的なネーム・サービスが扱う情報の種類を列挙し、それらのデータとしての性質について述べる。

3.3.2 大域的ネーム・サービスが管理する情報の性質

3.1節において、Colonia における大域的ネーム・サービスが管理する情報を示した。これらの管理情報は、以下に挙げるような共通の性質を持つ。

- データ構造が巨大である

  データの構造がシステム全体にわたるため、それぞれのノードごとにコピーを持たせることはできない。

- 更新される可能性はあるが、その頻度は低く、かつ更新の規模は小さい

  例えば、ファイルシステムのようなデータは更新される可能性はあるが、その頻度は低いことが予想される。また、ユニットの移送などのイベントによる、ユニット位置の情報の更新なども、数 ms に一回くらいの割合でしか発生しないと思われる。

- 更新よりも参照の頻度の方がずっと高い

  更新の頻度が低いため、キャッシュされたデータに対するヒット率が高くなる。

- 局所性がある

  情報へのアクセスはランダムに発生するのではなく、時間的・空間的な局所性があると予想される。例えば、同一ファイルへの参照や更新は同じミッションに属するユニットによる可能性が高い。

3.3.3 共有メモリと、大域的なデータの性質との相性

本項では前項において示した、大域的ネーム・サービスが管理するデータの性質が、Colonia における共有メモリの構成方式に適したものであることを示す。

Colonia NIC のデータ転送パターン

Colonia NIC 間の通信により、リモートノードに存在しているデータを自ノードへ転送するための手順は、リモートノードでのデータの状態によって、以下のように大分される。

1. ホームページ上にデータブロックが存在し、なおかつ状態が **Valid**(有効) である場合。

   データの状態が有効であるため、通信先の通信ボードは、メインプロセッサの手を借りることなくデータを転送させることができる。つまり、データの転送はハードウェアにより自動的になされ、転送にかかるコストは低く抑えられる。

2. ホームページ上にデータブロックは存在するが、状態が **Invalid**(無効) である場合

   ホームページのデータが無効であるということは、すなわちキャッシュ先でデータが更新されたということである。この場合、更新されたデータがキャッシュ先のノードに存在しており、そのデータを転送しなければならない。そこで、キャッシュ先のノードを特定するために、通信ボード上のプロトコルプロセッサは、大域的ネーム・サービスが保持しているキャッシュディレクトリのデータをメインメモリから取得し、キャッシュ先のノードを特定しデータの転送を行う。

   この場合、プロトコルプロセッサが動作する分、1 の場合に比べて若干データ転送時のコストが高くなる。

3. ホームページがページアウトされている場合

   対象としているページがスワップアウトされており、ホームノードの物理メモリ上に存在しないため、ページをスワップ先から物理メモリ上にもってくる必要がある。リモートノードのメインプロセッサ上でカーネルが起動され、スワップ先からデータを物理メモリ上に転送してくる。スワップ先からのデータの読み込み、およびコンテキストの切換えなどに時間がかかるため、データの転送にかかるコストは非常に高いものになる。

データの性質との親和性の考察

前節において示した大域的ネーム・サービスが管理する情報の性質上、Colonia NIC の転送パターンは先の 1 のパターンがほとんどであり、2 と 3 のパターンの頻度は非常に小さいものと考えられる。その根拠をデータの性質にもとづいて述べる。

- 更新頻度が低い

  データが更新される頻度が低いため、キャッシュされたデータが無効化されることが少ない。したがって、キャッシュヒット率が非常に高くなることが予想される。また、ホームノードでのデータの状態が Invalid である可能性も同様に低くなることが予想されるため、データの転送パターンは1 のパターンが大半であると考えられる。

- 更新よりも参照の頻度の方がずっと高く、また参照の局所性がある

  データに対するアクセスは、参照が更新に比べて頻度が高く、またそのアクセスパターンには時間的および空間的局所性がある。したがって、アクセスされるデータは、たいての場合ホームノードの物理メモリ上に存在している、と予想される。つまり、3 のパターンによりデータが転送される頻度を低く抑えることになる。

以上の考察により、Colonia において大域的ネーム・サービスを共有メモリを利用して実現した場合、データの参照はたいていの場合キャッシュによりまかなうことができ、リモートノードからデータを転送する必要

