2005－OS－99（3）
2005／5／25

Web サーバクラスタのための TCP マイグレーションを用いたリクエスト分散機構

高橋 雅彦　小比賀 亮仁　菅原 智義　田中 淳裕

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

今後の Web サーバシステムには十分なスケーラビリティを有し、かつ柔軟なリクエスト分散機構を持つシステムアーキテクチャが必要である。本稿では、TCP マイグレーション機構を用いた新しいアーキテクチャを提案する。提案方式は、プライベートネットワーク内に構築されて、クライアントからの HTTP リクエストをまずディスパッチャが受け取り、その後、Web サーバに TCP セッションをそのままの状態で転送してリプライを送信する。通信中のセッション毎にユニークな仮想 IP アドレスを利用しているため、セッションを転送してもゲートウェイにおいて書き換えるパケットのアドレスは変更しなくてよい。更に、シーケンス番号などを書き換える必要もない。また、ディスパッチャとパケット転送を物理的に分離しているので、ディスパッチャではパケット転送が行なわれず、ディスパッチャに障害が発生しても、既に Web サーバに転送されたセッションには影響が及ばない。TCP マイグレーション機構を Linux カーネルに実装し、その性能評価を行なったところ、mini_httpd と wget を用いた実際のオーバヘッドは、セッション転送を行なわない場合と比べて 1 ミリ秒程度の増加であった。また、仮想 IP アドレスを多数用いることによるオーバヘッドは、事実上、無視できるレベルであった。

A New HTTP Request Distribution Architecture using TCP-migration for Cluster-based Web Servers

Masahiko Takahashi, Akihito Kohiga, Tomoyoshi Sugawara, and Atsuhiro Tanaka

System Platforms Research Laboratories, NEC Corporation

Abstract

A cluster-based server system is a developing technology that could achieve high scalability by using some dispatchers, such as L4 or L7 switches, to distribute requests from clients appropriately. Many recent Web server systems have been developed as cluster systems. This paper proposes a novel architecture based on *TCP-migration* mechanism that provides complete redirection (displacement) of a TCP connection between a dispatcher and Web servers. The key idea is physical separation of L7 switch functionality: forwarding and dispatching mechanisms. With NAT mechanism on the L4 switch and sophisticated virtual private IP address management on the servers, the dispatcher is free to relay or translate both in-bound and out-bound TCP packets.
We have designed and implemented this mechanism on Linux 2.4 kernel and evaluated its performance. The experimental results show that the overhead for handling virtual IP addresses is almost negligible. Furthermore, by executing `mini_httpd` as a Web server and `wget` as a client, the overhead of TCP-migration is approximately 1 ms, regardless of the reply size, on 3.6 GHz Xeon machines.

1 はじめに

E ビジネスなど、WWW (World Wide Web) 技術に基づいたインターネットサービスは今や、我々の日常に必要不可欠な社会インフラとなっている。したがって、これらのサービスを実行しているシステムは、サービスの発展とその潜在的需要に備えて拡張性 (スケーラビリティ) と可用性 (アベイラビリティ) とを高める必要がある。

スケーラビリティを高める一手法としてクラスタシステムがある。クラスタシステムでは、数多くのサーバをプライベートネットワーク内に立ち上げ、クライアントからのリクエストをそれらに分散する。リクエスト分散にはレイヤ 4 スイッチ (L4 スイッチ) やレイヤ 7 スイッチ (L7 スイッチ) といった装置が使われることが多い。しかしながらこれらの装置は、リクエストをバックエンドの Web サーバに振り分けるディスパッチ機能と、Web サーバとクライアント間の通信のパケット転送機能を同一装置内に備えているために柔軟性が不十分で、例えば、装置に故障が発生するとディスパッチだけでなくて既にサーバから送信されているリプライの通信にも影響を与えることがある。

我々はこのようなクラスタシステムに適した新しいリクエスト分散アーキテクチャを提案する。特に Web サーバシステムに適している。提案方式は、ディスパッチ機能とパケット転送機能を物理的に分離し、ディスパッチ機能はサーバ上で動作させ、パケット転送機能は L4 スイッチなどのハードウェアなどを用いて高速に処理する。

