特別研究報告書

分散 OS Colonia における
ホームページ移送機構の実装

指導教官　　富田 眞治 教授

京都大学工学部情報学科

鳥居 大祐

2000 年 2 月 13 日

分散 OS Colonia におけるホームページ移送機構の実装

鳥居 大祐

内容梗概

コンピュータ・コロニーでは、単体でも動作可能な計算機をネットワークで結ぶことにより、全体として 1 つの大型計算機としての性能を発揮できる分散計算機システムを開発している。現在、学校や職場で利用されている LAN で結んだ分散システムとは違い、各計算機の持つ資源を並列処理により有効利用し、理論上 1 つの大型並列計算機と同等の能力を持つようなシステムを目指している。

各ユーザは、今までの分散システムと同様に、1 つの計算機として端末にログインし、システム内のどの端末からログインしても同じように計算機を利用することができる。この時、ユーザはシステムが複数の端末からできていて、自分が投入したジョブが複数の端末で並列動作していることを意識することなく、利用することができる。さらに、コンピュータ・コロニーでは、複数のユーザが公平に資源を利用できるようなフェア・シェアスケジューリングを行う。また、コンピュータ・コロニーはプロセッサ、主記憶、ネットワークインターフェースをモジュール化することにより、個々の端末の計算機資源を自由に増やすことが容易になり、同時に低価格性を実現する。

このようなシステムの実現のため、コンピュータ・コロニーでは、各端末を高速なネットワークで結び、各端末同士が緊密に協調動作できる環境を整える。そのために、本研究室では、ハードウェアと分散オペレーティングシステム Colonia の両方から設計を行っている。

各端末同士が緊密に協調動作するためには、通信機構が重要な役割を果たす。コロニーではハードウェアによるキャッシュコヒーレンス制御を行う分散共有メモリベースの通信を行う。

各ノードは複数のプロセッサと主記憶装置、各ノードを緊密に結び合うためのネットワーク・インターフェースからなる。それぞれは、システムバスにより結ばれるが、特にネットワーク・インターフェースも同様にシステムバスにより結ばれることで、リモートメモリのキャッシングなどで生じるコヒーレンス制御をハードウェアにより高速に行うことが可能となる。

Colonia では分散共有メモリ空間を実現する方法として、Shared Virtual Mem-

ory を採用している。すべての仮想共有メモリにはホームページと呼ばれるデータの実体が物理アドレス上に存在する。各仮想ページがすべてホームページにマッピングされることで、仮想的なメモリ共有を実現している。

ホームページを参照するには、ホームページが存在するノードに通信を行う必要があるが、アクセスレイテンシ、通信トラフィックの軽減のために、各ノードではホームページを自ノードの主記憶にキャッシングする。このようなキャッシングされたページをコピーページと呼ぶが、これらのコヒーレンス制御のために、ホームページを有するノード（ホームノード）ではキャッシュディレクトリなどの管理情報をもつ。各ノードのコピーページに対する処理の際、リモートアクセスの必要性が生じた場合、各ノードのネットワーク・インターフェースは、ホームノードへアクセスする必要がある。このように、ホームノードへは多くのアクセスがあることが予想され、ホームページをどこのノードに配置するかにより、通信や各ノードの処理の効率に影響を及ぼす。

Colonia では、ホームページの配置を適宜変更するホームページ移送を行い、常に最適なノードに配置することを目指している。このようなホームページ移送の際には、GDS の管理する表の書き換えとキャッシュディレクトリの転送が主な作業になる。最も問題となる、キャッシュディレクトリ転送中にやってくるパケットの処理は、キャッシュディレクトリを転送する際のパケットの往来が少ないと考えられる環境のもとで最適と考えられる方法である、パケットを送信元ノードに送り返す方法を選択した。

ホームページ移送は元ホームノードと新ホームノード間のパケットのやり取りにより処理が進行していく。実装に際しては、ホームページ移送処理が高速に行われるために、ホームページ移送プログラムを NIC のプロトコルプロセッサ上で動作させ、さらにパケットの送受信をなるべく割り込みを少なくし、ポーリングで対処するようにした。また、適宜、専用のハードウェアを設けることでも全体の高速化を図った。

最後に本実装に対する考察をした後、今後の研究の展望や関連研究について述べる。

The implementation of homepage migration mechenism for distributed operating system Colonia

Daisuke Torii

Abstract

We are developing a distributed computer system "Computer Colony". By connecting computers on network, the total system will work as one big computer power.Nowadays, distributed systems are used in companies or schools by connecting computers on LAN. Apart from this,our system uses each computer resource efficiently by executing process in parallel, and theoretically working as one big parallel computer.

In our system, users access to one of computers, and can use the same computer resources wherever computer they are accessing to. They use this system, without recognizing that this system is consist of many computers.Also,we make a scheduling system for distributing fairly this computer resource.Finally,we make a module card in which processors, memory, and network interface are placed,so that users can increase the computer resource freely with ease, and we make this system with low cost performance.

To realize such a system, which we call Computer Colony, we connect each computer with high performance network, and make each computer communicate tightly with other computers. To realize this environment, we approach the structure of hardware system and a distributed operating system "Colonia".

The communication mechenismis important for each computer to communicate tightly. On Computer Colony, each computer communicates by distributed shared memory, and the cache-coherence is controlled by the hardware system.

Each node is consisted of more than one processor, memory, and network interface. Each part is connected with the system bus, but special network interface card is connected to the same system bus, so that it makes possible to execute cache-coherence control speedy by the hardware system. The cache-coherence control is necessary in caching the remoted memory.

To realize the distributed shared memory space, we have adopted the concept of shared virtual memory. In all shared virtual memory, there is a page called "homepage" at physical memory. Each virtual memory being mapped to a

homepage, we realize the virtual shared memory.

To refer to a homepage, it is necessary to communicate with the node where the homepage exists, but to decrease access latency and communication traffic, each node caches a part of the homepage to the main memory of each node. We call such a cached page "copypage". For the cache-coherence control of these copypages, there exists the information to control cache-coherece at the node where homepage is located(we call this node "homenode"). At each node, when accessing a copypage,if the necessity of remote access is neccessary, the network interface of each node needs to access to the homenode.In this way, many accesses to a homenode are expected and the efficiency of communication and execution of program at each node will depend on which node the homepage is located.

Colonia aims to change the placement of homepage appropriately and allocate to the best node. We call this "homepage migration". In such migration of the homepage, The main process will be the rewriting of the table managed by GDS, and the migration of the cachedirectory.The main problem, the treatment of the packet which comes to the homenode is implemented by returning the packet to the sending node.We consider this implementation is the best if there are a few packets on network when we execute the homepage migration.

The execution of homepage migration is proceeding with the communication of packets between the old homenode and the new homenode. About the implementation, to execute the homepage migration speedy, we make the program of the homepage migration execute on the protcol processor of the network interface card, and when receaving or sending of packets, we try to use the polling more than the interrupt. Moreover, to finish the migration speedy, we implement the hardware only for the execution.

Finally, I consider this implementarion, and notice the view of this reseach, and the relational reseach.

分散 OS Colonia におけるホームページ移送機構の実装

目次

第 1 章　はじめに

本研究では、独立した計算機の集合から、1 つの高性能計算機としての性能を実現するコンピュータ・コロニーの開発を行っている。一般的に良く利用されている各端末をネットワークで結んだ分散環境とは違い、すべての計算機資源をトータルなものとしてとらえ、状況により計算機資源を有効に利用する。そのためには分散している端末同士を強力なネットワークで結び、各ノード同士が緊密に協調動作することが求められる。

コンピュータ・コロニーを実現するため、高速通信を支援するハードウェアと、その通信機構を効率的に利用できる分散 OS Colonia の両方からアプローチをおこなっている。

Colonia が動作するハードウェア環境の実現例として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している。これは処理ノード間を共有メモリという通信プロトコルで結合したクラスタシステムである。Colonia では物理的に共有メモリ空間を持たないので、仮想的な共有メモリ空間を実現する方法として、Shared Virtual Memory を採用している。

全ての仮想共有ページには、いずれかのノードにホームページとよぶ物理ページが存在し、仮想的な共有をするために、各仮想ページはホームページにマッピングされている。各仮想ページはホームページのデータを自ノードにキャッシングすることができるが、これらのキャッシングデータの一貫性制御のためにホームページを有するノードでは管理責任がある。したがって各仮想ページは、有効なデータの取得や、他ノードのデータを無効化するといった処理をする際ホームページにアクセスする必要がある。このように、ホームページを持つノードには多くのアクセスがあることが予測され、ホームページをどのノードに配置するかは各種処理の効率に影響を及ぼす。

