Vol.2009-ARC-183 No.1
Vol.2009-OS-111 No.1
2009/4/22

レイヤー 7 負荷分散のための TCP 接続移送機構

藤田 肇[†1] 石川 裕[†2,†1]

本研究報告では，ブロードキャスト型の単一 IP アドレスクラスタにおいてレイヤー 7 負荷分散を行うための TCP 接続移送機構，DTS(Distributed TCP Splicing) を提案し，Linux 上での実装方針について述べる．DTS はパケットフィルタを用いて TCP ヘッダのシーケンス番号を調整することによって，TCP プロトコルスタックを改変せずに TCP 接続を他のサーバーノードに移送することができる．接続の移送後は，移送元のノードは移送先を介さずにクライアントと通信を行う．この機構はクライアント，サーバーアプリケーション，サーバーの TCP プロトコルスタックにとって透過的である．

TCP Connection Handover Mechanism for Layer-7 Load Balancing

Hajime Fujita[†1] and Yutaka Ishikawa[†2,†1]

In this paper a new TCP connection handover mechanism named DTS(Distributed TCP Splicing) is proposed and its implementation on Linux kernel is described. DTS aims at realizing layer-7 load balancing on broadcast-based single IP address clusters. DTS migrates existing TCP connections to another server node without any modification to TCP protocol stack. This is achieved by adjusting sequence numbers on TCP headers at a packet filter. After a connection handover is performed, the destination of the handover communicates with the client without involving any other node. DTS is transparent to clients, server applications, and TCP protocol stack.

†1 東京大学情報基盤センター
Information Technology Center, The University of Tokyo
†2 東京大学情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

1. はじめに

インターネット上でサービスを提供するサーバーの性能と可用性を向上させるために，複数のサーバーノードでクラスタを構成するという手法が用いられる．この際，サーバークラスタに 1 つの IP アドレスを割り振って単一 IP アドレスクラスタとすることで，クライアント側に変更を要求せずにサーバー側のノードを増減させることができるようになる．

サーバークラスタの構成及び負荷分散手法は主に Web サーバーの分野で広く研究されており，既に様々な手法が存在する[1)]．既存の手法を特に単一 IP アドレスクラスタに絞って考えると，クラスタの構成法と負荷分散手法により分類することができる．

クラスタの構成法に着目すると，既存の手法は代表ノード型とブロードキャスト型に分類できる．代表ノード型クラスタはクラスタを代表するノードが 1 つ存在し，このノードが外部に見せる IP アドレスを持ちクライアントとの通信を担当する．特に，クライアントからサーバーに送られる IP パケットは必ずこの代表ノードによって一旦受信され，背後のサーバーノード (バックエンドノード) の 1 つに転送される．一方ブロードキャスト型は代表ノードを持たず，全てのサーバーノードが同じ IP アドレスを持ち，この IP アドレスをクライアントに見せる．また，スイッチ等を設定することでクラスタ宛のパケットがクラスタの全ノードにブロードキャスト (またはマルチキャスト) されるようにしておく．その上で，クラスタ宛のパケットをクラスタの 1 ノードのみが受け取って返答することで，クライアントから見ると 1 台のサーバーが動いているように見える．

一方負荷分散の手法に着目すると，レイヤー 4 負荷分散とレイヤー 7 負荷分散の 2 種類に分けることができる．これは，クライアントからのリクエストに含まれる情報をどの程度利用して負荷分散の指標にするかという観点で分類したものである．レイヤー 4 負荷分散は，OSI 参照モデルのうち第 4 層，すなわちトランスポート層ヘッダに含まれる情報までを用いて負荷分散を行う．TCP を用いるアプリケーションの場合は，クライアント・サーバー双方の IP アドレス及びポート番号が考慮の対象となる．一方のレイヤー 7 負荷分散は OSI 参照モデルの第 7 層，アプリケーション層に含まれる情報も用いて負荷分散を行う．例えばサーバーアプリケーションが Web サーバーであれば，レイヤー 7 負荷分散ではクライアントから送られてくる HTTP リクエストを読み取ってその内容に応じて負荷分散を行う．レイヤー 7 負荷分散の代表的な応用例として，同じファイルに対するリクエストを同じノードに担当させることで局所性を高める手法がある (Locality Aware Request Distribution)[2),3)]．

