社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－OS－96 (5)
2004／6／17

データセンタ環境に適した TCP 無切断プロセスマイグレーションの実現

高橋 雅彦　菅原 智義

日本電気株式会社 システムプラットフォーム研究所

要旨

24 時間 365 日運用可能なサービスを実現するためには、サービスを実行しているサーバの定期メンテナンスなどのときにサービスを透過的に移動できなければならない。そこで我々は、TCP セッションが切断されないプロセスマイグレーション方式を提案する。提案方式はオーバヘッドが低く、クライアント側の OS やアプリケーションの改造を必要としない方式である。したがって、既存のサービスにそのまま適用でき、透過的なサービス移動が実現出来る。これを実現するために、プロセスマイグレーションを同一サブネット内に特化し、OS レベルのプロセスマイグレーション実装と、プロセスごとに割り当てられた仮想 IP アドレスを用いた。本方式を実装し、仮想 IP アドレスを使用することによるオーバヘッドを測定したところ、約 0.15% と極めて低いことを確認した。更に、Apache や HelixDNAServer といった実アプリケーションも、クライアントとの通信が切断されることなく移動可能であり、実用性を検証した。

Implementation of Process Migration with TCP Session Continuity suitable for Data Center Environments

Masahiko Takahashi and Tomoyoshi Sugawara

System Platforms Research Laboratories, NEC Corporation

Abstract

To improve a service availability operating 24 hours a day for 365 days a year, it is important to support a service continuity mechanism independent on execution hardware and network environment. To achieve it, we propose a method of process migration with TCP session continuity. A virtual IP address is provided for each service process, and is also migrated when the process is migrated to another computer. Since our method enables a server process migration without changing its communication address, there is no need to modify client-side systems. Our prototype's overhead on usual network communications is extremely low about 0.15% with additional 19 virtual IP addresses on a NIC. Furthermore, we have validated that real applications such as Apache and HelixDNAServer are able to be migarated without lost connections.

1 はじめに

インターネットは今や、重要な社会インフラの一部となりつつある。web サービスを始めとした各サービスのサーバ群は、24 時間 365 日の運転を切望されながらも、現実的にはハードウェアの定期的なメンテナンスが必要であり、本当の意味でサービスを継続できているとは言い難い。例えば、金融系のインターネットサービスでは、週末の夜に定期メンテナンスをしているところが多い。

現在の web サービスは、リクエスト処理に要する時間が比較的短いので、負荷分散装置 (ロードバランサ) でサーバを意図的に変更することが出来る。しかし、処理時間が長いリクエストの場合は、そのリクエストが終了するまではサーバを変更することが出来ない。また、たとえ接続が切断されたとしても、特に web サービスはトランザクションをベースに処理しているので、必要であればユーザがブラウザの「再読み込み」を実行するという運用面でカバーしている面もある。しかしこれは、システムが未熟なのをユーザに押しつけているに過ぎない。

24 時間 365 日のサービスの継続性を確立するためには、メンテナンスなどのときにも接続が切断されたりすることなく透過的にサービスを移動させられることが重要である。

このようなサービスを移動する手法の 1 つに、プロセスマイグレーション [6] がある。プロセスマイグレーションは、計算機上で実行中のプロセスの実行状態をそのまま別の計算機に移動する技術である。システム層で実現されるので、サービスを提供している既存のアプリケーションに対して特に改造を行なうことなく適用可能な手法であり、適用範囲が非常に広い。

我々は、特にサブネット内の環境の場合に、TCP セッションが切断されないプロセスマイグレーション方式を提案する。これによって、クライアントからのリクエストを処理している最中でも、サービスや通信コネクションを継続したままサービスを他の計算機に移動させることが可能となる。

提案方式は、新たな通信プロトコルなどを導入することなく実現しており、クライアント側でプラグインやドライバなどの改造を必要としない。また、通常の通信に与えるオーバヘッドは極めて低い。

我々はこれを Linux 2.4.18 に実装し、その性能を測定したところ、通常の通信に与えるオーバヘッドは 0.15% 程度であった。提案方式の実用性の面では、現在広く使われている web サーバの Apache や 動画配信サーバの HelixDNAServer がプロセスマイグレーション可能であり、実用性があることを確認した。