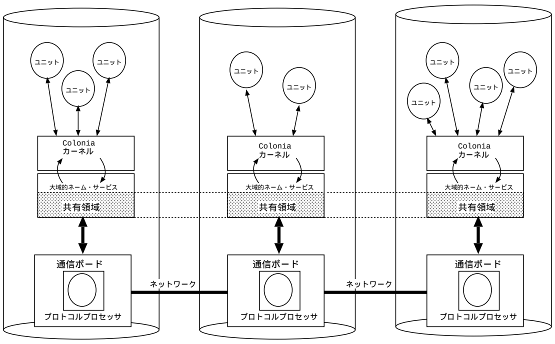

図 5: 大域的ネーム・サービスのシステム内での構成

が生じても、それにかかるコストは非常に小さくて済む。したがって、分散システムに必要な柔軟性を確保しつつ、高パフォーマンスを維持することができる。つまり、大域的ネーム・サービスを共有メモリを使用して実現することは有効である。

3.4 GDS のミッション・ユニットモデルによる実現

Colonia の大域的ネーム・サービスは、共有メモリ領域を利用するためにミッション・ユニットモデルにより実装し、ユニット間の共有領域にデータを配置する。

本節では、まず大域的ネーム・サービスが提供する、すべての機能に共通なミッション・ユニットの構成について述べる。しかし、GDS は通常のミッションとして作成することはできないので、3.4.2項におてGDS の特別な実装方法を示す。

3.4.1 ミッション・ユニットの構成

大域的ネーム・サービスをミッション・ユニットモデルにより実装する目的は、システム全体にわたる共有領域を構築し、位置透過性を実現することにある。大域的ネーム・サービスのユニットは、システムを構成している全てのノードごとに一つ存在し、共有領域を構成する。ユニットは全てのノードに存在するため、これによりシステム全体にわたる共有領域が構成されることになり、どのノードからもこの領域にアクセスすることができる。ネーム・サービスは、この領域に大域的な情報を配置し、その管理を行う。

ユーザからの要求、たとえばあるファイルを開きたい場合やユニットの位置を特定したい場合、ユーザは Colonia カーネルに要求を出し Colonia カーネルはネーム・サービスを呼ぶ。ネーム・サービスは自身の共有領域に配置されているテーブルを使い要求を処理する。要求がファイルを開くことであれば、ユーザはファイルが存在しているノード位置を知る必要がない。すなわち、位置透過性を実現することができる。

大域的ネーム・サービスのユニットがシステム全体に構築されている様子を図 5に示す。

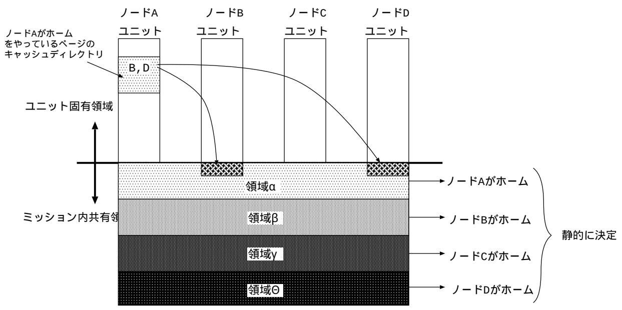

図 6: 共有領域管理ミッションの構造

3.4.2 ホームノードの静的な設定

大域的ネーム・サービスのうち、GDS は通常のミッション・ユニットモデルにより実装することはできない。なぜなら、通常のミッションに対して、ユニット間のページ共有を実現するために必要な

- ホームページが存在しているノード位置
- コピーページの位置を記録しているキャッシュディレクトリ

の二つの情報を与えるのが自分自身であるため、自分自身の共有領域のページに対して、その対応関係を管理しているオブジェクトが存在せず、共有ページのセマンティクスを実現することができないからである。

そこで、GDS は通常のミッションではなく、特別な構成をとることにする。すなわち、外部から情報を与えられることなく、自前で共有メモリのセマンティクスを実現する。

具体的には、GDS は以下の点で通常のミッションとは異なる構成をとる。

- 共有領域のホームノードの静的な割り付け

  共有領域のページのホームノードを静的に決定しておき、ホームノードを仮想アドレスから静的に決定できるようにする。そのためにまず、共有領域をページの集合として適当な大きさに切り分け、それぞれの領域に対しホームノードを静的に割り付ける。同一の領域に属するページは、同じノードをホームノードとする。

  Colonia NIC は、仮想アドレスからホームノードを決定する。GDS はシステム内で唯一の存在であり、そのユニットもノード内でただ一つのみ存在するため、ホームページを持つユニットを特定することが可能である。対象となるユニットが特定できれば、Colonia NIC がデータを転送してくる処理は通常のミッションと同じである。

- それぞれのノード毎のユニットがキャッシュディレクトリを保持する

  キャッシュディレクトリは、各ノードに存在するネーム・サービスのユニットが固有領域に配置し、その管理を行う。各ユニットは、自ノードがホームノードを担当しているページごとに, キャッシュディレクトリを作成する。