本論文では、ホームページの配置を適宜変更し、常に最適なノードを行うためのホームページの移送機構について述べる。

以下、2 章ではコンピュータ・コロニーのあるハードウェア環境とその上で動作する分散 OS Colonia について述べる。3 章ではホームページ移送機構について述べ、4 章ではその実装法とその考察を行う。

第 2 章　コンピュータ・コロニー

本章ではコンピュータ・コロニーが目指す目標、また分散メモリ環境を実現するためのハードウェア環境、プログラミングモデル、分散メモリシステムについて述べる。

2.1　コンピュータ・コロニーの目標

コンピュータ・コロニーは、分散システムを発展させたマルチユーザ環境である。その上で、今日の分散システムの欠点である通信コストの大きさを克服し、並列アプリケーションの実行能力を実用レベルまで高めることを目指すものである。分散システムの持つ導入の手軽さや優れた拡張性といった利点を生かしつつ、従来の分散システムでは困難とされていた並列処理における高性能を目指す。また、プロセッサ、メモリ、ネットワークインターフェースを搭載したモジュールを開発することにより、システム構成の変更が容易に行え、同時に低価格性を実現する。

コンピュータ・コロニーではさらに、OS の働きによって計算機資源の位置透過性と高性能な負荷分散の実現を目指す。ユーザがコンピュータ・コロニーに投入したプログラムは処理ノードを一切意識する必要がなくなる。また、システムを利用している各ユーザに課金状況に応じたコンピュータ資源の割り当てをするフェア シェアスケジューリングを行う。

2.2　ハードウェア環境

コンピュータ コロニーのシステムモデルとして、i-Cartridge と Rack によるモデルを示す。

2.2.1　ハードウェア環境構成例

コンピュータ・コロニーのハードウェア環境の構成例を図 1 に示す。個々のノードは i-Cartridge と呼ぶカートリッジである。各 i-Cartridge は複数のプロセッサとキャッシュ、主記憶装置、通信専用ハードウェアから成り立つ。この通信専用ハードウェアを NIC(Network Interface Card) と呼ぶ。この i-Cartridge は単体でもコンピュータとして動作可能なものである。

プロセッサ、主記憶装置、NIC はお互いシステムバスに接続されている。主記憶は、ローカルなものだけではなく、リモートノード（他のノード）の主記

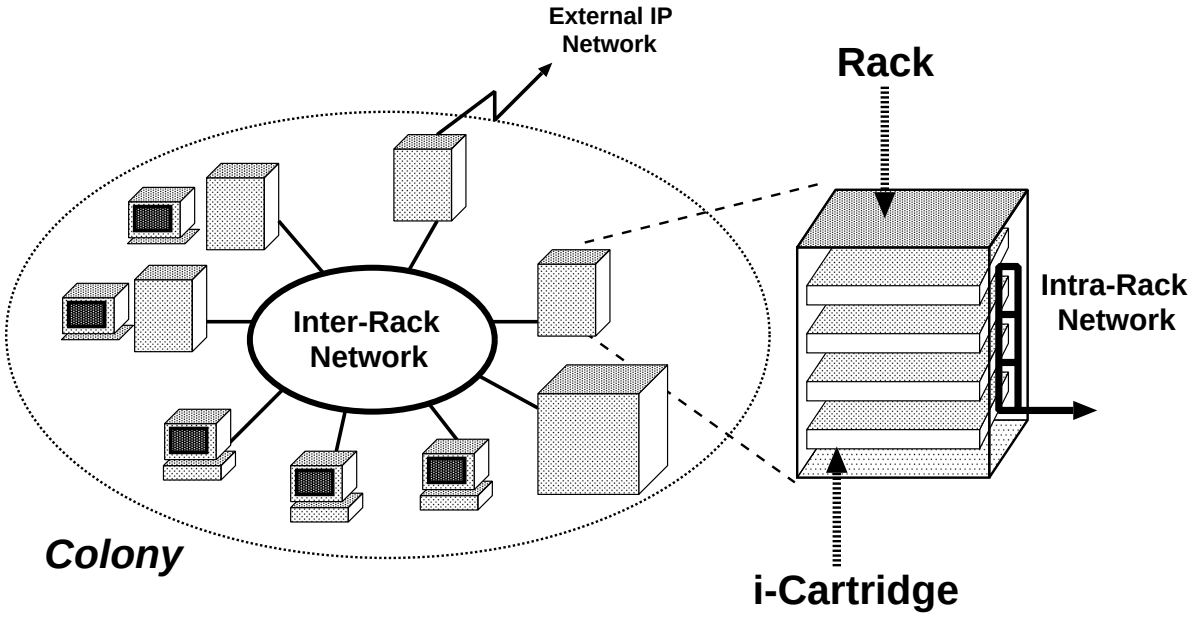

図 1: コロニーのハードウェア環境構成例

憶へのアクセスに対するキャッシュとしても使用する。NIC は、高速、低遅延な通信を提供する専用のハードウェアで、共有メモリベースの管理コントローラとして動作する。

i-Cartridge は Rack に収容される。Rack には、i-Cartridge を収容する Bay, 収容する i-Cartridge 間を結ぶ Inter-Rack Network、システム間を結ぶ Inter-Rack Network へのインターフェース、その他の I/O スロット、電源装置を内蔵する。Inter-Rack Network で接続されたシステム全体を Colony という。

2.2.2 NIC(Network Interface Card)

NIC は共有メモリ環境を実現するために重要な役割を果たす。

NIC がシステムバスに接続することにより、リモートメモリのキャッシングなどのコヒーレンス制御をハードウェアにより高速に行うことができる。

NIC には、システムバスとの通信をコントロールするインターフェース・コントローラがあり、常にシステムバス上の主記憶上へのアクセスをスヌープし、リモートアクセスの必要性を判断する。

コロニーの通信機構では、システムバスに接続した NIC により、ブロック単位のキャッシュコヒーレンス制御を行う。このために、NIC に主記憶のブロック単位の状態を示した主記憶タグ (B_Map) を持つ。NIC 内のインターフェース・コントローラはこの主記憶タグにアクセスを行い、リモートアクセスの必要性

を判断する。

このようなリモートアクセスの際に、大域アドレスへの変換を行う必要があるが、このようなアドレス変換に必要なテーブル類の管理は、NIC 上にあるプロトコルプロセッサが行う。また、プロトコルプロセッサはハードウェアでは実装できない様々な処理を行う。NIC は、テーブル管理の他に、さまざまなネットワーク通信の制御を行う。

2.3 ミッション ユニットモデル

ミッション・ユニットモデルは、分散共有メモリ環境の中で、並列プログラミングを効率良く実行するために Colonia が提供するプログラミングモデルである。

ミッション・ユニットモデルではプログラムをひとつのミッションとする。その中に複数のユニットを並列に記述する。ユニット内は、ミッション内共有領域とユニット固有領域に別れている。

これにより、スレッドがタスクの持つアドレス空間を完全に共有するタスク・スレッド方式とは違い、不要なデータ共有による性能低下やユニットのデータを破壊されるなどの危険を回避でき、効率良いプログラミングモデルを提供できるのである。

2.3.1 ミッションとユニット

ミッション ユーザがコンピュータ内で立てる代理であり、システムに投入された一連の仕事をおこなうプログラムである。ユーザはシステムにミッションを複数投入することができる。ミッションは協調動作する複数のユニットよりなる。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。

ユニット ミッションを並列実行可能な単位に細分化したものがユニットである。同一ミッションに属する各ユニットは、異なるノードに対して割り付けられ、ノード内のプロセッサで実行される。このようにしてユニットは並列に実行される。ユニットには、同一ミッション内で一意に識別可能な Unit ID が付与される。

2.3.2 アドレス空間

ミッション・ユニットモデルでは、ユニットがそれぞれ固有の仮想アドレス空間を有している。そのアドレス空間には、同一ミッション内のユニットと共

有するミッション内共有領域と、各ユニットで独自に持つユニット固有領域がある [1]。

ミッション内共有領域　ミッション内共有領域は、すべてのユニットにおいて同じ仮想アドレスに配置される。よって、どのユニットでも同じポインタで同じ内容を指すことができる。ここでは、共有データやテキストなどが配置される。