以下の本稿の構成は、まず 2 節で従来のサービス移動手法とその課題について述べる。3 節で提案方式を説明したのち、4 節で Linux 上における実装について述べる。そして 5 節でプロセスマイグレーションの関連研究を述べ、6 節でまとめる。

2 従来のサービス移動の手法

定期メンテナンスや負荷分散の目的のために、計算機 (以下、ノード) 上で実行中のサービスを他のノードに移動させる主な手法にフェイルオーバとプロセスマイグレーションがある。

フェイルオーバは、サービスを提供しているプロセスが終了した後に、他のノードでそのサービスを起動させることでサービスの移動を実現する技術である。サービス移動のためにプロセスを終了させる際には通信中のコネクションは切断されるので、リクエスト処理を続けるためにはクライアントから再接続しなければならない。したがって、フェイルオーバに対応させるための仕組みがクライアントに必要となり、既にあるサービスへの適用という意味で適用範囲が限られる。

プロセスマイグレーションは、ノード上で実行中のプロセスを実行中の状態のままで他のノードに移動させる技術である。実行状態のままで移動するので、プロセスの終了及び再起動は行なわない。プロセスマイグレーションの技術は分散 OS (Operating System) の研究に由来する。分散 OS はその名の通り分散処理を主目的とした OS であり、その手段としてプロセスマイグレーションが開発され利用されていた。例えば、Amoeba[7], Mach[1], MOSIX[2], Sprite[3] などが挙げられる。

これらの中には通信を維持したままプロセスを移動できる OS もあったが、移動できる対象プロセスが限られていたり、TCP/IP(Transport Control Protocol/Internet Procotol) ではなく独自の通信プロトコルであったりして、汎用性が低く、やはり適用範囲が限られていた。

最近の実装として、OpenSSI cluster[8] もプロセスマイグレーションを実装した OS である。OpenSSI には、プロセスが最初に起動したノード (ホームノード) に依存するという課題がある。例えば、システムコールの種類によっては、システムコールをホームノードに転送して実行することがある。このため、プロセスマイグレーション後でもそのノードをメンテナンス出来なくなる。これはまた、通常の実行時のオーバヘッドでもある。

このように、従来手法では透過的なサービス移動が実現できていない。ここでいう透過的とは、アプリケーションやユーザが特に対応することなくシステム層だけで機能を実現することを指す。透過的なサービス移動を実現するにあたっての従来手法の問題点は、

- クライアント側での対応が必要
- 独自プロトコルである
- ホームノード依存性がある
- オーバヘッドがある

ということが挙げられる。

3 透過的なサービス移動の実現

我々は、従来手法の問題点に対して、オーバヘッドが低く、TCP セッションが切断されないプロセスマイグレーション方式を提案する。提案方式は

1. プロセスマイグレーションはサブネット内に限る
2. プロセスごとに IP アドレスを割り当てる
3. プロセスマイグレーションを OS レベルで実現する

ことを特徴とする方式である。1 によって独自プロトコルを使用せず、2 と 3 によってクライアント側での対応を不必要とする。また、1, 2 及び 3 によって、オーバヘッドを小さくする。ホームノード依存性に関しては、プロセス間通信を行なうプロセスはまとめて移動させるなど、一定の条件を課すことで回避する。

本節では、プロセスごとに割り当てる IP アドレスの説明を中心に提案モデルの全体の流れを説明し、そして OS レベルでの実現について述べる。また、これを実現するために必要な DNS での対応と、プロセスマイグレーション時にパケット落ちを起こさないための仕組みについて述べる。

3.1 提案モデル

TCP 無切断プロセスマイグレーションを実現するために、サービスを行なうプロセスごとに IP アドレスを割り当て、クライアントとの通信にはこの IP アドレスを使用する。そして、プロセスの移動時には、割り当てられた IP アドレスも一緒に移す。この割り当てられた IP アドレスを仮想 IP アドレスと呼ぶことにする。