著者らは既にブロードキャスト型の単一 IP アドレスクラスタにおいて負荷分散を行うた

Vol.2009-ARC-183 No.1
Vol.2009-OS-111 No.1
2009/4/22

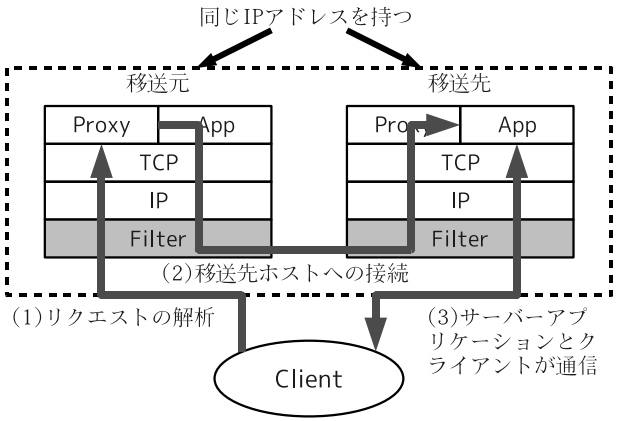

図 1 DTS の概要

めの機構 FTCS を提案している[4] が，これはレイヤー 4 負荷分散のみを考慮している．著者らは現在，FTCS をレイヤー 7 負荷分散に拡張することを考えているが，本研究報告ではそのための第一歩として，ブロードキャスト型の単一 IP アドレスクラスタにおいて TCP 接続を他のノードに移送する機構 DTS(Distributed TCP Splicing) を提案する．

DTS は TCP splicing[5] と同じように，TCP ヘッダのシーケンス番号を書き換えることで，本来別々の TCP 接続に属する端点同士を通信可能にする．この書き換えは TCP/IP プロトコルスタックよりもより下の層で行われるため，クライアントやサーバーアプリケーションはもちろん，サーバーの TCP プロトコルスタックに対してさえも透過である．

2. 設　　計

本節では，DTS の設計について述べる．

DTS の大まかな構成を**図 1** に示す．本機構は，オペレーティングシステムの TCP/IP プロトコルスタックの外側で動作することを想定しており，パケットフィルタや仮想ネットワークデバイスドライバ，最終的には物理的に異なるネットワーク機器においても同様の機構を実現することを目標にしている．

DTS の動作の概要を図 1 を使用して説明する．

(1) クライアントからサーバ側に新しい TCP 接続要求が送られる．この接続要求はサーバーノード群中の 1 つのノードによって受け付けられ，このノードとクライアントとの間に TCP 接続が確立される．このときクライアントから送られてきたリクエスト内容は，クライアントからの接続要求を待っているサーバーアプリケーションではなく，リクエスト内容を解析するためのアプリケーション (プロキシ) に渡される．プロキシはリクエスト内容を解析してそのリクエストを処理すべきサーバーノードを決定し，OS カーネルに対して TCP 接続の移送を要求する．

(2) 移送元のカーネルは，接続の移送先に対して新しい TCP 接続を確立し，プロキシが読み取った分のリクエストデータを送信する．この際移送先のパケットフィルタでは，この新しい接続があたかもクライアントから直接来たかのように IP ヘッダ及び TCP ヘッダを書き換えてサーバーアプリケーションに渡す．

(3) 移送元が全てのリクエストデータを送信しきったら，その後移送元ではクライアントから送信されてきたパケットを直接受信する．これ以降は，移送元は一切通信に関与せず，クライアントと移送先のノードの間のみで通信が行われる．

2.1 前提とする環境

本機構はブロードキャスト型の単一 IP アドレスクラスタを前提としている．従って，クラスタ内の各サーバーノードは同一の IP アドレスを持つ．また，クラスタに向かうパケットはクラスタの全ノードに向けてブロードキャストされる．よってクラスタ内の全ノードは同時に同じ IP パケットを受けとることになる．一方で，TCP 接続の移送を行う際にはサーバーノード間での通信を行う必要があるため，各ノードはノード間通信用にもう一つのネットワークインターフェースと IP アドレスを持つ必要がある．