このように、提案方式は L4 スイッチとディスパッチャと Web サーバクラスタで構成される。クライアントとの最初の TCP (Transmission Control Protocol) セッション確立はまずディスパッチャが受け付け、HTTP (HyperText Transfer Protocol) リクエストヘッダなどをスキャンして適切なサーバを選択した後に、クライアントとの TCP セッションを完全にそのサーバにリダイレクトする。リダイレクト後は、クライアントとサーバが直接通信を行うため、ディスパッチャがパケット転送を行う必要はない。その代わり、L4 スイッチの NAT (Network Address Translator) 機能を用いて IP (Internet Protocol) 層でのアドレス書き換えを行なう。したがって、仮りにディスパッチャが高負荷になってレスポンスが悪化したとしても、既にリダイレクト済みのセッションに対して影響は及ばない。

提案方式を実現するために、我々はソケットマイグレーション機構の実装と仮想 IP アドレスを用いたアドレスマネージメントとの 2 つを行なった。前者は Linux カーネル (バージョン 2.4) に実装し、TCP セッションの詳細な状態 (シーケンス番号やソケットバッファなど) をそのままの状態で別サーバに転送して TCP セッションを継続する。後者に関しては、各 TCP セッション毎に仮想 IP アドレスを割り当て、TCP セッションがサーバ間を転送される際には仮想 IP アドレスも一緒に移動させる。このとき、スムーズな通信再開を行なうため、周辺のサーバやスイッチが保持している ARP (Address Resolution Protocol) キャッシュに対して、仮想 IP アドレスと転送先のサーバの MAC アドレスの対応の更新を反映させる。

提案方式を実装したプロトタイプによる性能評価では、多くの仮想 IP アドレスを使用する本方式においてもサーバの性能低下は発生せず、また、リクエスト分散のためにセッションを移動させる時間は、1 ミリ秒前後と極めて小さいことが分かった。

以下の本稿の構成は、まず 2 節で従来のリクエスト分散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 上における評価について述べる。そして 5 節で関連研究を述べ、6 節でまとめる。

2 従来のリクエスト分散手法

Web サーバシステムをクラスタ構成で構築した場合、実際にリクエストを処理する Web サーバはデータセンタなどのプライベートネットワーク内に置かれ、その前段にディスパッチャもしくは負荷分散装置を置く。対外的には 1 つ (もしくはごく少数の) グローバルな IP アドレスを使い、クライアントはインターネットを経由してその IP アドレスに対してセッションを確立し、リクエストを送信し、リプライを受け取る。一方、Web サーバ

側では、クライアントとのセッションをディスパッチャが受け取り (このとき、TCP セッションを終端するかどうかはディスパッチャの機能及び実装による)、ディスパッチャがリクエストのパケットを分析して適切な Web サーバを選択し、その Web サーバにリクエスト処理をリダイレクトする。このディスパッチャとしては、L4 スイッチや L7 スイッチが広く使われている。

L4 スイッチはトランスポート層を含めた下位層の情報を判断基準に用いてリクエスト分散を行なう。例えば、TCP の SYN パケットが届いた時点で、パケットのポート番号から FTP と HTTP とを分散したり、送信先もしくは送信元の IP アドレスで分散したりする。処理内容が単純なだけにハードウェアでの処理も可能であり、高速処理はそれほど難しいことではない。しかし、極端に Web トラフィックに偏った最近のインターネットにおいては、トランスポート層の情報だけでは適切なリクエスト分散は難しい。

より複雑化してきている最近の Web サーバでは、リクエスト分散を柔軟に行なうためには、アプリケーション層の情報は不可欠である。例えば、アプリケーション層の情報を見ることで、プロトコル種別 (GET や POST など) で振り分けることも、URL のパス名で振り分けることも可能となる。また、Cookie も利用できる。このようなアプリケーション層の情報に基づく負荷分散装置が L7 スイッチである。L7 スイッチは非常に高価な装置であり、ネットワーク処理部分をハードウェアで実装し、HTTP ヘッダの分析や負荷分散アルゴリズムの実装などをソフトウェアで行なっている。

