システムソフトウェアと
オペレーティング・システム 85－4
（2000．8．4）

分散OS Coloniaにおける並列アクティビティの高速移送

増田峰義† 鳥崎唯之† 五島正裕†
森眞一郎† 富田眞治†

分散OS Coloniaは、我々が開発中の分散システムである，コンピュータコロニーのためのOSである．コンピュータ・コロニーでは，効率的な負荷分散を実現するために，ノード間でのアクティビティ移送機構を持つ．並列ジョブの高速実行や，高いスケーラビリティを実現するためには，移送を高速に行う必要がある．我々は，移送開始から移送先で再実行可能になるまでの移送コストを，できるだけ短くする移送機構を実装した．この移送機構をエミュレータ上で実行し，そのソフトウェア・オーバヘッドの評価を行った．実行サイクル数で測定したところ，9726サイクル(理想値2199サイクル)という結果を得た．この結果に基づく考察により，アクティビティの移送を最短で50μs程度で行う展望を得た．

Low latency migration mechanism of distributed operating system Colonia

Mineyoshi Masuda,† Yuishi Torisaki,† Masahiro Goshima,†
Shin-ichiro Mori† and Shinji Tomita†

Colonia is a distributed operating system designed for the Computer Colony we have proposed. Computer Colony has process migration mechanism between nodes for realizing efficiently dynamic load balancing. To realize good scalability and high speed execution of parallel jobs, we propose a low latency migration mechanism that minimizes the time between activity stop on source node and activity restart on destination node. Our simulation results shows that the software overhead is 9726 cycle(ideal 2199 cycle). As a result, we have a prospect that migration can be done in 50μs in the best case.

1. はじめに

現在，我々は新しい計算機環境として，コンピュータ・コロニー[1),2)]を開発している．コンピュータ・コロニーは，様々な性能を持つ計算機を，多数高速なネットワークで接続した分散システムである．コンピュータ・コロニーの各ノード上では，分散OS Coloniaが動作し，ユーザに対して分散共有メモリをベースとした，細粒度の並列計算環境を提供する．

コンピュータ・コロニーは，以下の機能の実現を目指す．

- SSI(Single System Image)[3)]
  コンピュータ・コロニーは，ユーザに対してSSIを提供する．ユーザには，システム全体が一つのマシンに見え，どのノードでジョブが実行されているのかを意識しない．
- 動的負荷分散
  システム内のノードの負荷に応じて，動的な負荷分散を行う．負荷分散は，負荷の重いノードから軽いノードへ，プロセスを移送することで実現する．プロセスの移送は，ユーザに不可視である．
- フェアシェア(Fair Share)[5)]
  システム内の資源は，ユーザに対して公平(Fair)に分配される．ユーザは課金の多寡に応じてチケットを持ち，チケットの分だけ資源を使用することができる．

我々は，このような計算機環境を実現するために，コンピュータ・コロニー向け並列アクティビティ・モデルとして，ミッション-ユニット・モデルを考案した[4)]．ミッション-ユニット・モデルは，タスク-スレッド・モデルを分散環境向けに拡張したモデルである．ミッションはタスクに，ユニットはスレッドに相当し，ユニットはノード間で移送可能である．このユニットの移送機構を用いて動的な負荷分散を行い，高い計算能力とスケーラビリティを実現する．

† 京都大学大学院情報学研究科
Graduate School of Infomatics, Kyoto Univ.

しかし，移送には以下に示すコストがかかる．

(1) グローバル・スケジューリングのコスト
どのユニットを，どのノードへ移送するかを決定するのにかかるコスト．

(2) データ転送のコスト
移送元でユニットを止め，管理データおよびページを転送し，移送先で継続実行の準備をするコスト

これらのコストが非常に大きい場合，効率的な負荷分散は実現できない．また，同期を多用するミッション，リアルタイム性を要求するミッションでは，移送ユニットの実行停止時間が，ミッション全体の実行を妨げる．したがって，高速なユニット移送機構が求められる．

本稿では，特に(2)のコストを可能な限り削減した移送モデルを示す．さらに，そのモデルをエミュレータ上で実装し，移送時のソフトウェア・オーバヘッドを測定した結果を報告する．