プロセスごとに仮想 IP アドレスを割り当てるのは、プロセスと仮想 IP アドレスのバインドを維持するためである。これは、IP 通信を行なうときのソケットには基本的に IP アドレスを明記するので、透過性なサービス移動を実現するためには IP 層以上を書き換えてはいけないからである。バインドを維持しているので、従来の IP 通信が IP アドレスをノードに固定的にしていたのに対して、提案方式ではプロセス (もしくはサービス単位) を通信相手として識別することになる。

この仮想 IP アドレスは、ノードの NIC (Network Interface Card) につけられた IP アドレスとは異なるが、同じサブネット内の IP アドレスである。OS 内では、複数の IP アドレスを 1 つの NIC (MAC アドレス) にマッピングすることが出来る。これは、NIC に IP アドレスがいくつマッピングされていようとも、ブロードキャストの ARP(Address Resolution Protocol) リクエストを受け取った OS が自ノードの IP アドレスかどうかを判断すればいいからである。更に、このマッピングは、単に ARP リクエストに OS が返答するかという問題であり、プロセスからも認識できないので、必要に応じてマッピングを変えるのも難しいことではない。

したがって、プロセスマイグレーション時には、移動元ノードでの仮想 IP アドレスの IP-MAC マッ

ピングを削除し、移動先ノードに新しい IP-MAC マッピングを設定する。このように、IP-MAC マッピングだけを変更し、プロセスと仮想 IP アドレスのバインドは変更しないので、プロセスから見て透過性が実現される。

このマッピングは、仮想 IP アドレスがない通常の IP 通信でも行なわれているものであり、通信時に到着したパケットに対応するソケットを検索する際には IP アドレスやポート番号でハッシュを引くので、仮想 IP アドレスを追加したことによるオーバヘッドはほとんど発生しない。

図 1 は、提案方式とネットワークプロトコル階層の関係を表したものである。ノード A 上のサービス X を提供するプロセスは、仮想 IP アドレス V にバインドされる (現実にはこのときポート番号も指定するが、本稿ではポート番号に関する記述は省略する)。OS 内では仮想 IP アドレス V と NIC の MAC アドレス P との間のマッピングを設定する。

プロセスマイグレーションの際には、ノード A (IP アドレス a) からノード B (IP アドレス b) に対してプロセス情報が転送される。ノード A では仮想 IP アドレス V と MAC アドレス P のマッピングを削除し、ノード B では仮想 IP アドレス V と MAC アドレス Q のマッピングを設定する。このように、OS 内ではマッピングを削除・設定するだけである。

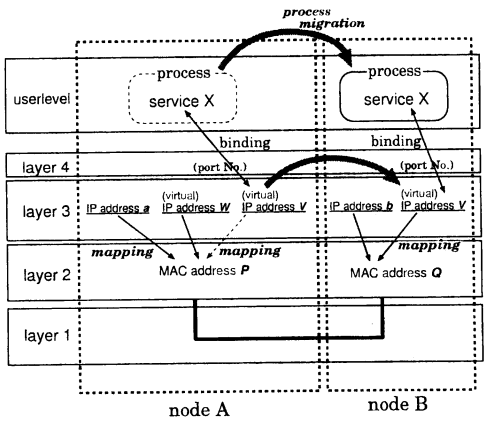

図 1: 提案方式とネットワーク階層の関係

3.2 OS レベルでの実現

TCP 無切断プロセスマイグレーション機能の実現は、OS レベルでの実現が必要である。これは、TCP パケットのシーケンス番号などの情報を正しく引き継いだり、TCP バッファにたまっているデータを透過的に移動させるために必要となる。従来より、プロセスマイグレーションの実装は OS レベルとユーザレベルの 2 通りがあったが、OS レベルで実現することで、透過的なサービス移動はサーバ側だけの実装で実現でき、クライアント側での対応は不必要になる。したがって、提案方式の適用範囲は広い。