  自ノードがホームであるページのデータが更新された場合には、この情報が通信ボードにより参照される。

このミッションの構造を図 6に示す。

図では、共有領域を 4 つの領域α ~θ に分割し、それぞれノード A ~ D をホームノードとして割当てている。

3.4.3 ホームノードの変更

このミッションにおいて、あるノードで新しく管理情報が生成されると、そのノードがホームを担当しているアドレス領域に、そのデータは配置される。データが生成された直後は、そのデータに頻繁にアクセスがあるノードとホームノードは、一致している可能性が高く、なんら不都合は生じない。しかし、時間の経過とともに、ユニットの移送などの要因により、そのデータへのアクセスが頻繁に発生するノードと、ホームノードとの物理的な距離が離れてしまう状況が発生する可能性がある。この場合、キャッシュ先でのデータ更新が起こる度に、物理的に離れたホームノードまで無効化信号を送らなければならず、無駄である。

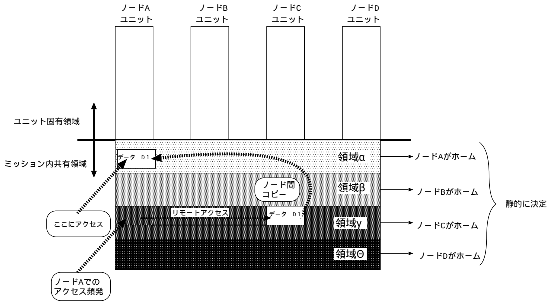

図 7: GDS におけるホームノードの変更

この状況は、データをアクセスが頻繁に発生するノードがホームノードである領域に、データをコピーしてやることで解消できる。この様子を図 7に示す。この図では、はじめデータ D1 はノード C で生成され、ノード C がホームを担当している領域γ に配置されていた。しかし、時間の経過に伴い D1 に頻繁に変更がノード A のユニットから加えられるようになったため、データ D1 をノード A がホームを担当している領域α にコピーする。こうすることで変更が頻発するノードとホームノードが一致するため、無駄な通信を削減できる。

管理情報をノードからノードへコピーするには、そのデータに関わるすべてのテーブルを変更しなければならないため、メンテナンスの面でのコストが大きい。そこで、今後の課題として、できるかぎりメンテナンスのコストを抑えて、ホームノードの変更ができるような機構が求められる。

3.5 ミッションおよびユニットに関する情報管理テーブルの実装

本節では、この GDS が提供する機能の内、ミッションとユニットの識別子管理、および位置情報の管理の実装について述べる。

3.5.1 実装の方針

実装の方針は、集中管理をできるだけ避け、アクセスによる負荷を分散させることである。管理機能の分散を行う利点を以下に挙げる。

- パフォーマンス

  集中管理では、特定のノードに負荷がかかり、システムの規模が大きくなるにつれ、パフォーマンスの低下を招く。これは、分散システムの特徴である拡張性の高さを損なう結果となる。したがって、情報は分散して管理することが望ましい。

- 排他制御

  共有メモリ上のデータの変更は排他的に行われなければならない。分散して管理を行うことで同一テーブルを同時に更新しようとする可能性が低くなり、メモリロック等による『待ち』の影響を受ける部分が小さくて済む。

- 耐故障性

  管理情報が集中しているノードに、何らかのトラブルが発生した場合、全システムにおいて実行が不可能になる。分散管理を行うことで全システムが実行不能に陥いる可能性は低くなる。

3.5.2 管理機構の全体図

実際の実装について述べる前に、ミッションおよびユニットの動作について、いくつかの仮定を設ける。識別子および位置情報の管理は、この仮定にもとづいている実装されている。

- ユニット間の通信は、ほとんどが同一ミッション内でのものであり、異なるミッション間での通信の頻度は低い。
- ユニットは生成されたノード近辺に存在することが多く、また同一ミッション下のユニットは互いに近傍に存在する。

以上の仮定に基づく管理機構の全体の構成を図 8に示す。

識別子は、マスターテーブルとスレーブテーブルにより二元的に管理され、管理のための機能は分散される。カーネルからの要求の大半を、スレーブテーブルのみの参照で処理し、マスターテーブルへの参照を極力抑えることを目的とする。