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、他ユニットから完全に保護される。ここでは、そのユニットのスタックなどが配置される。

2.4　Colonia におけるメモリ共有

ユーザは仮想アドレスを用いてメモリアクセスを行う。メモリアクセスがあるとプロセッサに付属する MMU(Memory Manager Unit) がページテーブルを参照して仮想アドレスを物理アドレスに変換し、物理メモリにアクセスがなされる。

同一ノード内のみのメモリ共有は、同一の物理ページに複数の仮想ページを割り当てることで実現される。

異なるノード間でのメモリ共有には、ネットワークを通じてアクセスする必要がある。Colonia ではこのような共有メモリ空間を実現する方法として、Shared Virtual Memory[2] を採用している。

2.3.2で述べたように、ミッション内共有領域には同じ仮想アドレスが割り当てられる。すべての仮想共有ページにはホームページと呼ばれるデータの実体があるノードの物理メモリ上に一つ存在する。ミッション共有領域上の各仮想ページが全て、NIC の働きにより、ホームページにマッピングされる仮想的なメモリ共有を実現している。

ホームページが同じノード内に存在しない場合、NIC によってホームページのあるノードにアクセスする。ホームページへのアクセスが何回も起こる場合、アクセスレイテンシ、通信トラフィックの軽減のため、ホームページを同ノード内の主記憶にキャッシングする。このキャッシングされたページをコピーページと呼ぶ。

このようなコピーページの内容は一貫性制御の対象であり、そのためのキャッシュディレクトリなどの情報はホームページが存在するノードに置かれ、管理

される。この一貫性制御は、ネットワークインターフェースが自律的にリモートアクセスの必要性を検知し、通信を開始することにより行われる。

このように、コンピュータ•コロニーではコピーページのメモリ領域の確保は、ページ単位で行われる。しかし、リモートアクセスの際のデータ転送はブロック単位で行われる。これにより、共有単位をページ単位とした場合のfalse-sharingや、データ単位とした場合のキャッシュディレクトリの容量の増大の問題を解決することができる。

各仮想共有ページのホームページには、ホームとなるユニットが一つ対応づけられ、そのページのホームユニットという。また、ホームページが存在するノードをホームノードという。

第 3 章　共有メモリ管理機構

これまでで述べてきたように、共有メモリ環境はNICの働きによって実現される。コピーページにおいて、無効なブロックにアクセスした場合、NICは、一旦ホームノードにアクセスし、キャッシュディレクトリを引いて、有効なデータを保持しているノードへアクセスする必要がある。また、ホームノードを特定する際、NICはGDSが管理する表を引く必要がある。この章では、これら管理情報である、キャッシュディレクトリとGDSについて説明した後、これらが、実際にどのように使用されるかについて述べる。

3.1　キャッシュディレクトリ

前章で述べたように、ホームページの内容をキャッシュしたコピーページが存在する場合、これらのキャッシュ内容は一貫性制御する必要がある。あるユニットがコピーページ内の無効なブロックにアクセスを行うと、NICはそれを検知して有効なデータの場所をホームノードに問い合わせる。ホームノードではキャッシュディレクトリという有効なデータがどこのノードに存在するかを示す表をブロック毎に持っていて有効なデータの場所を知ることができる。図2にキャッシュディレクトリの例を示す。

ホームページに有効なデータのノードを問い合わせたノードを「Initiatorノード」、有効なデータを保有していたノードを「ownerノード」と呼ぶ。

また、キャッシュディレクトリは、ユニット内で共有するページに対しての

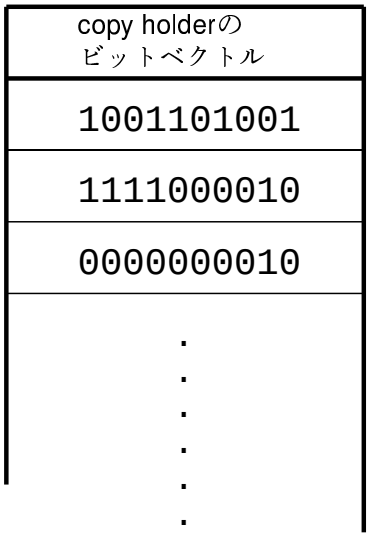

図 2: キャッシュディレクトリの例

み動的に生成されるため、必要なページのキャッシュディレクトリを検索するために、キャッシュディレクトリ検索用のテーブルも用意する。

3.2 GDS(Global Directory Service)

Colonia の分散仮想共有メモリ環境では、仮想共有ページとホームノードの対応を示したテーブルが必要であり、これらのテーブルを管理する機構が必要となる。

Colonia ではこのテーブルの管理を含めた、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を行う。その中でも、分散共有メモリを実現するための大域的な情報を管理を行う Global Directory Service(以下 GDS)[3] について述べる。GDS は以下の機能を果たす。

ネーミング　ミッション・ユニットに大域的な識別子を与える。

位置情報管理　システム内に分散する全てのミッション・ユニットの位置情報を把握する。

ホームユニット情報の管理と割当　各共有ページのホームユニットは、システム内で唯一であるので、この情報を管理する。具体的には、大域仮想アドレスとホームユニットの対応表を保持、管理する。また、ホームユニットの割り当を行う。