L7 スイッチは、クライアントとの TCP セッションを一旦終端し、リクエスト内容に応じて選択した Web サーバとの間にも別の TCP セッションを確立する。そして、Web サーバにリクエストを送信して受け取ったリプライをクライアントに転送する。このように、セッション毎、そして、リクエスト毎にアプリケーション層の情報を用いて適切な Web サーバ選択が行なえるという点では柔軟性は高いが、一方で、Web サーバから受け取ったリプライを一旦ユーザレベルで読み込んでからクライアントに転送するという処理も行なわなければならないため、処理負荷が高い。そこで、いくつかの研究においてこの処理負荷を軽減する方式が提案されている [10, 2, 4]。これらはいずれも TCP スプライシング機構 [3] を用いている。

TCP スプライシング機構は、2 つの TCP セッションをつなぎ合わせる技術である。トランスポート層でパケット転送を処理することによってユーザレベル (アプリケーション層) までデータを上げなくなり、高速な処理が可能になる。ただし、もとは独立している 2 つの TCP セッションをつないでいるので、パケットヘッダの書き換えが必要である。しかも、IP アドレスやポート番号の書き換えに加えて、シーケンス番号や、場合によっては、Ack 値や広告ウィンドウサイズ値も書き換える必要がある。このためディスパッチャはスプライシングのための変換テーブルを持ち、Web サーバとクライアントの IP アドレスやシーケンス番号の初期値などを保持している。

したがって、TCP スプライシングを用いた従来の手法には、以下の 2 つの課題が存在する。

1. パケット転送のための一定のオーバヘッド
2. パケット書き換えのための繁雑な情報管理

まず 1 に関しては、ディスパッチャの IP アドレス宛にパケットが送信されて来てしまうために、サーバにリダイレクトした後でもパケットの転送を行なわなければならないからである。例えば TCP-handoff [4] ではクライアントへのパケットはサーバからディスパッチャを通らずに直接送っているが、クライアントからの上りパケット (主に Ack パケット) がディスパッチャに送られてくるため、それをサーバに転送するためにスプライシングを行なっている。また、ディスパッチャ機能とパケット転送機能が同一の装置にあるため、仮りにディスパッチャに障害が発生したとき、既にリクエスト処理をサーバにリダイレクトしたセッションにも影響を与える可能性がある。

そして 2 に関しては、例えば、クライアントが Keep-Alive を用いて複数リクエストを要求した際に、ディスパッチャがリクエスト毎にサーバを切替

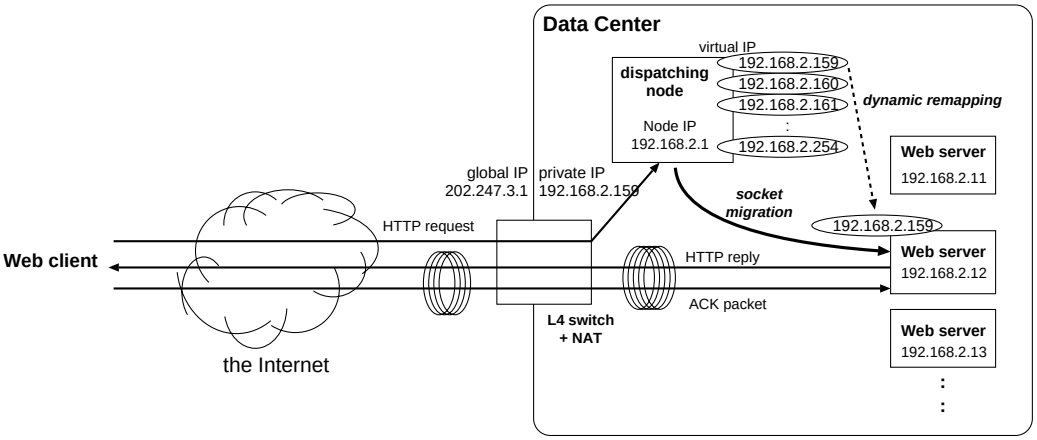

図 1: TCP マイグレーションを用いたリクエスト分散アーキテクチャ

えるとする。ディスパッチャにはスプライシングのための変換テーブルがあるが、サーバを切替える際にはこのテーブルの中のアドレスやシーケンス番号の初期値を書き換える必要がある。また、一般にこのテーブルは各装置の独自実装なので、装置外部から書き換えを指示するのは難しい。