マスターテーブル、スレーブテーブルおよびレコードの機能は以下の通りである。

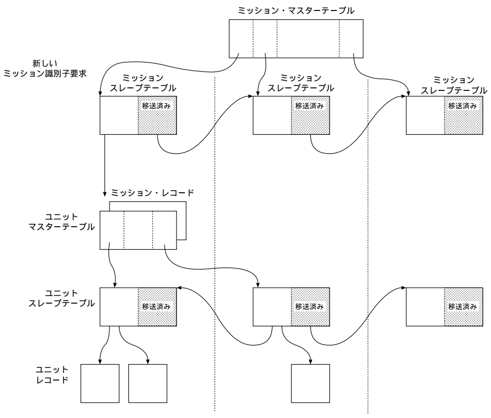

図 8: 管理機構の全体図

- マスターテーブル

  マスターテーブルは識別子を一定量 (例えば 128) ごとに管理するテーブルである。スレーブテーブルから空き識別子の要求があれば、識別子のセットを与え、どのスレーブテーブルにそのセットを与えたかを記録しておく。したがってマスターテーブルは、スレーブテーブルが空き識別子を使い果し新しい識別子のセットを要求してきたときのみ更新される。そのため、マスターテーブルの内容がキャッシュされた場合、そのデータへのヒット率は高くなる。

  ミッション マスターテーブルはシステム内でただ一つ存在し、ユニット マスターテーブルはミッションごとに存在する。

- スレーブテーブル

  マスターテーブルから一定量の空き識別子を与えられており、その管理を行う。手持ちの空き識別子に関しては、自由な裁量権があり、ミッション (またはユニット) が生成されたときに、手持ちの空き識別子の中から自由に識別子を与えることができる。手持ちの空き識別子が不足した場合は、マスターテーブルに新しい空き識別子を要求する。どのミッション (またはユニット) に識別子を与えたかを記録しておく。

  また、スレーブテーブルはミッション (ユニット) のレコードの位置を記録する。レコードが別の領域に移動された場合には、移動先のスレーブテーブルの位置を記録しておく。レコード位置を取得したい際には、仮にそのレコードが移送されていてもリンクを辿ることで目的の

レコードを取得することができる。したがって、マスターテーブルの参照頻度が現象しマスターテーブルに対する負荷を軽減することができる。

- レコード

  新しくミッション (ユニット) が生成されると、レコードが作成される。レコードにはミッション (ユニット) ごとの位置情報などが記録される。

管理機構をこのような構成にすることで、識別子の発行の要求は大半の場合スレーブテーブルにより処理され、マスターテーブルの更新は発生しない。

また、ユニット間の通信においても、以下のことがいえ管理を分散できている。すなわち、先のミッションとユニットの動作に関する仮定より、異なるミッションとの通信のために、その位置を取得する機会はまれであり、なおかつミッションの生成は人間活動のレベルでしか発生しないため、ミッション・マスターテーブルへの参照、更新の頻度は低い。また、同一ミッションに属するユニットは、ユニット・マスターテーブル周辺に存在していることが多いので、ユニット間の通信もスレーブテーブルのみの参照でよい場合が多い。

4 おわりに

本稿では、分散 OS Colonia における大域的ネーム・サービスが提供する機能、およびその実装について述べた。

大域的ネーム・サービスは、システム内に存在する多様なオブジェクトの位置管理を行なう、Colonia における最も重要な機構の一つである。

大域的ネーム・サービスが提供する管理情報の内、GDS はアプリケーションの実行効率に関わるため即答性が要求される。その要求に応えるために、コンピュータ・コロニーのハードウェアと、扱うデータの特性を考え併せた結果、ユーザレベル・ミッションとして大域的ネーム・サービスを実装し、その共有領域にデータを配置して管理を行うことは有効である、と結論づける。

大域的ネーム・サービスは、共有メモリを利用するために、共有領域のページのホームノードが静的に決定されている、特別なミッションとして構成される。この構成では、アクティビティの移送に伴い、生成時におけるデータのホームノードの位置と、現状で最も望ましいホームノード位置との間に齟齬が生じる可能性があり、適宜ホームノードを変更するためのノード間データコピーが必要になる。しかし、このようなコピーはコストが高く、できるかぎり避けたい。そこで、常に最適なノードがホームノードとなるような機構を考案することが今後の課題となる。また、本稿では述べなかった、ファイルシステムなどの大域的データを管理するネーム・サービスの実装も、今後の課題である。

謝辞

本研究の機会を与えて頂いた、本研究室の富田眞治教授に深甚な謝意を表します。

また、本研究に関して数々のご指導を頂いた、森眞一郎助教授、五島正裕助手に深く感謝の意を表します。

最後に、本研究の共同研究者である山添博史氏、伊達新哉氏、Damien Le Moal 氏、鶴崎剛大氏、杉野秀和氏をはじめ、日頃から様々な角度からご助力頂いた、京都大学大学院情報学研究科通信情報システム専攻富田研究室の諸兄に心より感謝致します。