まず，2章でミッション・ユニットモデルについて説明し，3章でユニットの高速移送の手法について述べる．その後，4章において，エミュレータ上で行ったユニットの移送コストの評価結果を示す．

2. ミッション-ユニット・モデル

本章では，Colonia の並列アクティビティ・モデルである，ミッション-ユニット・モデルについて述べる．

2.1 モデル概要

ミッション-ユニット・モデルは，共有メモリ上の並列アクティビティモデルであるタスク-スレッド・モデルを，分散共有メモリ環境向けに拡張したモデルである．

ミッションはタスクに，ユニットはスレッドに相当する．ユニットが複数ノードで動作することで，ミッションを並列実行する．システムは，ノードの負荷状況に応じて，ユニットの投入，移送を行い，負荷を動的に分散させる．移送に伴うユニットの位置の変化は，Global Directory Service(GDS)[7]が管理する．

このモデルでは，タスク-スレッド・モデルと同様に，プログラミング環境として共有メモリを提供する．ただし，分散環境下での共有領域の管理コストを低減するため，全アドレス空間ではなく，アドレス空間の一部を共有する．

以下，このモデルのアクティビティ，ユニットの共有アドレス空間および共有領域の管理，GDS の順に説明する．

2.2 アクティビティ

2.2.1 定　　義

ミッション-ユニット・モデルでは，以下の 3 つのアクティビティを定義する．

ミッション　タスク-スレッド・モデルのタスクに相当する．複数のユニットにより構成され，それらが並列に動作することでミッションが実行される．ミッションは，システム内で一意な識別子 (ミッション ID) を持つ．

ユニット　タスク-スレッド・モデルのスレッドに相当する．並列実行の単位である．ユニットは，ミッション内で一意な識別子 (ユニット ID) を持つ．また，プロセッサ割り当ての単位としてスレッドを持つ．

スレッド　プロセッサ割り当ての単位であり，プロセッサ・コンテキストを持つ．スレッドはノード・ローカルなアクティビティであり，ノード内で一意な識別子 (スレッド ID) を持つ．

2.2.2 システム・ミッション

Colonia では，システムもミッション・ユニットの形をとる．すなわち，分散システムを構成する各ノードに一つづつ，カーネルに相当するシステム・ユニットが存在し，全体でシステム・ミッションを構成する．ノード内で動作しているアクティビティの管理は，システム・ユニットが行う．

ユニットの移送は，システム・ユニット間のやりとりで実現する．システム・ユニットは，他ノードのシステム・ユニットとの間で調整を行い，移送するユニット，移送先ノードを決定する．移送に伴うデータ転送は，システム・ユニット間のメッセージ通信で行う．

2.2.3 システム・スレッド

システム・ユニットはユーザ・ユニットと同様，スレッドを持ち，これをシステム・スレッドと呼ぶ．システム・スレッドは，システム・ユニットの行う処理の内，一定時間以内に終了することが保証されず，長く継続するような処理を担当する．

ミッション，ユニットおよびスレッドの関係を図1に示す．図上部は，ユーザ・ミッション，下部はシステム・ミッションであり，それらのユニットが二つのノードにまたがっている様子を示している．波線の矢印はプロセッサ・コンテキストであり，システム・ユニットは複数のプロセッサ・コンテキストを持つことを示している．

図1は，同時にアドレス空間も示している．アドレス空間については，次節で詳述する．

2.3 アドレス空間

タスク-スレッド・モデルでは，同一タスクに属するスレッドは全アドレス空間を共有するが，ミッション-ユニット・モデルでは，アドレス空間を部分的に共有する．ユニットのアドレス空間を二つの領域に分ける (図1)．一つは，ユニット固有領域 (unit private space) と呼び，スタックなどのユニットに固有なデータを配置する．もう一方の領域は，ミッション共有領域 (mission shared space) と呼び，同一ミッションに属する全てのユニットで共有する領域である．ミッション共有領域には，ユニットが共通に使用するデー