また本機構は対象とするアプリケーションレベルのプロトコル，すなわちレイヤ 7 プロトコルにも仮定をおいている．本機構が対象としているアプリケーションプロトコルは次の条件を満たす必要がある．まず，TCP 接続はクライアントから開始されることを想定している．これはクライアント-サーバー型のプロトコルでは自然であると考えられる．次に，TCP 接続の確立後はクライアント側が何らかのリクエストをサーバーに対して送信することが必要である．この点で，SMTP[6] のようなプロトコルは現在の設計では対象外である．

2.2 リクエストの解析

レイヤー 7 負荷分散を行うためには，クライアントとの間に一度 TCP 接続を確立し，クライアントから TCP で送信されたリクエストの内容を解析してそのリクエストがどのサーバーノードで処理されるべきか判断する必要がある．

DTS では，最初にクライアントからの TCP 接続を受け付けるサーバーノードにおいてリクエストの解析を行うための専用のプログラムを用いる．このプログラムをプロキシと呼

Vol.2009-ARC-183 No.1
Vol.2009-OS-111 No.1
2009/4/22

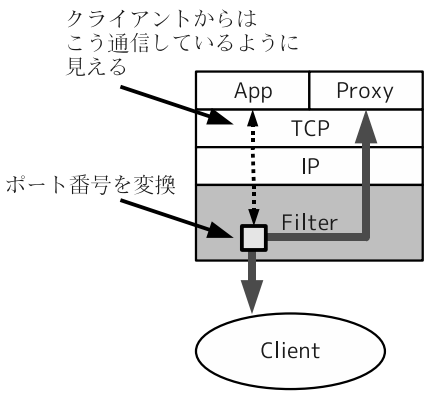

図 2 クライアントとプロキシを通信させるためのポート番号の変換

ぶ．プロキシはサービスを提供するサーバーアプリケーションとは異なる TCP ポートを 1 つ確保し，クライアントからの接続を待ち受ける．

クライアントからサーバーアプリケーション宛に TCP セグメントが来ると，パケットフィルタが宛先ポート番号をアプリケーションのものからプロキシが待機しているポート番号へと変更する (**図 2**)．逆にプロキシのポート番号を発信元として送信される TCP セグメントについては発信元をサーバーアプリケーションのポート番号に書き換えてから送信する．こうすることによって，クライアントからはサーバーアプリケーションと通信しているように見えるが，実際にはプロキシと通信している状態になる．

プロキシのプログラムは通常のソケットに対してするのと同じように `read` や `recv` システムコールを発行してクライアントからのリクエストを読み取り，リクエスト内容を解析する．プロキシのプログラムに対する制約として，ソケットに対する `write` を行ってはならないということがある．

2.3 TCP 接続の移送

プロキシはリクエストの解析を終えてそのリクエストをどのノードで処理すべきかを判断すると，**図 3** に示す `request_handover` システムコールを発行して接続の移送をカーネルに要求する．このシステムコールは，`socket` で指定されたソケットに関連した TCP 接続を `dest` で指定されたホストに移送する．この際，既に読み取ったリクエストを `request_buf` に引数として渡す．

```
int request_handover(int socket,
    const struct sockaddr *dest, socklen_t addrlen,
    const char *request_buf, size_t buflen)
```

図 3 TCP 接続移送を要求するシステムコール

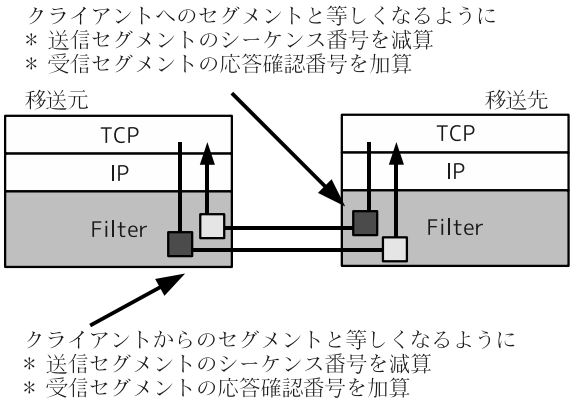

図 4 シーケンス番号及び確認応答番号の調整

カーネル内のシステムコールハンドラでは，移送先に対して新しい TCP 接続を確立し，`request_buf` に渡されたデータ及びプロキシのソケットバッファ中にある未 `read` の受信データを転送する．