3 提案モデルと実装

既に我々は、透過的なサービス移動機構 [9] を提案しているが、この方式では移動の単位がプロセスなので、リクエスト分散のために用いると、例えば 1 セッションに対して 1 プロセスを割り当てなければならないなど、転送コストが大きくなってしまう。

そこで、プライベートアドレスを豊富に使える環境とアドレス変換 (NAT) 機構を巧みに活用することによって、本稿では、TCP マイグレーション技術に基づくリクエスト分散アーキテクチャを提案する (図 1)。

提案方式のキーアイデアは、

- ディスパッチ機能とパケット転送機能の物理的な分離
- プライベート (仮想) IP アドレスのセッション毎の割り当て
- TCP セッションを OS 間でそのままの状態で転送

である。我々が提案する TCP マイグレーションを用いたリクエスト分散アーキテクチャでは、パケット転送はディスパッチャと分離されており、ディスパッチャとパケット転送のお互いの処理負荷や障害が他方には影響しなくなる。また、仮想 IP アドレスを TCP セッション毎に割り当てており、しかも、TCP セッションは OS レベルでそのままの状態で転送されるので、TCP セッションを転送してもアドレス変換テーブルを書き換える必要がない。ただし、仮想 IP アドレスと MAC アドレスの対応の更新を周辺のサーバやスイッチにブロードキャストする。

以下、本節では提案モデルと実装に関して述べる。

3.1 デザインアーキテクチャ

提案方式ではディスパッチャ部分とパケット転送部分を物理的に分ける。前者は計算機上のアプリケーションで行ない、後者は L4 スイッチなど装置に搭載されている NAT (RFC 1631, RFC 2663) で行なう。先述したように、下位層になればなるほど処理が簡単かつ高速になるので、NAT 処理はレイヤ 4 で高速な処理が可能である。ただし、提案手法では書き換えはレイヤ 3 だけで、FIN フラグなどの情報参照の目的でレイヤ 4 の一部を見ている。また、ディスパッチャ部分は、従来の Web サーバ (Apache や mini_httpd など) を改造してもよいし、リクエスト分散機能だけに絞った独自のアプリケーションを作成してもよい。これ自体はユーザレベルでの動作であるので、プログラム

の置き換えも容易であり、動作環境も PC やワークステーションなどで構わない。

更に、IP アドレスはディスパッチャや Web サーバのではなく、TCP セッション毎にユニークな仮想 IP アドレスを割り当てて使用する。ちなみに、セッション確立時の仮想 IP アドレス選択は L4 スイッチの NAT で行なうようにする。したがって、ディスパッチャも Web サーバも、NIC に固定の IP アドレスと、セッションに付随して追加される仮想 IP アドレスとの 2 種類の IP アドレスを同時に管理する。

我々がこのアーキテクチャを設計したのは次の理由による。セッション確立時に NAT が選択する IP アドレスとして固定的な IP アドレス (例えばディスパッチャの NIC のもの) を使うと、NAT テーブルを書き換えることでセッション毎に別のサーバに転送可能となる。しかしこれでは、NAT テーブルの繁雑な情報管理が必要となる。そこで、IP アドレスは 1 つの NIC (MAC アドレス) に対して複数割り当て可能なので、セッション毎に仮想的な IP アドレスを割り当てて、その仮想 IP アドレスを NIC に動的に追加・削除を行なうようにする。

図 2 は、実際の動作例を TCP/IP レベルで示したタイミングチャートである。まず、クライアントが SYN パケットを送信すると、NAT によって仮想 IP アドレスにアドレスが書き換えられる。仮想 IP アドレスとしては、そのときに NAT を通る通信に使っていない仮想 IP アドレスを選ぶ。つまり、同一の仮想 IP アドレスが複数のセッションで使われないようにする。通信中でない仮想 IP アドレスはディスパッチャが保持しているので、この SYN パケットはディスパッチャが受け取り、3way handshake が行なわれる。

次に、クライアントから HTTP リクエストが送信され、ディスパッチャがその内容を基に適切な Web サーバを選ぶ。ディスパッチャはその TCP セッションを Web サーバにそのままの状態で転送する。このとき、仮想 IP アドレスも一緒に動かす。そして、Web サーバはリクエストに応じたリプライを送信する。このとき、リプライのパケットも Ack のパケットも、Web サーバとクライアントの間で直接送受信され、ディスパッチャは経由しない。