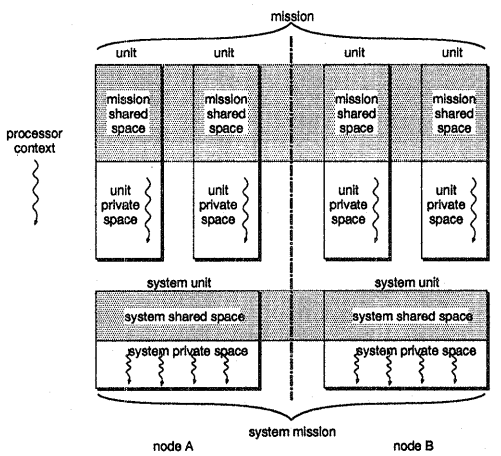

図1　アドレス空間の構成

タ，およびコードなどを配置する．システム・ユニットでは，これらの領域を特に，system private space, system shared space と呼ぶ．

アドレス空間を二つの領域に分けたのは，管理コストを軽減するためである．タスク-スレッド・モデルでは，全てのスレッドはアドレス空間を共有する．そのため，スレッドが固有に使用する領域を確保する際にも，他のスレッドとの調整および，そのための通信が必要となる．ミッション-ユニット・モデルでは，ユニット固有の領域を設けることで，このような不要なコストを削減する．

2.4　共有領域管理

Colonia は，ミッション共有領域という形で，ユーザに対して共有メモリを提供する．共有領域はページ単位で管理され，Shared Virtual Memory(SVM)[6] を構成する．

共有領域の各ページには，それぞれホームページと呼ばれるページが存在し，ホームページをリモートノードのメインメモリへキャッシュしたページをコピーページと呼ぶ．また，ホームページが存在するノードを，そのページのホームノードと呼ぶ．リモートノードで，共有ページのリードミスが発生した場合，ホームページからデータをフェッチする．ホームノードには，コピーページを管理するディレクトリが存在し，共有ページへのライトは，ホームノードを経由して一貫性制御が行われる．

ホームページは，ノードではなくユニットに対して割り付けられる．割り付けられたユニットを，そのホームページのホームユニットと呼ぶ．ホームページは，ホームユニットと同じノードに存在する．したがって，ホームノード = ホームユニットが動作しているノードである．ページとホームユニットの関係は，ホームユニット・テーブルによって記述され，GDS (2.5節参照) がその管理を行う．

システムの共有領域である system shared space も同様の方式で管理する．system shared space は，固定長の領域に分割され，各領域ごとにホームを担当するシステムユニットを静的に割り当てる．

2.5　GDS

ユニットの移送に伴う位置情報の変化は GDS が管理を行う．GDS は，システム内の全てのユニットごとに，管理情報を持つ．管理情報はレコードと呼ばれる構造体に格納される．

ユニットの移送が行われ，移送先でユニットが継続実行可能になると，移送先ノードのシステム・ユニットは GDS が持つレコードを移送ユニットの ID で検索し，レコード内の位置情報を変更する．ユニットの位置情報を知りたい場合も同様に GDS を ID で検索する．GDS は，system shared space に配置され，システム内の任意のノードから参照することができる．

3.　ユニット移送

ユニットの移送機構は，Colonia の目標である動的な負荷分散を行うための機能である．負荷の重いノードから，軽いノードへユニットを移送することで，計算資源の有効利用を図ることができる．効率的な負荷分散を行うためには，ユニットの移送は高速に行う必要がある．

我々は，ユニット移送機構のフェーズの内，移送元でユニットの実行を止めてから，移送先でユニットの実行が可能な状態になるまでの時間，をできるだけ短くする移送モデルを考案した．

以下では，まず，高速化のための方針について述べる．次に，移送処理の流れについて説明する．その後，ミッション-ユニット・モデルのデータ管理構造を示し，移送に際して行う処理について述べる．

3.1　方　　針

データ転送処理は，およそ二つのフェーズに分かれる．一つは，管理データ転送のフェーズである．このフェーズでは，移送元でユニットを止め，管理データを転送し，移送先で継続実行の準備をする，の一連の処理を行う．もう一つのフェーズは，ユニットが使用するページを転送するフェーズである．