3.3 DNS での対応

提案方式では、クライアントの通信相手はノードではなくサービスである。したがって、DNS にサービスと仮想 IP アドレスの変換機能を追加する。つまり、今までの DNS は「ノードと IP アドレスの対応表」であったが、提案方式ではそれに加えて「サービスと仮想 IP アドレスの対応表」が追加される。ただし、DNS の機能を拡張するのではなく、単に、サービス用に仮想 IP アドレスのエントリをするだけである。したがって、既存の DNS の機能だけで実現可能である。

3.4 パケット落ちの防止

プロセスマイグレーションに時間がかかると、その途中で到着したパケットはどのノードも受け取らない状況が発生する。この状況を避けるために、プロセスマイグレーションの実行中は、プロセスの移動元と移動先の両方のノードで、仮想 IP アドレスを両ノードの MAC アドレスにマッピングする。そして、プロセス移動先ノードの OS はクライアント (またはルータやゲートウェイなど) に対して、ARP キャッシュの更新 (もしくは無効化) をブロードキャストする。この更新プロトコルは独自プロトコルではない。

こうすることで、クライアントから送られてくるパケットをどちらのノードも受け取らない状況を回避する。ただし、仮想 IP アドレスが両ノードに存在するときに、他のノードから ARP リク

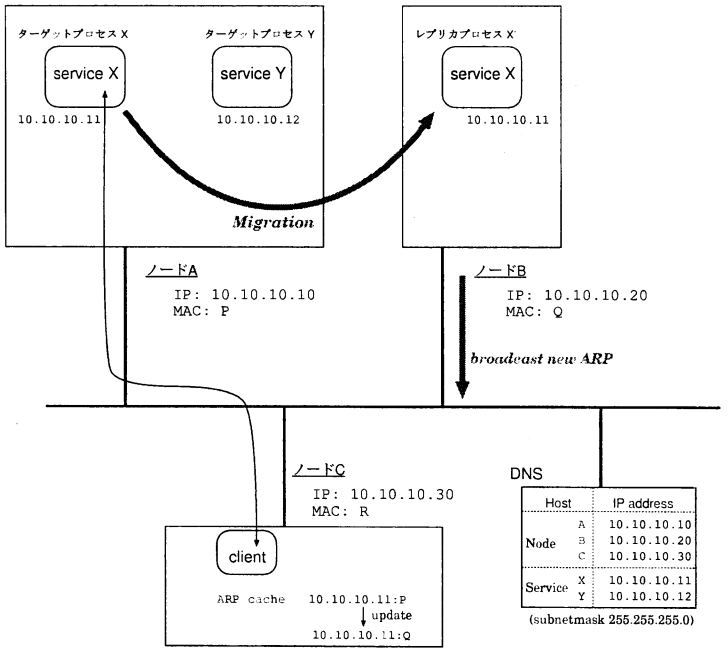

図 2: 提案方式の動作例

エストが送られると両ノードが返答を送り返す状況が発生する。これを避けるために、移動元ノードは、ARP キャッシュの更新のブロードキャストを受信したら、その後はこの仮想 IP アドレスの ARP リクエストには答えないようにする。ARP リクエスト以外の通信パケットが送られて来たら、プロセスマイグレーションが完了するまではバッファに蓄えておき、プロセスマイグレーション後に、バッファ内のデータを移動先ノードのものと結合する。

3.5 動作例

図 2 は、提案方式のプロセスマイグレーション時の動作例である。3 つのノードがあり、サーバとしてノード A (IP アドレスは 10.10.10.10、MAC アドレスは P) とノード B (IP アドレスは 10.10.10.20、MAC アドレスは Q) が、クライアントとしてノード C (IP アドレスは 10.10.10.30、MAC アドレスは R) がある。ノード A 上にはサービス X とサービス Y が動作しており、それぞれのプロセスには 10.10.10.11 と 10.10.10.12 の仮想 IP アドレスが割り当てられている。これらの IP アドレスと仮想 IP アドレスは、全て DNS に登録されている。いずれも 255.255.255.0 のサブネットマスク内の IP アドレスである。

ノード C 上のクライアントは DNS を引いてサービス X に接続し、ノード C には、ARP キャッシュとしてノード A の MAC P とサービス X の IP アドレス 10.10.10.11 が関連付けられている。ノード C 上のクライアントはノード A 上のサービス X と通信中なので、MAC アドレス P に対してパケットを送信している。