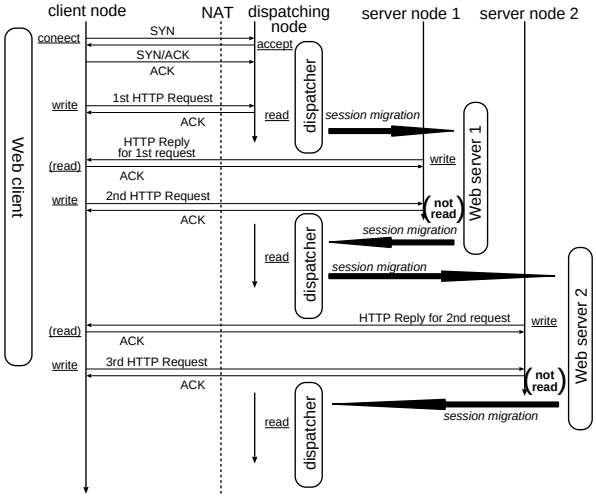

図 2: パケットの流れのタイミングチャート

1 つ目のリプライが終了したら、クライアントは Keep-Alive を利用して次のリクエストを送信することも可能である。このとき、Web サーバではリクエストを読み込まず、ソケットバッファを含めてそのままディスパッチャに TCP セッションを返す。ディスパッチャではソケットバッファのパケットに含まれている 2 つ目のリクエストを読み込み、再度、適切な Web サーバに TCP セッションを転送する。全リクエストに対するリプライが完了して TCP セッションを切断する際には、そのセッションが使っていた仮想 IP アドレスはディスパッチャに返される。また、TCP マイグレーションの間に到着したパケットは失われる可能性もあるが、TCP プロトコルの再送制御で再送される。

3.2 実装

TCP マイグレーションを Linux カーネル (バージョン 2.4) に実装した。API として、GetSocket と SetSocket の 2 つのシステムコールを提供する。この API は、TCP セッション及びそのセッションに関連する仮想 IP アドレスの情報を取得したり設定したりするためのものである。したがって、アプリケーション (ディスパッチャや Web サーバ) が明示的に API を用いてセッション (つまり、ソ

表 1: 測定環境

	サーバ、ディスパッチャ	クライアント
CPU	Xeon 3.06 GHz (HyperTheading Off)	Pentium4 2 GHz
メモリ	1 GB (DDR SDRAM)	512 MB (PC133)
バス	PCI-X (64bit/100MHz)	PCI (32bit/33MHz)
NIC	Intel PRO/1000 XT	Intel EtherExpressPro 100
OS	Linux 2.4.20-28.7 (RedHat 7.3)	Linux 2.6.5.1-358 (FedoraCore 2)

ケット) のデータを読み出して相手先に転送し、相手先のアプリケーションも明示的に API を用いてセッションを設定する。OS が設定後、アプリケーションにはファイルディスクリプタが返され、これを使ってクライアントとの維持された通信が再開できる。

実装した機能を API 別に説明すると、GetSocket では

- TCP セッション (ソケット) のデータ構造を取得した後、ソケットを消去する
- 仮想 IP アドレスを消去する

ということを行なう。一方、SetSocket では

- ソケットを作成し、転送されたソケット情報を設定する
- 作成したソケットを、確立済みのセッションとしてハッシュテーブルに登録する
- ソケットをアプリケーションのプロセスのファイルディスクリプタに割り当てる
- 仮想 IP アドレスを設定する

ということを行なう。なお、SetSocket の最後では、仮想 IP アドレスと MAC アドレスの対応の更新を周辺のサーバやスイッチに周知するために、サーバやスイッチが保持している ARP キャッシュの無効化や更新 (Gratuitous ARP [7] など) をブロードキャストする。

実装コード量は C 言語で 1,255 行である。これはカーネルモジュールとして実現されており、アプリケーションに対して先述した 2 つの API を提供している。

ちなみに、Linux では仮想 IP アドレスは 255 個までしか設定できないが、FreeBSD などではより多くが設定できる。

また、改造はサーバ側の OS 及び (必要な) アプリケーションだけであり、クライアント側の OS 及びクライアントソフトウェアの改造は一切必要ない。つまり、提案手法は、クライアント側からは透過的にサーバシステムに適用出来るということになる。