ホームノード情報の管理　ホームページの存在するノードを示す情報を管理

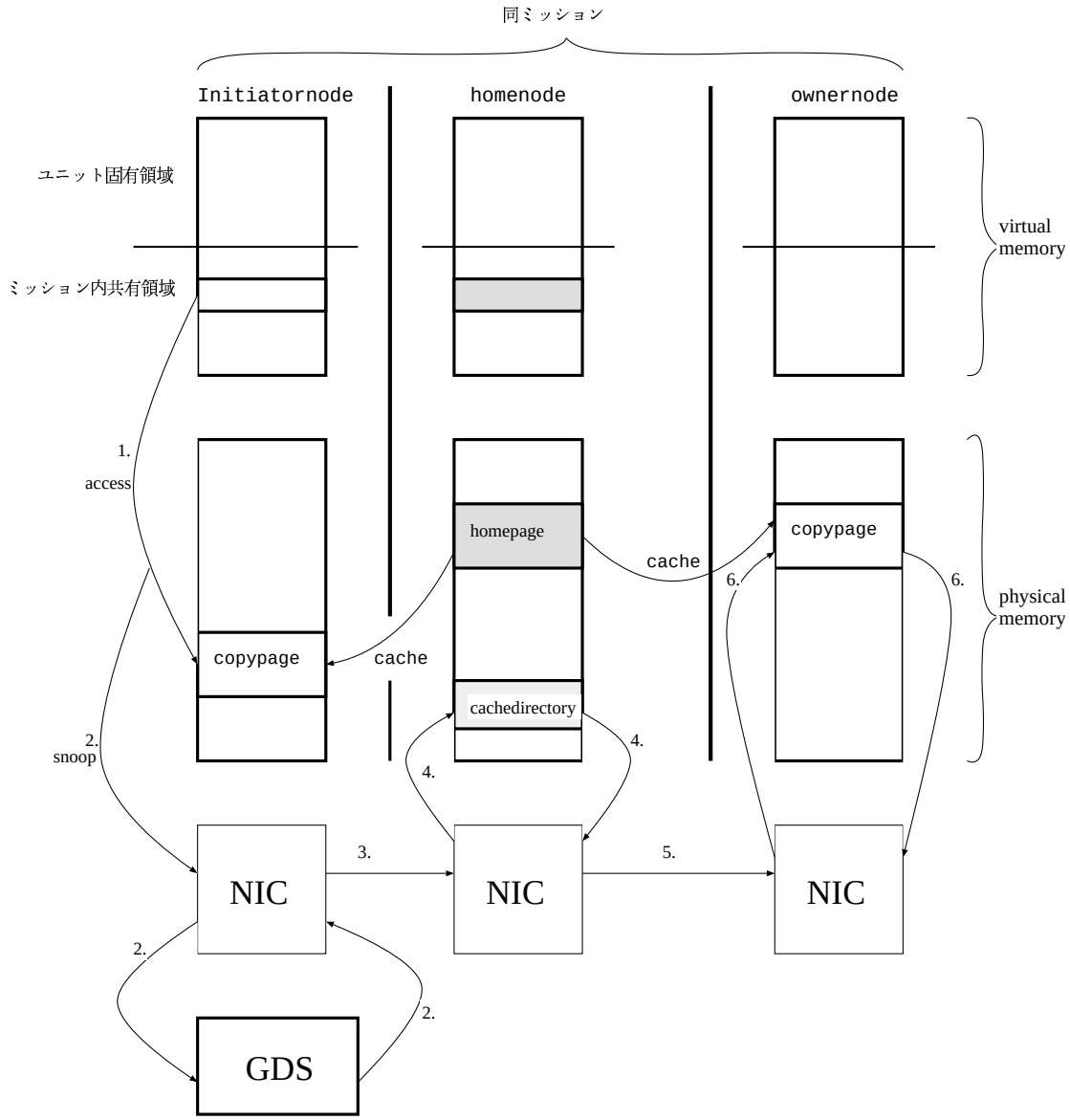

図 3: ネットワークを介したメモリアクセスの流れ

する。

以上で分散メモリ管理における管理情報について説明したが、次にこの管理情報が実際に分散メモリ環境においてどのように使われるのかについて述べる。

3.3 分散共有メモリ管理

ここでは、Colonia におけるネットワークを介したメモリアクセスの方法をキャッシュディレクトリ、GDS の働きを含めて説明する。

ネットワークを介したメモリアクセスの必要性は様々な場面（データ転送、データ無効化など）で発生しうるが、ここではコピーページを持つノードで有効なデータを保有していなかったために、有効なデータの転送をownerノードから行う場合について説明する。

1. ローカルノード（このノードをInitiatorノードと呼ぶ）では常に物理メモリへのアクセスはNICにより監視されており、アクセスしようとするブロックに対応するB_mapを参照することにより、データの有効性を判定している。これにより、アクセスしようとするブロックが無効と判定された場合、ネットワークを介したメモリアクセスの必要性が生じる。
2. その場合、NICは物理メモリへのアクセスを中断させ、一旦、変換された物理アドレスを逆引きページテーブルを用いて、再度仮想アドレスに変換する。次にこの仮想アドレスにMission IDを組み合わせることにより、大域仮想アドレスを生成する。
3. 大域仮想アドレスをキーにGDSが管理する大域仮想アドレスとホームユニットの対応テーブルを引き、Unit IDを得る。さらにUnit IDをキーにGDSが管理するホームユニットとホームノードの対応テーブルを引きNode IDを得る。
4. これにより、InitiatorノードのNICはホームノードへアクセスすることができ、該当するブロックのデータ転送要求パケットをホームノードのNICへ送る。
5. データ転送要求パケットを受け取ったホームノードのNICでは、そのページに関するキャッシュディレクトリを引き、有効なデータを保持しているノード（このノードをownerノードと呼ぶ）を調べる。
6. ホームノードのNICは該当するブロックのデータ転送要求パケットをownerノードのNICへ送る。
7. データ転送パケットを受け取ったNICは、Mission IDと仮想アドレスからページテーブルを参照し、有効データがあるブロックへアクセスする。そして、このデータをInitiatorノードへ転送する。

以上がネットワークを介したリモートアクセスの流れであるが、5.において[ホームノード＝ownerノード]であった場合は、ホームノードにおいて7.の手続きが行われる。以上の流れの様子を図3に示す。また、その際、利用するテーブルを図4に整理して示す。

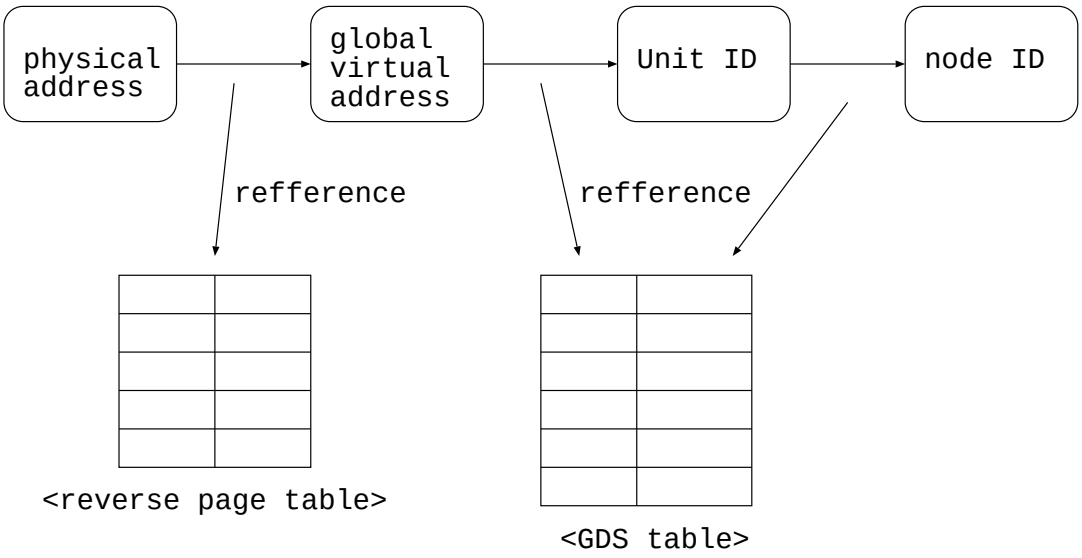

図 4: リモートアクセス時に利用するテーブル

第 4 章　ホームページ移送機構の実装

前章で述べたように、リモートメモリアクセスが必要と判断された場合は、ホームノードにおいて管理情報を参照する必要がある。このため、コピーページを持つノードに比べ、ホームノードへのアクセスは多く行われる。従ってホームページをどこのノードに配置するかは、ユニットが行う各種処理への効率や通信ネットワークの効率に影響を及ぼす。Colonia ではホームページの配置を適宜変更するホームページ移送を行い、常に最も各種処理の効率が高くなるよう最適なノードへ配置する。

本章では、ホームページ移送機構について説明した後、その実装法を述べ、最後にその考察を行う。

4.1　ホームページ移送

ここでは、ホームページ移送が行われるにあたって、そのタイミングや実行に関する様々な取り決めについて述べる。

4.1.1　ホームページ移送のタイミング

ホームページ移送が行われるタイミングについて、以下の 2 点を想定している。

- ユーザが明示的に移送のタイミングを示す。

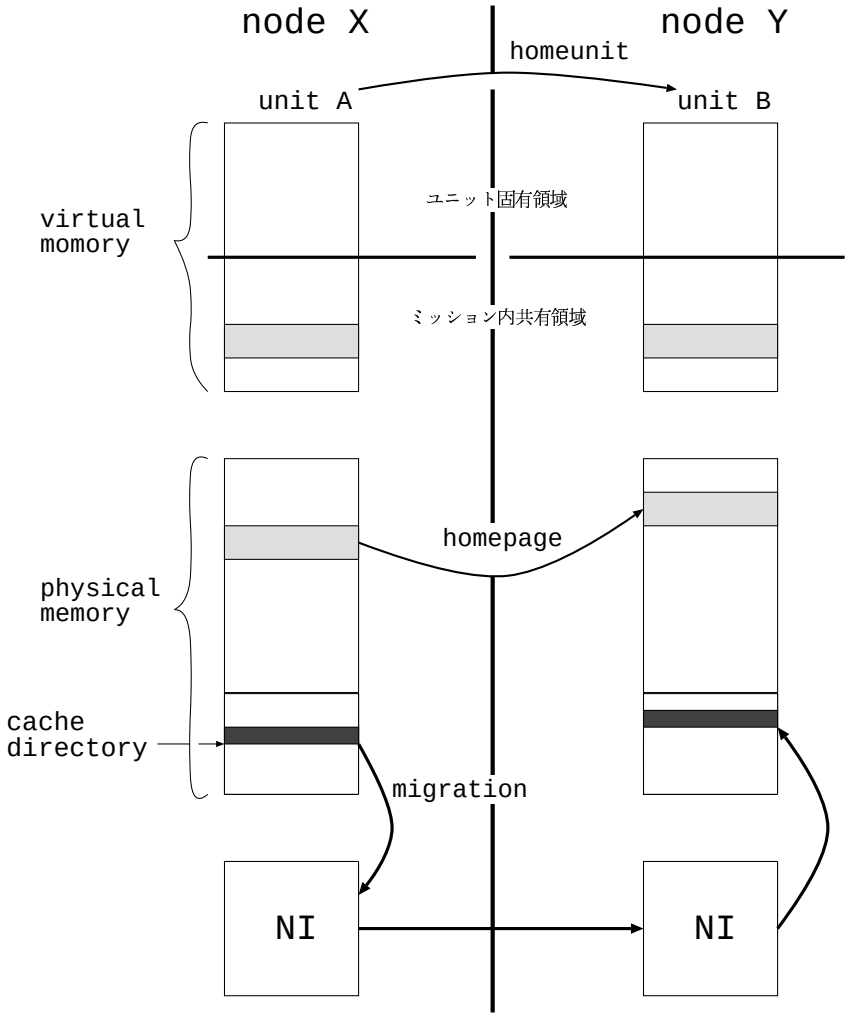

図 5: ホームページ移送の様子

- OS の判断により、移送を行う。

ユーザが移送のタイミングを示す場合は、例えば、アプリケーションの実行フェーズの移り変わる時期に、適切なホームページの位置（ノード）がユーザによって示される場合である。