この状況で、サービス X がノード B にプロセスマイグレーションしたとする。IP アドレス 10.10.10.11 はノード A 上から削除され、ノード B に設定される。ノード B は新しい ARP キャッシュのエントリとして「IP アドレス 10.10.10.11 と MAC アドレス Q」の関連付けをブロードキャストする。新しい ARP キャッシュを受け取ったノード C は ARP キャッシュを更新し、次のパケットからは MAC アドレス Q に対して送信する。

4 実装と評価

今回の実装には Linux を使用した。Linux は分散 OS でもなく、プロセスマイグレーションに対応してもいないので、最初にプロセスマイグレーションの実装に関して述べる。そして、TCP 無切断対応のために必要な、プロセスと仮想 IP アドレスのバインディング、及び、仮想 IP アドレスと MAC アドレスのマッピング、そして、Linux での実装の制約としてカーネル内時間の調整に関して述べる。また、性能評価についても述べる。

4.1 プロセスマイグレーションの実装

提案方式のプロセスマイグレーションは、Linux のカーネルモジュールとユーザレベルのデーモンとで実装している。カーネルモジュールは、OS 内の情報を採取、及び、設定するために使用する。デーモンはノード間の通信を行ない、移動させるプロセスの情報を送受信している。

プロセスマイグレーションを実現するためには、大きく分けて、プロセスが使用している資源の OS 内情報と、ユーザレベルのアドレス空間 (メモリ) の 2 つの情報が必要である。以下、これらについて述べる。

プロセスマイグレーションの実行時には、プロセスの task 構造体をはじめとして、プロセスが使用している資源 (シグナルやファイルディスクプタなど) の情報を OS から採取し、移動先ノードで空のプロセスを作成した後にその資源情報で再構築していく。OS のバージョンが同じであれば、基本的には OS 内のプロセス情報を読み出してそれをそのまま転送し、再び設定するだけである。特に、TCP などのソケットに関しては、OS 内で溜っているバッファや TCP の 3way handshake などの情報も含めて全ての情報を採取し、移動先ではそのままの状態で復帰させる。

メモリ (アドレス空間) のデータは、物理アドレスではなく、仮想アドレスで読み出す。したがってプロセスのページテーブルは転送せず、仮想アドレス空間のデータを転送しながらページテーブルを再構築している。このため、移動させるプロセスが使用している物理アドレスには制約を受けない。また、共有メモリにも対応しているので、マルチスレッドや fork したことによるマルチプロセス構成のプロセスも移動可能である。

なお、現在の実装には 2 つの制約がある。1 つ目は、共有ディスク (NAS など) の存在を前提としていること。共有ディスクがあるので、プログラムのテキスト部は転送しない。2 つ目は、ローカルデバイスにアクセスしているプロセスは移動出来ないということ。したがって、/proc をアクセスしているプロセスも移動できない。また、ローカルの HDD にテンポラリファイルやロックを作成しているプロセスも移動出来ない。これらを共有ディスク上に作成すればそのプロセスは移動可能である。

4.2 プロセスと仮想 IP のバインド

プロセスごとに割り当てられた仮想 IP アドレスの情報は、カーネル内の task 構造体に専用のエントリを作成して保存しておく。例えば、プロセスが TCP/IP 通信で connect する際には、connect システムコールをフックして、ソケットの source IP に仮想 IP アドレスを設定する。通常の OS では、connect する前にソケットに意図的に bind しない限り、NIC に割り当てられた IP アドレスを source IP とする。それではプロセスマイグレーション後に透過性が実現できなくなるので、提案方式では強制的に source IP をプロセスマイグレーション対象プロセスの仮想 IP アドレスにする。これによって、プロセスマイグレーション後も、同じ IP アドレスで通信できる。

4.3 仮想 IP と MAC のマッピング

MAC アドレスと IP アドレスのマッピングは、普段、NIC に仮想 IP を追加する方法

`ifconfig eth0:1` *IP_addr* `netmask .. up`

を実行する。削除する方法も同様である。

4.4 カーネル内時間の調整

Linux 上でプロセスマイグレーションを実装する場合は、両ノードの jiffies 値 (カーネルが起動してからの 10ms 単位の tick 値) をそろえておかなければならない。これは、TCP のセッションを継続