4 性能評価

TCP マイグレーションの性能を評価するために、以下の 3 つの測定を行なった。

1. 実アプリケーションを用いた応答時間
2. セッション転送に必要な時間
3. 仮想 IP アドレスを用いた場合の処理負荷

以下では、それぞれの測定に関して述べる。なお、測定環境は表 1 の通りである。

測定では Web サーバとディスパッチャとして mini_httpd (バージョン 1.15c) を改造して用いた。改造量は約 250 行であり、この大半はセッション転送のため操作である。Web クライアントとしては wget (バージョン 1.9) を無改造で用いた。オーバヘッドの評価のための比較対象は、無改造の mini_httpd である。

実験手順は次の通りである。wget がセッションをディスパッチャと確立し、リクエスト (141 バイト) を送信し、ディスパッチャがリクエストを読み込んで Web サーバにセッションを転送し、Web サーバが HTTP リプライ (ヘッダ部 223 バイトとボディ部) を送信し、セッションを切断する。今

表 2: mini_httpd を用いた応答時間

Reply Size	5KB	10KB	20KB	50KB	100KB	200KB	500KB
(A) TCP-migration [ms]	4.1	4.2	5.2	8.1	12.6	21.2	47.6
(B) Normal [ms]	3.3	3.7	4.5	7.4	11.4	20.2	46.9
(A) – (B) [ms]	0.8	0.5	0.7	0.7	1.2	1.0	0.7

表 3: 個別 API のオーバヘッドテスト

application	GetSocket	SetSocket
mini_httpd	49 μs	104 μs

回の測定ではリクエスト数は 1 としたため、Web サーバからディスパッチャへのセッションの戻しは行なわれない。

HTTP リプライのファイルサイズを変更することでボディ部の転送量を 5 KB から 500 KB まで変更して、それぞれの応答時間を測定した結果を表 2 に示す。(A) は TCP マイグレーションを用いた提案方式による応答時間であり、(B) は無改造の mini_httpd による応答時間である。両者の差が提案方式によるオーバヘッドであり、多少のバラツキはあるもののほぼ 1ms 以下であった。

このオーバヘッドを個別にみていくと、ディスパッチャから Web サーバへセッションを転送するのにかかる時間は、GetSocket (セッション情報の取得と消去)、セッション情報のネットワーク転送時間、SetSocket (セッション情報の設定) の 3 つの時間の合計となる。表 3 は、各 API の所要時間を `gettimeofday(2)` を用いて測定した結果である。GetSocket は 49 μs かかり、SetSocket は 104 μs かかった。今回の測定では、セッション情報は URL (1024 バイト) と `sock` 構造体及び `socket` 構造体で構成されているので、合計で 2,308 バイトのデータとなった。これを Gbit ネットワークで転送したため、ネットワーク転送時間は 500μs 以下であった。

提案方式では仮想 IP アドレスを多く扱う必要があるため、仮想 IP アドレスを追加した際に他のソケットに悪影響を与えるか否か、また、仮想 IP アドレスを使うソケットのパケット処理が通常のパケット処理と比べてどうかを測定した。測定

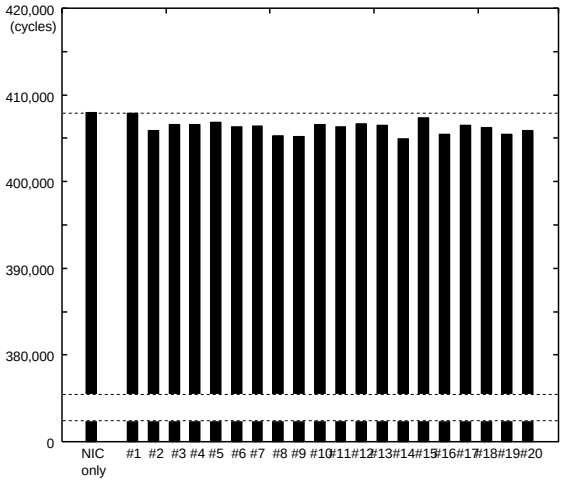

図 3: 仮想 IP アドレスのオーバヘッド

方法は、20 個の仮想 IP アドレスを追加し、それぞれの IP アドレスに対して 20,000 パケットを送信して、パケット毎の平均処理レイテンシを測定した。測定箇所は、bottom half の最初 (つまり netif_rx() 関数) とユーザレベルの read システムコール発行直後である。結果を図 3 に示す。縦軸は CPU サイクルである。仮想 IP アドレスを追加していない場合の 407,996 サイクルであったのと比較すると、追加された仮想 IP アドレスの処理レイテンシは平均して –0.43% のオーバヘッドと、ほぼ無視できる程度であった。