また、OS の判断により行われる場合は、あるノードからホームノードに対して、多くのリクエストパケットが発生している場合に、当該リクエストを行っているノードにホームページを移してやることで、無駄なノード間のパケットの往来を減らすことができる。

以上のいずれの場合においても、移送が発生する場合はノード間のパケットの往来が少ないと想定される。

4.1.2 ホームページ移送に伴い変更すべき管理情報

各ノードの NIC は GDS が管理する大域仮想アドレスと Unit ID のテーブルを参照する (図4参照) ことにより、ホームページの位置を認識する。したがって、GDS が管理するこのテーブルの内容を書き換えてやることで、各ノードの NIC はホームページのノード位置が変わったと認識するわけである。しかし、ホームページではキャッシュ一貫性制御のためにキャッシュディレクトリを保持する必要があるので、この書き換えと同時に、キャッシュディレクトリを新しいノードへ転送しなくてはならない。

即ち、ホームページ移送はキャッシュディレクトリの転送と、GDS が管理する表の変更によって行われる。これらの作業により、ホームページが移送され、それに伴い、ホームユニットとホームノードの位置も変更されるのである。この様子を図5に示す。

4.1.3 ホームページ移送に伴い考慮すべき事項

以下にホームページ移送に伴い考慮すべき事項を挙げる。

1. キャッシュディレクトリへのアクセスは転送中には行うことができないので、その間に発行されたパケットに対し、何らかの処理を施す必要がある。また、その処理に対する後処理をしなければならない。
2. ホームページ移送処理終了以前に発行されたパケットは、ホームページ移送終了後も元ホームノードへやってくる。これらネットワーク上で処理中のパケットを新しいホームノードへ向かわせる必要がある。このためには、すべての NIC に新ホームノードを速やかに知らせると共に、元ホームノードへ誤ってやってきたパケットに新ホームノードを通知してやらなくてはならない。また、元ホームノードにおいて、新ホームノードを知らせるための遍歴情報の削除のタイミングを考慮しなければならない。

4.1.4 ホームページ移送機構の方針

まず、前項1. で挙げたキャッシュディレクトリ転送中のパケットの処理に対する対処法を検討する。

キャッシュディレクトリへは通常、同ノードからと他ノードからのアクセスがある。同ノードからのアクセスは、システムバスのコントローラが、そのキャッシュディレクトリへアクセスを検知したら、そのアクセスを阻止するように対処させる。

また、他ノードから生じるキャッシュディレクトリへのアクセスに関しては、

転送中にキャッシュディレクトリへアクセスしようとするパケットに対する対処法を以下の 3 つの方法で比較検討し、最適な方法を選択した。

1. キャッシュディレクトリ転送中に元ホームノードに送られてくるホームページにアクセスしようとするパケットを新ホームノードに転送し、転送処理終了までペンディングしておく。ペンディング機構はハードウェア的に実装し、処理を高速に実行できるものとする。転送終了後、速やかに処理を再開する。ペンディングする領域はそのノードの主記憶内に専用に設け、多くのパケットの保存も可能にする。
2. キャッシュディレクトリを転送する前に、ホームユニットと同ミッションに属するユニットを持つノードに対して、キャッシュディレクトリ転送開始通知パケットを送り、キャッシュディレクトリ転送終了まで、ホームページへのアクセスを中止させる。その通知パケットを受け取った各々のノードでは、ホームページへのアクセスを中止する処理が完了した後、キャッシュディレクトリ転送通知パケットに対する返信を送る。元ホームノードでは全てのノードから返信を受け取った後、転送を開始し、転送処理終了後、転送処理完了通知を先ほどと同じノードに送り、ホームページへのアクセスを再開させる。
3. 通常のノード間通信において、パケットの受信先のノード NIC が多くの処理を抱えており、それ以上パケットの処理ができない場合は、送信元へパケットが送り返される。送信元のノードでは、送り返されれきたパケットを再送する。この場合、パケットの内容は一切変更されず、送信元と受信先のノード間を、受信先がパケットを処理可能になるまで往復する。これと同様に、キャッシュディレクトリ転送中にホームノードへやってくるパケットを送信元へ送り返し、送信元のノードとの間を往復させる。

1. の方法は、キャッシュディレクトリ転送中に他のノードの NIC の処理に影響を与えず、転送処理終了後、転送中に処理できなかったパケットを速やかに処理できる方法である。ペンディングの作業をソフトウェアで行うと、ペンディングを行う度にプロセッサに割り込みがかかり、例えペンディングするパケットの量が少ないとしても、割り込みがかかることはホームページ移送処理を大きく遅らせる原因になる。したがってペンディングをハードウェアで実装するものとする。しかし、ペンディングをハードウェアで実装するには、ハードウェ

ア資源に対する要求が多く、現時点でこれだけのハードウェア資源を使用できるかは判断できない。

2. の方法は、各ノードでホームノードへのパケットの送信を中止するので、キャッシュディレクトリ転送中に、キャッシュディレクトリへアクセスしようとするパケットが発生しないことが保証されている。しかし、キャッシュディレクトリ転送までに準備する処理のため、キャッシュディレクトリの転送開始が遅くなる。

それに比べ、3. の方法は、キャッシュディレクトリ開始までの準備時間が短い。4.1.1で述べたように、キャッシュディレクトリ転送中にキャッシュディレクトリにアクセスしようとするパケットの量は少ないという条件のもとでは、パケットを送信元に送り返すことによる通信トラフィックの増大が少ないと考えられる。したがって、この方法は、キャッシュディレクトリ転送処理開始までの時間が短く、ハードウェア資源に対する要求が比較的少ない。

また、1.、3. の方法は 2. の方法に比べ、他のノードの NIC に対し、特別な処理を要求しない。元ホームノードと新ホームノード以外の NIC は通常と同じ処理を行っていれば良い。

以上の事柄と他の方法への応用性を検討した結果、3. の方法を最適な処理法と考え、採用した。

次にこの方法のもとで前項 2. への対処法を述べる。

キャッシュディレクトリ転送終了後に、GDS の管理するテーブルは書き換えられるが、ネットワーク上のパケットは、その情報を知ることはできない。そこで元ホームノードへ誤ってやってきたパケットに対し、再度、送信元でホームノードの位置を確認するような情報をそのパケットに付加し、送信元へ送り返す。送信元では、GDS の対応表を再確認し、新ホームノードへアクセスが可能となる。

この処理は、各ノードの NIC がハードウェアとしてキャッシュしている GDS の管理テーブルを更新し、すべての GDS の管理表が更新された時点までサポートするものとする。

4.1.5 ホームページ移送機構の概要

本実装における、ホームページ移送機構の概要を述べる。

まず、OS あるいはユーザによりホームページ移送の判断がなされたのを受け

て、主CPUのプロセッサはNICのプロトコルプロセッサへホームページ移送プログラムの開始を命令する。以下はその後の処理である。

開始処理　元ホームノードから新ホームノードへホームページ移送することになる。ここでの処理は、元ホームノードと新ホームノードのNICが共にホームページ移送処理が行われることを認識し、新ホームノードでは、送られてくるであろうキャッシュディレクトリの領域を確保することである。