するために必要である。TCP の timestamp は論理時間であり、Linux ではカーネル内の変数 jiffies の値を timestamp として利用している。jiffies 値がそろっていないノード間でプロセスマイグレーションを行なうと、プロセスマイグレーション後、クライアントへ送られるパケットの timestamp がずれる。クライアントの OS によっては、TCP プロトコル処理で timestamp が大きくずれたパケットを受け取った場合に、そのパケットを不正なものであると判断してパケットを破棄する実装もあり得る。そのため、各ノードの jiffies 値をきちんとそろえて、TCP セッションを継続させる。

4.5 性能評価

提案方式の性能評価をするため、以下の実験を行なった。

- 仮想 IP が増えたときのオーバヘッド
- プロセスマイグレーションの性能

また、提案方式の実用性の検証として以下の実験を行なった

- 実アプリケーションの移動性能

測定環境は表 1 の通りである。

表 1: 測定環境

CPU	Xeon 2.4GHz (Hyper-Threading 不使用)
メモリ	4GB (DDR200 SDRAM, FSB 400MHz)
バス	64bit/66MHz PCI-X
NIC	Intel PRO/1000 XT
OS	RedHat Linux 7.3 (2.4.18-3)
スイッチ	Extreme Summit7i

4.5.1 仮想 IP アドレスのオーバヘッド

NIC に設定する仮想 IP アドレスを増やしたときに通常の通信に与えるオーバヘッドを測定するため、クライアントからサーバに 400MB のデータを転送するのに要する時間を測定した。

測定の結果、NIC の IP アドレスだけでデータを転送したときと比較して、19 個の仮想 IP アドレスを設定して 19 個目の仮想 IP アドレスにデータを転送した際のオーバヘッドは、5 回の平均で約 0.15% であった。実際の使用状況では 1 つのノード上で実行されるサービスは数個程度と予想されるので、仮想 IP アドレスを追加したことによるオーバヘッドはほとんど無視できる。

4.5.2 プロセスマイグレーションの性能

提案方式のプロセスマイグレーションの基本性能を測るため、プロセスが使用するメモリ量を変化させて、そのプロセスを移動させるのに必要な時間を測定した (表 2)。比較対象として、OpenSSI (Linux 2.4.14) でのプロセスマイグレーション時間を測定した。移動させたプロセスは、指定したメモリ量を使用した後はずっと sleep するプログラムを実行したものである。測定結果より、提案方式のプロセスマイグレーションは OpenSSI より 10〜30% ほど性能が高いことが分かった。

表 2: プロセスマイグレーション測定結果

プロセスの使用メモリ	マイグレーション時間		性能向上率
	提案方式	OpenSSI	
10 MB	177.1 ms	197.8 ms	+10.46%
50 MB	664.0 ms	968.8 ms	+31.46%
100 MB	1,270.9 ms	1,934.6 ms	+34.31%
200 MB	2,481.2 ms	3,892.4 ms	+36.26%
300 MB	3,691.7 ms	5,842.5 ms	+36.81%
400 MB	4,922.9 ms	7,816.1 ms	+37.02%
500 MB	6,133.0 ms	9,750.5 ms	+37.10%

4.5.3 実アプリケーションの移動性能

提案方式の実用性を評価するために使用した実アプリケーションは、web サーバの Apache (1.3.27) と動画配信サーバの HelixDNAServer (9.0.4.25) である。これらをプロセスマイグレーションした結果を表 3 に示す。Apache も HelixDNAServer も TCP セッションが切断されることなく移動できた。所要時間に関しては、現在の実装では何回にも分けてプロセス情報を転送しているため、所要時間がプロセス数や資源数に依存してしまう。したがって、14 スレッドある HelixDNAServer は転送時間が長い。今後は、資源数やプロセス数が多くとも 1 回で全てのプロセス情報が転送できるように最適化する予定である。