以上より、提案方式は HTTP リプライのサイズに依存せずに 1ms 程度のオーバヘッドであり、また、追加した仮想 IP アドレスに対するパケットの処理負荷も通常の負荷と同等なので、実用的な性能であることが分かった。

5 関連研究

TCP handoff [4]、AnyPoint [10]、非対称スプライシング方式 [2] は、TCP スプライシングを利用した HTTP リクエスト分散機構であるが、Web サーバを切替える際にはアドレス変換テーブルを更新する必要がある。

VNAT [8] や Racks [11, 12] でも、変換テーブルを OS 内に持ち、通信相手が移動した際には専用のプロトコルを用いて IP アドレスの変更を互いの OS に通知する。同様に、TCP Migrate option [5, 6] では TCP プロトコルに “Suspend” と “Resume” を追加し、ソケットマイグレーション時にはこれらの制御プロトコルを利用して通信の一時停止や IP アドレスの変更の通知を行なう。

我々の提案方式は、既存のプロトコルには修正を加えていないため、クライアント側の OS やアプリケーションに変更を必要としない。

6 まとめと今後の課題

本稿では、クラスタ構成の Web サーバにおける柔軟かつ拡張性の高いアーキテクチャを提案した。提案方式は、ディスパッチャ部分とパケット転送部分を物理的に分離しており、ディスパッチャがクライアントとの TCP セッションを確立し、HTTP リクエストの記述に基づいてサーバを決定し、サーバに TCP セッションをリダイレクトする。TCP セッションは完全にサーバに転送されるので、リダイレクト後の通信パケットはディスパッチャを通らず、したがって負荷もかけない。

今後は、ゼロウィンドウ広告 [1] などの方式を適用することによって、セッション転送時におけるパケットロスを発生させないようにする予定である。更に、異なる OS 間のセッション転送に適用する予定である。このために、セッション情報を Linux に依存しないデータ構造にし、転送されたセッションを再構築する実装をする予定である。

謝辞 本研究の一部は、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] T. Goff, J. Moronski, D. S. Phatak, and V. Gupta. Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments. In *Proceedings of the IEEE INFOCOM '00*, 2000.

[2] M. Kobayashi and T. Murase. Asymmetric TCP Splicing for Content-Based Switches. In *Proceedings of the IEEE International Conference on Communications*, 2002.

[3] D. A. Maltz and P. Bhagwat. TCP Splicing for Application Layer Proxy Performance. Technical Report IBM technical report RC 21139, IBM Research Division, March 1998.

[4] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1998.

[5] A. C. Snoreren, D. G. Andersen, and H. Balakrishnen. Fine-Grained Failover Using Connection Migration. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, 2001.

[6] A. C. Snoreren and H. Balakrishnen. An End-to-End Approach to Host Mobility. In *Proceedings of the 6th International Conference on Mobile Computing and Networking*, August 2000.

[7] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[8] G. Su and J. Nieh. Mobile Communication with Virtual Network Address Translation. Technical Report Technical Report CUCS-003-02, Columbia University, February 2002.

[9] 高橋、菅原. 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. **情処研報** *2004–OS–96*, June 2004.

[10] K. G. Yocum, D. C. Anderson, J. S. Chase, and A. M. Vahdat. Anypoint: Extensible Transport Switching on the Edge. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.

[11] V. C. Zandy and B. P. Miller. Reliable Network Connections. In *Proceedings of the 8th International Conference on Mobile Computing and Networking*, September 2002.

[12] V. C. Zandy and B. P. Miller. Checkpoints of GUI-based Applications. In *Proceedings of the USENIX Annual Technical Conference*, June 2003.