キャッシュディレクトリ転送のための前処理　ここでは、キャッシュディレクトリ転送中に、キャッシュディレクトリへアクセスがあった場合に行う処理の準備をする。具体的には、他ノードからやってくるパケットへの処理（送信元のノードへ送り返す）と同ノードからのアクセスに対する処理（システムバスをロック）の2種類である。

キャッシュディレクトリ転送処理　キャッシュディレクトリ本体を新ホームノードへ転送し、所定の場所にキャッシュディレクトリを格納する。

キャッシュディレクトリ転送後の処理　ここでは、ホームページ移送処理が終了し、システムが通常状態にスムーズに移行するための処理を行う。具体的には、「キャッシュディレクトリ転送のための前処理」で行った処理に対して、それぞれを通常処理へ戻す処理とGDSの管理するテーブル本体とそのキャッシュを変更することである。また、GDSの管理する表を変更する以前に発行されたパケットが新ホームノードへ正しくアクセスできるような処理もする。

4.2　ホームページ移送機構の実装

ここでは、ホームページ移送機構の実装について詳細に述べる。

4.2.1　ホームページ移送機構実装の要点

ホームページ移送機構の高速化の要点として以下の3点を挙げる。

1. NICのプロトコルプロセッサ上でホームページ移送のプログラムが動作すること。
2. パケットの送受信において、なるべくポーリングをすることで割り込みの回数を減らす。
3. 一部、専用のハードウェアを設けることで高速化を図る。

ホームページ移送機構は、キャッシュディレクトリの転送とGDSの管理する表の書換えが主な作業である。また、元ホームノードと新ホームノードの間の

パケットのやり取りにより処理が進行していく。

したがって、1. のように、キャッシュディレクトリやGDSを通常利用し、パケットの処理を司るNICのプロトコルプロセッサ上でプログラムが動作するのがホームページ移送が高速に実行されるため必要と考えられ、そのように実装した。

また、通常、パケットがやってきたノードでは、そのパケットを特別なハードウェアにより処理できない場合は、プロトコルプロセッサに割り込み[4]をかけることで、そのパケットの種別を判別する。また、割り込みをかけることで、プロトコルプロセッサによる複雑な処理を指定することができる。しかし、割り込みがかかることは、ハードウェアのみでの処理に比べ、多くのサイクル数を要する。よって、ホームページ移送処理を高速に行うためには、割り込みを最低限にすることである。本実装では、2. のように、あるパケットの送信に対し、その返信が速やかに帰ってくると想定される場合はパケットを送信したのち、その返信を受け取るまでポーリングする。これによって、返信パケットを受け取った後の処理の再開が速やかに行うことができる。

3. において、専用に設けたハードウェアは以下の通りである。

- キャッシュディレクトリ転送中に同ノード内からキャッシュディレクトリへアクセスされないように、NIC内のコントロールレジスタに専用の領域を設け、その処理をハードウェアで実装する。
- キャッシュディレクトリ転送中に元ホームノードへやってきたパケットを送信元へ送り返すために、NIC内に専用のレジスタを設け、送り返す処理をハードウェアで実装する。
- ホームページ移送後に、元ホームノードへ誤ってやってきたパケットに対し、再度ホームノードの位置を確認するような情報をそのパケットに付加し、送信元へ送り返すための専用のレジスタを設け、その処理をハードウェアで実装する。

4.2.2 ホームページ移送機構実装の概要

ホームページ移送のプログラムはNICのプロトコルプロセッサによって実行される。ホームページ移送のプログラムは大きく分けて、2つの関数から構成される。ホームページ移送を始める元ホームノードの処理を司る関数と、ホームページ移送に関するパケットを受信したときに処理を司る関数である。

元ホームノードからの処理は、start_page_migration() 関数が呼び出されるこ

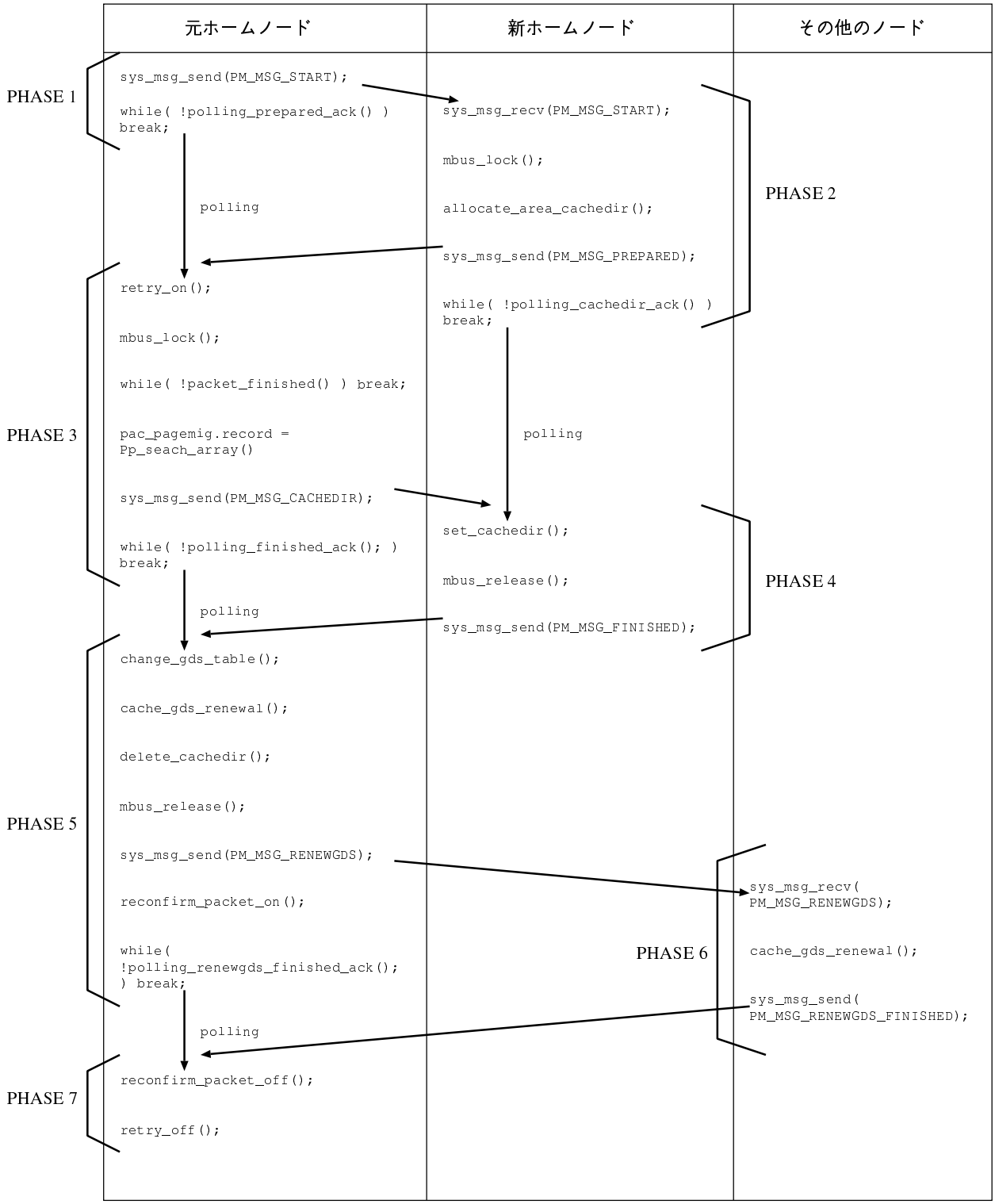
元ホームノード
新ホームノード
その他のノード
PHASE 1
PHASE 2
PHASE 3
PHASE 4
PHASE 5
PHASE 6
PHASE 7
sys_msg_send(PM_MSG_START);
while(!polling_prepared_ack()) break;
polling
retry_on();
mbus_lock();
while(!packet_finished()) break;
pac_pagemig.record = Pp_seach_array()
sys_msg_send(PM_MSG_CACHEDIR);
while(!polling_finished_ack();) break;
polling
change_gds_table();
cache_gds_renewal();
delete_cachedir();
mbus_release();
sys_msg_send(PM_MSG_RENEWGDS);
reconfirm_packet_on();
while(!polling_renewgds_finished_ack();) break;
polling
reconfirm_packet_off();
retry_off();
sys_msg_recv(PM_MSG_START);
mbus_lock();
allocate_area_cachedir();
sys_msg_send(PM_MSG_PREPARED);
while(!polling_cachedir_ack()) break;
polling
set_cachedir();
mbus_release();
sys_msg_send(PM_MSG_FINISHED);
sys_msg_recv(PM_MSG_RENEWGDS);
cache_gds_renewal();
sys_msg_send(PM_MSG_RENEWGDS_FINISHED);