表 3: 実アプリのマイグレーション所要時間

対象アプリ	所要時間	プロセス数
Apache	0.26 秒	3 プロセス
HelixDNAServer	1.41 秒	14 スレッド

5 関連研究

Zap[9] は、OS に仮想化層を作成して、TCP セッションを維持したままプロセスマイグレーション可能にする実装である。このために、VNAT[10] を利用しており、OS 内に IP-IP 変換のマッピングを持っている。通信しているクライアントとサーバの両方でこのマッピングを持ち、プロセスマイグレーション後には独自プロトコルを用いてマッピング情報を更新する。この IP-IP 変換機能は、通信しているサーバ及びクライアントの両方での実装が必要である。VNAT は特にモバイル環境への適用も想定しており、広域ネットワークでプロセスマイグレーションが出来るようにと考えられた汎用的な手法であるが、IP-IP 変換のオーバヘッドが 2〜7% 程度 [10] 存在する。

ROCKS[11] は送信データをバッファリングして管理するユーザレベルのライブラリであり、通信が切断された場合は自動的に再接続する。そのため、フェイルオーバや接続中の通信が切断するプロセスマイグレーションなどの弱点を補うことが可能である。ただし、サーバだけでなくクライアントでの実装も必要となる。

分散 OS の研究である Amoeba[7] は、分散 OS 間の通信に特化した FLIP[5] という独自の通信プロトコルを使用している。このため、TCP/IP のような汎用的な通信は出来ないが、FLIP の枠組みの中では通信セッションの移動が可能である。

6 まとめと今後の課題

本稿では、サービスの継続性を高めるために TCP セッションの切断が起きないプロセスマイグレーション方式を提案した。提案方式は既存のネットワークプロトコルの枠組みに沿って実装されており、クライアントが特に対応しなくてもサービス移動が実現できる。また、通常の通信に与えるオーバヘッドは極めて低い。本稿では同一サブネット内でのプロセスマイグレーションに特化しているが、[4] のセッションマイグレーションでは、我々の手法を VLAN とともに使ってルーティングすることで、広域のプロセスマイグレーションを実現している。

今後は、プロセスマイグレーションの時間を短縮する方式として更新メモリ (dirty page) だけを転送する仕組みを実装する予定である。更に、応用として、ノードの突発的な障害発生 (ダウン) 時のリカバリを実現する予定である。

謝辞　本研究は、NEDO 技術開発機構 基盤技術研究促進事業 (民間基盤技術研究支援制度) の一環として委託を受け実施している「大規模・高信頼サーバの研究」の成果である。

参考文献

[1] M. Accetta, et al., Mach: A New Kernel Fundation for UNIX Development. USENIX Summer Conference, 1986.

[2] A. Barak, et al., MOSIX: An Integrated Multiprocessor UNIX. USENIX Winter Technical Conference, February 1989.

[3] F. Douglis, et al., Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software - Practice and Experience*, Vol. 21, No. 8. August 1991.

[4] 今井 他, 広域分散データセンタ間でのサービス無停止障害回避方式. 電子情報通信学会 NS 研究会, 2004 年 3 月.

[5] M. Kaashoek, et al., FLIP: An Internetwork Protocol for Supporting Distributed Systems. *ACM Transactions on Computer Systems*, Vol. 11, No. 1, February 1993.

[6] D. Milojicic, et al., Process Migration. *ACM Computing Surveys*, Vol. 32, No. 3, September 2000.

[7] S. Mullernder, et al., Amoeba – A Distributed Operating System for the 1990's. *IEEE Computer*, Vol. 23, No. 5, May 1990.

[8] Open Single System Image Cluster Project, `http://ssic-linux.sourceforge.net/`.

[9] S. Osman, et al., The Design and Implementation of Zap: A System for Migration Comuting Environment. *Symposium on Operating Systems Design and Implementation*, December 2002.

[10] G. Su, et al., Mobile Communication with Virtual Network Address Translation. Technical Report CUCS-003-02, Columbia University, February 2002.

[11] V. Zandy, et al., Reliable Network Connections. ACM MobiCom, September 2002.