それぞれのフェーズで，高速移送を実現する上で問題となる点と，解決のための方針を以下にまとめる．

3.1.1　管理データの転送

移送処理は，移送先ノード内に新たなユニットを生成する点で，新規ユニットの生成処理と同じである．そのため，メモリのアロケートおよび，その初期化処理を多く含む．ただ，初期化処理には，送られてきたレコードと無関係なものが多い．例えば，ページ・テーブルのエントリをクリアする処理などは，送信されてくるデータと依存性がないため，移送開始前に行うこ

とができる．このような処理は，移送時に行わず，オブジェクトの返却時やブート時に初期化しておく．使用時には，初期化を行わず，単にフェッチするだけである．このように，移送時に行わなければならない処理量を減らす．

また，Colonia では，ノード内の複数のユーザ/ミッションで共有するデータが存在する．移送された時点で，このような共有データが存在する場合には，作成する処理を省くことができる．

3.1.2 ページの転送

移送先へ転送しなければならないページの全てが移送先で必要とは限らない[8]．多量のページを使用しているユニットを移送する際，即座に必要とならないページの転送のために，移送コストが増大することは避けたい．そこで，移送時には，移送先で即座に必要となる最低限のページのみ転送し，残りのページは必要に応じて転送する方式 (Copy on reference 方式) を採用する．ページ転送については，3.5節で詳しく述べる．

3.2 移送処理の流れ

ユニットの移送で移送先ノードへ転送するデータを以下に示す．

- 管理データ
  ユニットの管理データ．
- 固有領域のページ
  ユニットの固有領域にマッピングされたページ．
- ホームページ
  ホームページおよび，一貫性制御のためのディレクトリ．

これらのデータを図2に示す順序で転送する．なお，データの転送は，移送元および移送先のシステム・ユニットの間でメッセージを送受信することで実現する．

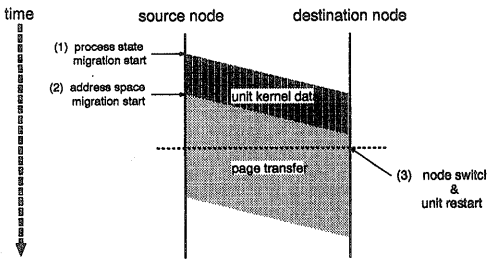

図2 データ転送順序

(1) 管理データの転送開始
移送先ノードで管理データの転送を開始する．

(2) ページの転送を開始

(3) ノード位置変更
移送先からページが一枚届いた時点で，GDS が管理しているユニットのノード位置を移送元→移送先へ変更する．これを node switch と呼ぶ．ユニットは走行可能となり，実行キューに繋がれる．

本稿で評価した移送コストは，(1) と (3) の間で移送元および移送先ノードでの処理にかかる，ソフトウェア・オーバヘッドである．

3.3 管理構造

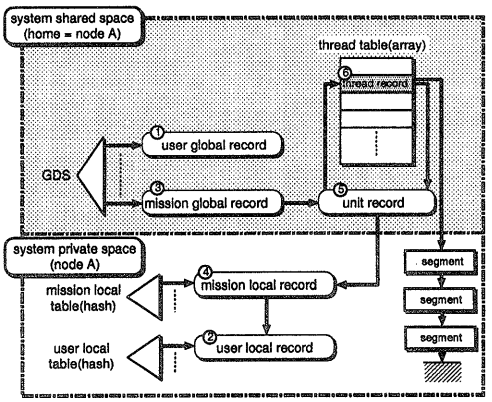

図3 ミッション・ユニットの管理構造

Colonia のアクティビティ管理のためのデータ構造を図3に示す．各四角は，データ構造体を表し，矢印はポインタによるリンクを表現している．図中上側の領域は，ノード A がホームを担当している system shared space であり，図中下側の領域は，ノード A の system private space である．三角形のオブジェクトは，ハッシュなどの検索テーブルを表している．