図 6: プログラムの進行

とにより開始される。これにより、ホームページ移送は開始する。新ホームノードや、その他コピーページを持っているノードでの処理は所定のパケットの受信によって開始される。

ホームページ移送が判断されると、主 CPU からホームページ移送に必要な情報とともに NIC のプロトコルプロセッサへホームページ移送開始通知が送られる。それを受けて、プロトコルプロセッサは、start_page_migration() を実行する。ここからホームページ移送プログラムが始動する。この関数には引数として、ホームページの大域アドレスを構成する mission ID と仮想アドレス、さらにどこからどこのノードへ移送するかを示すノード ID を記述する。この関数は、ホームページを移送する側、即ち元ホームノードの NIC で行う処理を記述している関数で構成される。 途中、あるノードに対する要求パケットの応答待ちにより処理が一時休止する。

受信側のノードの関数は、start_page_migration() で発行されたパケットを受信することで動作を開始するが、受信するパケットの種類により起動する関数が異なる。受信側のノードの関数は、パケットの受信で起動し、応答パケットを発行することにより、終了または、一時休止する。

ホームページ移送機構のなかで、ハードウェアで実装される部分はエミュレータにより実装した。

4.2.3 ホームページ移送機構の実装

ここでは、ホームページ移送機構のプログラムの詳細を記述する。本実装では以下のパケットを使用する。

PM_MSG_START 元ホームノードからホームページの移送を新ホームノードに伝えるパケット

PM_MSG_PREPARED 新ホームノードで、[PM_MSG_START] パケットの受信を受けてキャッシュディレクトリの受信準備をし、それの完了通知パケット

PM_MSG_CACHEDIR 元ホームノードから新ホームノードへキャッシュディレクトリの転送をするパケット

PM_MSG_FINISHED 新ホームノードでキャッシュディレクトリを受信し、キャッシュディレクトリを主記憶に格納した後、元ホームノードに対して送信するパケット

PM_MSG_RENEWGDS　同ミッションに属する他のノードに対し、キャッシュしているGDSの表の更新を実行させるパケット

PM_MSG_RENEWGDS_FINISHED　[PM_MSG_RENEWGDS]パケットに対する実行完了通知パケット

パケットの送受信は、sys_msg_send() と sys_msg_recv() の2つの関数により行われる。パケットには、基本的にホームユニットが属するmission IDと、ホームページの仮想アドレス、新ホームノードのノードID、データ（今回はキャッシュディレクトリ本体）を送る。データは必要なときのみ付けられ、それ以外はNULLが格納されている。パケットのあて先には、送り先ノードを指定する。

プログラムの進行の様子を図6に示す。以下、ホームページ移送で実行される関数をパケットの送信、受信によるフェーズ分けをして説明する。

PHASE 1　元ホームノードにおけるプログラム

- sys_msg_send(PM_MSG_START)：元ホームノードから新ホームノードに対し、ホームページ移送開始を通知する。
- polling_prepared_ack()：新ホームノードからの返信パケット ([PM_MSG_FINISHED]) を受信したら、1を返す。この関数を常に監視することでポーリングをする。ただし、ここでのポーリングはPHASE2で割り込みがかかるため、このポーリングより優先度の高い処理が発生した場合は、プロセッサをそちらの処理に割り当てることができる「優先度付きポーリングである。通常なら[PM_MSG_START]パケットに対する返信パケットを受け取るまでの時間が多くかかると予想される場合は、他の処理が実行できない時間が多くなるため、ポーリングを選択することはできない。しかし、ここで割り込みをかけると以下のPHASEの開始のたびに、割り込み処理をしなければならなくなり、全体の処理の終了が大幅に遅れる。したがって、ポーリングを選択した。

PHASE 2　新ホームノードにおけるプログラム

- sys_msg_recv(PM_MSG_START)：元ホームノードからのホームページ移送通知を受信する。ここでNICのプロトコルプロセッサに割り込みがかかり、プロセッサが起動する。

- mbus_lock(): ホームページの仮想アドレスを引数に取る。同ノード（新ホームノード）からホームページへのアクセスを中断させる。実際はNIC 内のコントロールレジスタにこの仮想アドレスを書きこむことでハードウェア的に実装する。
- allocate_area_cachedir()：ホームページの仮想アドレスを引数に取る。主記憶上のキャッシュディレクトリ用の領域から、転送されてくるキャッシュディレクトリを格納するための領域を切り出す。キャッシュディレクトリ用の領域は主記憶上に存在し、NIC 専用の領域である。
- sys_msg_send(PM_MSG_PREARED)：元ホームノードに対し、キャッシュディレクトリの転送準備が整ったことを知らせる。
- polling_cacehdir_ack()：元ホームノードからのパケット ([PM_MSG_CACHEDIR]) を受信したら、1 を返す。この関数を常に監視することでポーリングをする。

PHASE 3 新ホームノードから

パケット [PM_MSG_PREARED] を受け取ることで開始する。元ホームノードでは、このパケットの受信までポーリングしていたので、速やかに処理を開始する。

- retry_on(): ホームページの仮想アドレスを引数に取る。キャッシュディレクトリへアクセスしようとするパケットを送信元に送り返す。これはNIC 内のコントロールレジスタ部にこの仮想アドレスを書きこむことでハードウェア的に実装される。
- mbus_lock(): ホームページの仮想アドレスを引数に取る。同ノード（元ホームノード）からキャッシュディレクトリへのアクセスを中断させる。
- packet_finished()：retry_on() 実行以前に、キャッシュディレクトリにアクセスがあったパケットの処理が完全に終了したら、1 を返す。
- Pp_seach_array()：キャッシュディレクトリ本体を転送するために、パケットの内容を記述する構造体に、キャッシュディレクトリを示すポインタを格納する。この関数はキャッシュディレクトリの格納されているアドレスを返す。
- sys_msg_send(PM_MSG_CACHEDIR):新ホームノードへキャッシュディレクトリを転送する。

- polling_finished_ack()：元ホームノードからの
  パケット ([PM_MSG_FINISHED]) を受信したら、1 を返す。この関数を常に監視することでポーリングをする。

PHASE 4　元ホームノードからパケット [PM_MSG_CACHEDIR] を受け取ることで開始する。新ホームノードでは、このパケットの受信までポーリングしていたので、速やかに処理を開始する。

- set_cachedir()：受信したキャッシュディレクトリを PHASE 2 で確保した主記憶上の領域に格納する。
- mbus_release：PHASE 2 での mbus_lock() で行った設定を解除する。
- sys_msg_send(PM_MSG_FINISHED)：元ホームノードへキャッシュディレクトリの転送作業が終了したことを通知する。

PHASE 5　新ホームノードからパケット [PM_MSG_FINISHED] を受け取ることで開始する。元ホームノードでは、このパケットの受信までポーリングしていたので、速やかに処理を開始する。

- change_gds_table()：GDS にアクセスして、GDS が管理する対応表を書きかえる。このとき、GDS ではこの対応表をキャッシュしているノードに対し、無効化要求を送りすべてのキャッシュの内容を更新する。GDS ではこの作業終了後、元ホームノードに対し処理完了を通知する。この通知を待って次に進む。
- cache_gds_renewal()：同ノードで NIC の専用レジスタにキャッシュしている GDS の対応表を更新する。引数としては新ホームノードのノード ID をとる。
- delete_cachedir()：元ホームノードにあるキャッシュディレクトリを消去する。
- mbus_release()：PHASE 3 での mbus_lock() で行った設定を解除する。
- sys_msg_send(PM_MSG_RENEWGDS):同ミッションに属する他のノードに対し、キャッシュしている GDS の表の更新を実行するように通知する。
- reconfirm_packet_on()：GDS の変更後、既にネットワーク上にあるパケットは、しばらくの間元ホームノードに送られてくるので、これらのパケットに「GDS 再確認要求」を付加して、返信するように NIC 内

のコントロールレジスタにホームページの仮想アドレスを書きこむ。

- polling_renewgds_fineshed_ack()：同ミッションに属する他のノードからのパケット ([PM_MSG_RENEWGDS_FINISHED]) を受信したら、1を返す。この関数を常に監視することでポーリングをする。