ここで，ユーザ/ミッションの拡大/縮小の定義を行う．

- ユーザの拡大
  ユニットの移送もしくは新規作成により，これまでそのユーザのユニットが動作していなかったノードにそのユーザのユニットが生成される．
- ユーザの縮小
  ユーザの拡大と逆に，移送もしくは終了によりノードからそのユーザのユニットが全てなくなる．
- ミッションの拡大
  ユニットの移送もしくは新規作成により，これまでそのユニットのミッションが動作していなかったノードにミッションが生成される．
- ミッションの縮小
  ミッションの拡大とは逆に，移送もしくは終了によりノードからそのミッションのユニットがなくなる．

以下，図中 (1)〜(6) のデータについて，ユーザ管理，ミッション管理，ユニット/スレッド管理に分けて説明する．

3.3.1 ユーザ管理

ユーザの管理データには，グローバルなものと，ノード・ローカルなものがある．グローバルレコードは，ユーザがログインした時点で設定される静的なデータであり，ローカルレコードは，グローバルレコードを基に，各ノードの状況に応じて変化する動的なデータである．それぞれのレコードが格納する情報を以下に示す．

① ユーザ・グローバルレコード:(184Byte)
- ユーザ名，グループ ID，ユーザが持つチケット情報など．

② ユーザ・ローカルレコード:(96Byte)
- ノード内に存在するこのユーザのミッションが持つチケットの総量
- 上記ミッションの内，アクティブなミッション☆が持つチケットの総量
- ユーザの CPU 使用状況を記録する history window

グローバルレコードは，ユーザがシステムにログインした時点で生成され，ローカルレコードは，ユーザの拡大時に生成され，縮小時に解放される．グローバルレコードの取得には，GDS をユーザ ID で検索する．ローカルレコードの取得には，ユーザ・ローカルテーブル (ハッシュ) をユーザ ID で検索する．

3.3.2 ミッション管理

ミッション管理データにも，ユーザの管理データと同様，グローバルなものと，ノード・ローカルなものがある．それぞれのレコードが格納する情報を以下に示す．

③ ミッション・グローバルレコード:(20Byte)
- ホームユニット・テーブルへのポインタ
- ミッションが持つチケット

④ ミッション・ローカルレコード:(28Byte)
- ノード内に存在するこのミッションに属するユニットが持つチケットの総量
- 上記ユニットの内，アクティブなユニット☆☆が持つチケットの総量
- ミッション共有領域のページテーブルへのポインタ

グローバルレコードは，ミッション生成時にシステム内で一つ生成される．ローカルレコードは，ミッションの拡大時に生成され，縮小時に解放される．グローバルレコードの取得には，GDS をミッション ID で検索する．ローカルレコードの取得には，ミッション・ローカルテーブル (ハッシュ) をミッション ID で検索する．

☆ アクティブなユニットを持つミッションを指す．

☆☆ アクティブなユニットとは，実行キューに繋がれているユニットを指す

3.3.3 ユニット/スレッド管理

ユニットおよびスレッドの管理情報は，ユニット・レコード，スレッド・レコードに格納される．ユニットの移送の際に，これらのレコードは移送元ノードから移送先ノードへ転送される．それぞれのレコードの内容を以下に示す．

⑤ ユニット・レコード:(60Byte)
- ユニットが持つチケット，ユニットの状態など．

⑥ スレッド・レコード:(228Byte)
- プライオリティなど，スケジューリングのための情報
- レジスタ・コンテクスト
- 仮想アドレス空間管理のためのセグメント・テーブル．図 3に示すように，このテーブルはリスト構造である．

3.4 管理データの転送

本節では，前節で示したデータ構造をふまえて，ユニットを移送する際のデータ転送の流れ，およびそれぞれのノードのシステム・ユニットが行う処理の詳細について述べる．

3.4.1 処理の流れ

移送処理における管理データの転送の流れを図 4に示す．この図では，node A から node B へ，ユニットが移送される様子を示している．以降では，この図を用いて処理の流れを説明する．

移送元ノード での処理

移送元では，移送するユニットの動作を停止しユニットが移送先で動作するために必要なデータの転送を行う．処理内容を以下に示す．

(1) detach 処理
移送選択したユニットが実行中であれば停止し，実行キューから外す．

(2) pack 処理
ユニット・レコードおよびスレッド・レコードをメッセージに詰め移送先ノードへ送信する．

移送先ノード での処理

送られてきたユニットを実行可能な状態にする処理を行う．処理内容を以下に示す．

(1) unpack 処理
メッセージを受信し，メッセージからレコードを取り出す．

(2) ローカルレコードの作成
移送先でユーザ/ミッションの拡大が起きた場合，ローカルレコードを作成する．

(3) ページ・テーブルの処理
移送ユニット用のページ・テーブルを用意する．

以下では，それぞれの処理について詳述する．

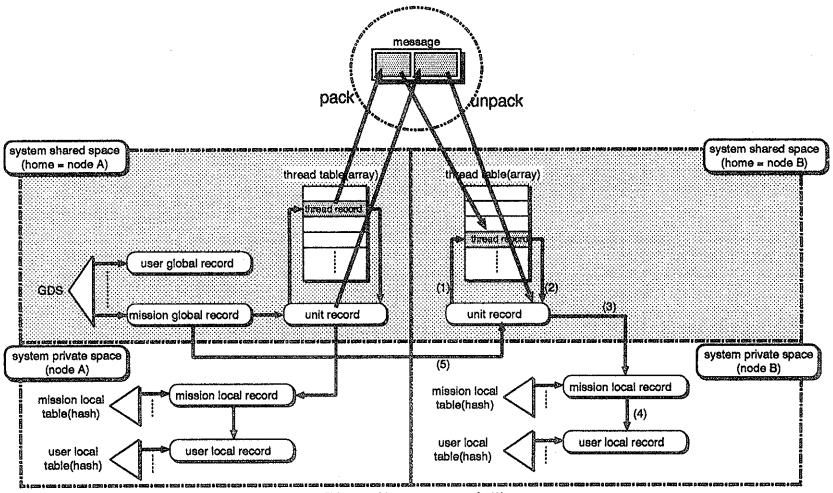

図4　管理データの転送

3.4.2 detach

スレッドの detach

スレッド・レコードを実行キューから外す．実行キューからの除去に伴い，ノード内のチケット情報が変化する．この変化をユーザ/ミッション・グローバルレコードの情報を基に，ユーザ/ミッション・ローカルレコードへ反映する．

3.4.3 pack/unpack

移送元での pack 処理，移送先での unpack 処理は，対になる処理である．すなわち，pack 処理は，転送が必要なレコード (ユニット・レコードおよびスレッド・レコード) をメッセージに詰める処理，unpack 処理は，メッセージからレコードを取り出す処理である．処理は，それぞれのノードのシステム・ユニットが行う．

pack 処理で注意すべき事は，ポインタの扱いである．移送元ノードでのポインタは，移送先ノードでは正しい値ではないからである．したがって，リスト構造のような，ポインタ・ベースのデータを pack 処理する際には，リストの要素数も添えて転送し，受信側でリストの再構築をする処理が必要となる．

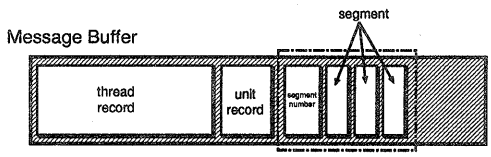

図5　pack/unpack 処理

図5に，Colonia での pack/unpack 処理の様子を示す．以下，スレッド・レコード，ユニット・レコードの pack/unpack 処理について説明する．

スレッド・レコードの pack/unpack

スレッド・レコード中のセグメント・テーブルには，セグメントのリスト構造を含むため，前述のような処理が必要である．unpack 処理では，スレッド・テーブルから空きエントリを確保し，そこへ，メッセージ中のスレッド・レコードをコピーする．同時に，セグメントを一つずつ取り出し，ポインタの張替を行い元のリスト構造に戻す．

ユニット・レコードの pack/unpack

ユニット・レコードはポインタ・ベースのデータを含まないため，ユニット・レコードの pack 処理はレコードをメッセージ領域へコピーするだけである．unpack 処理も，ユニット・レコード用に確保した領域へメッセージ中のユニット・レコードをコピーするだけである．その後，スレッド・レコードとの相互リンク (図4のリンク (1) と (2)) を作成する．