PHASE 6　このフェーズは同ミッションに属する他のノードで実行されるプログラムである。

- sys_msg_recv(PM_MSG_RENEWGDS)：元ホームノードからのパケット [PM_MSG_RENEWGDS] を受け取る。ここでは、このパケットの判別をハードウェアで行い高速に以下の処理に進む。
- cache_gds_renewal()：同ノードで NIC 内の専用レジスタにキャッシュしている GDS の対応表を更新する。引数としては新ホームノードのノード ID をとる。
- sys_msg_send(PM_MSG_RENEWGDS_FINISHED)：
  cache_gds_renewal() の処理が完了したことを元ホームノードへ知らせる。

PHASE 7　元ホームノードでの処理である。このフェーズでホームページ移送処理は完了する。元ホームノードでは、このパケットの受信までポーリングしていたので、速やかに処理を開始する。

- reconfirm_packet_off()：PHASE 5 で reconfirm_packet_on() が行った設定を解除する。
- retry_off()：PHASE 3 で retry_on() が行った設定を解除する。

4.2.4　プロセッサ上のプログラム、ハードウェアの処理の相関関係

本実装では、主 CPU 上のプログラム、NIC のプロトコルプロセッサ上のプログラム、またハードウェアにより実装された処理が相互に関わって全体の処理を実行している。ここではその関係をまとめる。

アプリケーションプログラムは Colony の OS である Colonia の動作環境で実行されている。このアプリケーションプログラムの実行状況により、OS Colonia、あるいはユーザがホームページ移送のタイミングを判断する。

OS が判断した場合は、主 CPU から NIC のプロトコルプロセッサにホームページ移送の開始を通知する。ユーザが判断した場合、システムコールを発行して一旦、主 CPU にホームページ移送の開始を命令する。

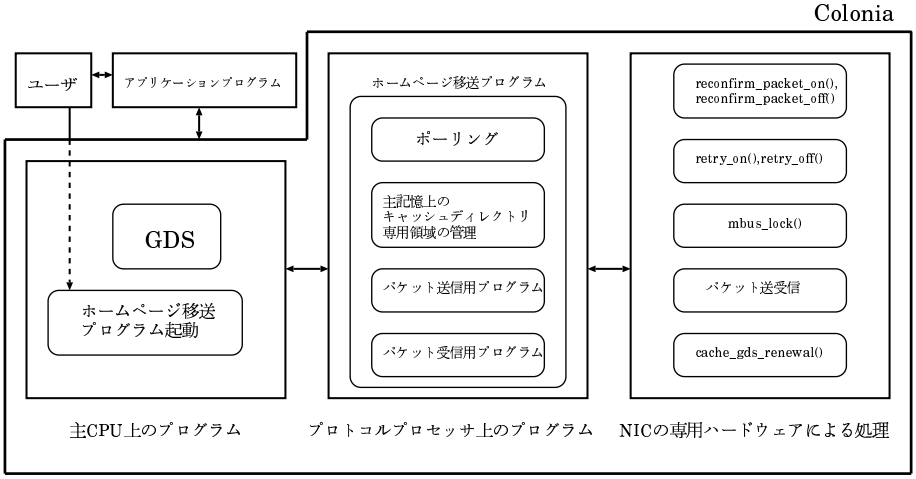

図 7: 各処理系の相関図

ホームページ移送の開始がプロトコルプロセッサに通知されると、NIC のプロトコルプロセッサに処理の主導が移る。プロトコルプロセッサは NIC 上のハードウェアと相互に関わりながら、ノード間通信や、パケットに対する処理を実行する。ホームページ移送が開始されると、ほとんどの処理はプロトコルプロセッサと、ハードウェアの処理で占められるが、途中 GDS の管理するテーブルの書換えは主 CPU のプロセッサが動作する。

主 CPU 上のプログラム、プロトコルプロセッサ上のプログラム、NIC のハードウェアによる処理の相互関係を図 7に示す。

4.3 考察

コンピュータ・コロニーでは、共有メモリを利用した通信を行うことで、無駄な通信回数を減らす通信の最適化 [5] や効率的な負荷分散を目標としているを。本研究では、そのような目標達成の一環として、ホームページの移送機構を考案した。

本実装では、キャッシュディレクトリ転送中にやってくるパケットを送信元に送り返すことが主な特徴である。4.1.1で述べたように、実装の前提として、キャッシュディレクトリ転送中はパケットの往来は少ないものとしている。パケットの往来が多いと仮定すると、この方法では、元ホームノードと送信元を行き交うパケットの量が多くなるので、通信トラフィックの増大が無視できなくなり、そのページ以外のパケットの通信に弊害がでると予想される。

キャッシュディレクトリ転送中にやってくるパケットの往来が多いものとすると、本実装で採用した方法は取れない。この場合、4.1.4で述べた 1.、2. の処理法ならば、対処することができる。

したがって、今後の研究により、ホームページ移送のタイミングや使用できるハードウェア資源が確定すれば、検討した他の方法へ移行するということも考えられる。

第 5 章　おわりに

コンピュータ・コロニーでは、共有メモリを利用した通信を行うことで、無駄な通信回数を減らす通信の最適化や効率的な負荷分散を行う。本論文では、そのような目標を達成するためにホームページの移送機構を考案した。

ホームーページ移送機構の実装法としては、その高速性とハードウェア資源、他の実装法への応用を考え合わせ、キャッシュディレクトリ転送中にホームページにアクセスしようとするパケットを送信元に送り返す方法を採用した。また、ホームページ移送のタイミングは、アプリケーションプログラムの実行におけるフェーズの移りかわりなど、ホームページに対するパケットの総量が少ないときを想定した。

今後の研究では、ホームページ移送を誰がどのような判断で行うのが最も効率的であるか、即ち、ユーザあるいは OS がどのようなタイミングで判断して行うのが最も効率的であるかという問題の研究が課題であるし、また、そのようなタイミングにおいてもっとも適切な実装法を考案しなければならない。本論文では、他の実装法の例を挙げたが、これらの実装法も状況に合わせて適用される可能性がある。

関連研究としては、ユーザがプログラムを投入し、ミッション、ユニットが生成されたときにホームページをどのノードに配置されるかという研究が挙げられる。この研究は本論文でも、ホームページ移送のタイミングを決定する問題に大いに係わるものである。

また、すでにコンピュータ・コロニーでも扱われているが、ユニット移送の研究 [6] がある。これも通信の最適化、効率的な負荷分散のために行われる。ここではユニットそのもののデータ転送と共に、GDS の管理する Unit ID と Node ID の対応を換える。ホームページにリモートアクセスを頻繁に繰り返すユニッ

トがあれば、そのユニットをホームノードへ移送してやれば良い。このような場合、本論文のアプローチとユニット移送のアプローチには違いはあるが、通信の最適化、効率的な負荷分散の実現を同じ目標としている。したがって、どちらの方法を選択するかは、それぞれの移送のコストと移送後のそれぞれが与える影響を考慮して決定しなくてはならない。これも今後の研究として挙げられるだろう。

第 6 章　謝 辞

本研究の機会を与えてくださった、本研究室の富田眞治教授に深甚な謝意を表します。

また、本研究に関して適宜ご指導、ご鞭撻を賜った森眞一郎助教授、五島正裕助手に深く感謝致します。

さらに、共同研究者である増田峰義氏、鳥崎唯之氏、石川智祥氏をはじめとして、日頃様々な角度から助力して下さっ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。

参考文献

[1] 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, Vol. 97-OS-34 (1997).

[2] KaiLi, PaulHudak: Memory Coherence in Shared Virtual Memory Systems, *The ACM Transactions on Computer Systems*, Vol. 7, No. 4, pp. 321–359 (November 1989).

[3] 増田峰義, 他: 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, 信学技報, Vol. CPSY99-53 (1999).

[4] JeanBacon (藤田昭平, 篠田陽一, 今泉貴史訳): 並列分散システム、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総合的アプローチ, 株式会社トッパン (1996).

[5] 伊達新哉: ネットワーク分散型並列システムを実現する通信機構 (2000).

[6] 山添博史, 他: コンピュータ・コロニーにおける高速移送可能な並列アクティビティの実現, システムソフトウェアとオペレーティングシステム, Vol. 82-12 (1999).