3.4.4 ローカルレコードの作成

unpack 処理の後，ユーザ/ミッション・ローカルレコードの作成を行う．しかし，ノードの状況によってはレコードを作成する必要がない．すなわち，ノードの状況が，

- ユーザの拡大が発生しない場合
- ミッションの拡大が発生しない場合

である場合，既にユーザ/ミッション・ローカルレコードはノード内に存在するため，レコードを作成する必要がない．そのため，処理自体を省くことができ，移送コストを削減できる．

処理を省くことができるかどうか，つまり，レコードの有無は，ユーザ/ミッション・ローカルテーブル (ハッシュ) を検索することで判明する．

以下では，処理を省くことができず，レコードを作成する場合の処理について説明する．

ユーザ・ローカルレコード

- レコード領域のアロケート
- 現在の時刻の記録
- history window の作成
  history window を作成する．history window のサイズは 2460Byte あり，初期化にかかるコストが大きい．そこで，初期化済みの history window を多数用意しておき，レコード作成時は，そのフェッチのみ行い，コストを削減する．
- ノード内チケット総量への加算
  移送先ノードに新しくユーザが加わったことになるので，ノード内のチケット総量に，このユーザが持つチケットを加算する．

ミッション・ローカルレコード

- レコード領域のアロケート
- レコード内のユーザ ID，ミッション ID を設定
- ユーザ・ローカルレコードのミッションチケットの総量に，このミッションのチケットを加算

その後，ユーザ・ローカルレコードへのリンク (図 4(4) のリンク) および，ユニット・レコードからのリンク (同図 (3) のリンク) を作成する．

3.4.5 ページ・テーブルの処理

移送されてきたユニットのために，ページ・テーブルの新規作成を行う．また，移送ユニットが属するミッションがノード内に存在する場合，ページ・テーブルの共有設定を行う．

ページ・テーブルの作成

移送されてきたユニット用に，ページ・テーブルを新規に作成する．ここでも，ページ・テーブルの初期化にかかるコストを削減するために，初期化済みのテーブルを多数用意しておく．作成時には，これをフェッチしてテーブルを構成する．なお，テーブルの初期化は，テーブルの返却時およびブート時に行う．

ミッション内共有ページテーブル

同じミッションのユニットは，ノード内でページ・テーブルの共有領域部分を共有する．移送されてきたユニットのミッションが既にノードに存在する場合，ページ・テーブルの共有領域部分を共有するように設定する．

最後にスレッドを実行キューに繋げ，実行可能状態にする．

3.5 ページ移送

3.5.1 移送時の転送

ユニットの移送に伴い，多量のページを転送しなければならない場合，転送時間が増大し，移送上のボトルネックとなる．これを避けるため，Colonia では，移送時には必要最低限のページのみ転送し，残りのページは移送先で参照される度に転送する方式 (Copy on reference 方式)[8] を採用する．

移送時に転送するページは，参照頻度で選別する．すなわち，スタックが配置されている固有領域のページ，参照頻度が高い共有領域のページなどが最初に転送される．移送時の転送ページ数は，10 ページ程度と考えている．

3.5.2 残りのページの転送

Colonia では，移送時には最低限のページしか転送しないため，転送しなければならないページが移送元に残る．これらのページは，以下の方法で移送先に転送される．

- 移送先で必要になった時に転送
  移送先で，未転送のページへアクセスが発生した場合，システム・ユニットが移送元へページの転送要求を出す．移送元のシステム・ユニットがこの要求を受信し，ページの転送を行う．ページ転送ごとに起動オーバーヘッドがかかるため，移送時に一括してページ転送を行う方式と比べて，トータルでの転送時間は長くなる．これを軽減するために，ページのプリフェッチといった技術が考えられる．
- 移送元のシステム・スレッドによる転送
  Colonia では，各ノードごとに，残されたページを転送するためのシステム・スレッドを起動しておく．このシステム・スレッドは，適宜スケジューリングされ，移送先へページの転送を行う．

4. 評　　価

Colonia のユニット移送機構をエミュレータ上に実装し，移送元および移送先での処理にかかる，ソフトウェア・オーバヘッドを測定した．エミュレータの実行には Sun Microsystems 社の WS，Ultra-30(UltraSPARC-II 296MHz) を使用した．コンパイラには gcc，最適化オプションは -O2 を用いた．tick レジスタを用いて，実行サイクル数を測定した．

4.1 評価と結果

4.1.1 移送するユニット

評価結果を表 1に示す．数値はサイクル数である．なお，計算量の参考のために，キャッシュヒット率がほぼ 100%の理想値も併記した．理想値は，同一処理を繰り返し行い，一回あたりの平均として算出した．

移送先に移送ユニットのミッションおよびユーザが既に存在する場合 (ユニット 1) と，共に存在しない場合 (ユニット 2) を用意した．この二つのユニットでは，(3)(4)(6) の処理が異なる．両ユニット共，セグメントを 16 個 (合計 448Byte) 持つ．

4.2 結果の評価

評価結果より，移送時のソフトウェア・オーバヘッド

表1　サイクル数

移送元処理	ユニット 1	ユニット 2
(1) スレッド detach	1141 (124)	
(2) スレッド pack	1842 (593)	
(3) ユニット pack	334 (131)	
小計	3317 (848)	
移送先処理	ユニット 1	ユニット 2
(1) スレッド unpack	2338 (637)	
(2) ユニット unpack	866 (134)	
(3) ユーザ・ローカルレコード	442 (40)	898 (109)
(4) ミッション・ローカルレコード	311 (41)	1630 (112)
(5) ページ・テーブル	761 (133)	
(6) ミッション共有 ページ・テーブル	1691 (366)	-
小計	6409 (1351)	6493 (1125)

は，移送元ノードで約 11μs，移送先ノードで約 21μs であった．事前に初期化処理を行うことで (3)(4) のコストが小さくできたため，ユニット 1 とユニット 2 で処理時間がほぼ同じとなった．しかし，ユニット 1 は共有ページがある程度揃っているため，実行を再開した後の起ち上がりユニット 2 よりも速い．

4.3 考　　察

実際の移送では，ここで示したサイクル数に加えて管理データの転送時間，および 1 ページ分のデータの転送時間がかかる．管理データは，最大で 1KB 程度，ページは 4KB である．よって，これらの転送にかかる時間は，バンド幅が 1Gbps のネットワークでも，最低 8μs，および 32μs かかる．しかし，ページの転送は，移送処理と依存関係がないためオーバーラップさせることが可能である．したがって，ページを分割転送するなどして，転送処理を可能な限り移送処理とオーバーラップさせる工夫を行えば，移送処理の大半を転送処理で隠蔽できる．以上の考察により，移送全体を最短 50μs 程度で実現できると考えている．

5. ま と め

本稿では，コンピュータ・コロニーの並列アクティビティであるユニットの，移送コストの評価を行った．我々は，効率的な動的負荷分散を行うためには，高速なアクティビティ移送機構が不可欠であると考え，移送コストができるだけ小さくなるように移送機構を実装した．この移送機構をエミュレータ上で実装し，移送の際のソフトウェア・オーバヘッドの評価を行ったところ，移送元での処理に 3317 サイクル，移送先での処理に 6409 サイクルかかった．

今後は，今回の評価で得られた移送コストを基に，データ転送を含めた移送処理の最適化を行い，更なるコスト低減を図る．さらに，そのモデルをエミュレータ上に実装し，グローバル・スケジューラの評価を行っていく予定である．

謝　辞

なお，本研究の一部は文部省科学研究費補助金 (基盤研究 (B)(2) 課題番号 12480072 ならびに 12558027) による。

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,97-OS-34,1997
2) 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,pp55-60,199
3) Kai Hwang, Hai Jin, Edward Chow, Choli Wang, Zhiwei Xu : Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space, IEEE Concurrency,pp.60-69,Jan-Mar,1999
4) 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
5) Damien Le Moal, 他 分散システムにおける Fair Share プライオリティスケジューラ, 情報研報 99-OS-80(SWoPP 下関'99),1999
6) 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
7) 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 99-CPSY-53(SWoPP 下関'99), pp.49-56
8) Zayas, E. R. 1987. Attacking the process migration bottleneck. In 11th ACM Symposium on Operating Systems Principles (1987), pp.13–24