2.4 シーケンス番号の調整

TCP 接続が移送される際には新しい TCP 接続が移送元と移送先の間に生成される．最終的にはこの接続を用いて移送先のノードはクライアントと通信することになるが，この接続が使用している TCP のシーケンス番号は，一般にクライアントとプロキシ間の接続で使われているシーケンス番号とは同期していない．このため，単純にパケットの IP アドレスおよびポート番号を書き換えるだけではこの新しい接続をクライアントとの通信に用いることはできない．そこで，TCP/IP プロトコルスタックの下で入出力パケットを監視し，TCP ヘッダ中のシーケンス番号及び応答確認番号を適切に書き換えることで，TCP 接続移送先の TCP スタックとクライアントの TCP スタックが通信できるようにする (**図 4**)．

具体的にはまず移送元から移送先ホストに対して新しい TCP 接続を開始する際の SYN

Vol.2009-ARC-183 No.1
Vol.2009-OS-111 No.1
2009/4/22

図 5 TCP 接続移送時に送られる TCP オプション

表 1 TCP オプションのフィールド

フィールド	長さ	説明
オプション種別	8 ビット	現在のところ値 32(10 進数) を使用している.
Len	8 ビット	このオプションの長さ (オクテット単位). オプション種別及び Len フィールド (合計 2 オクテット) も含む.
DTSraddr	32 ビット	クライアントの IP アドレス.
DTSrport	16 ビット	クライアントのポート番号.
DTSinitseq	32 ビット	接続移送元のプロキシがクライアントに送った SYN+ACK セグメントのシーケンス番号.

セグメントのシーケンス番号を，クライアントが最初にサーバー側に送信した SYN セグメントのシーケンス番号と等しくなるように一定量の減算を行う．移送先ホストから応答のセグメントが返ってくる際には，逆に確認応答番号に同じだけ加算して移送元 TCP プロトコルスタックが期待する確認応答番号とする．

また同様に，移送先ホストは移送元からの TCP 接続要求 (SYN セグメント) に応答 (SYN+ACK セグメント) を返すが，この際のシーケンス番号を最初にサーバーノードとクライアントが接続した際にサーバーノードが SYN+ACK セグメントで返したシーケンス番号と等しくなるように調整する．

2.5 TCP 接続移送に必要な情報の伝達

TCP 接続の移送を実現するためには，移送元から移送先に対して本来クライアントとの間に存在している接続についての情報を伝える必要がある．例えば，クライアントの IP アドレス，ポート番号，クライアントに対してサーバー側から最初に返したシーケンス番号などである．これらの情報は，TCP 接続の移送元から移送先に新しく接続を試みる際に TCP オプションとして渡される．

TCP オプションの構造は**図 5** に示す通りである．それぞれのフィールドは**表 1** に示す意味を持つ．複数バイトに渡るデータは全てネットワークバイトオーダーで格納される．

2.6 リクエスト内容の送信と接続移送の完了

移送元と移送先との間に新しい TCP 接続が確立されると，移送元はこの接続を利用して既にプロキシが読み取ったリクエスト内容を移送先に送信する．この接続を通して移送先に到着した IP パケットは，パケットフィルタによって送信元アドレスとポート番号を変換され，移送先の TCP プロトコルスタックにとってはクライアントから送られてきたパケットのように見える．最初にクライアントがサーバーに接続した時とは異なり，接続の移送を行う際には移送元ではプロキシではなく，サーバーアプリケーションに直接データを配送する．

なおこの際，移送元でリクエストを受け取ったサーバーアプリケーションが移送の完了前からデータを送信することがありえる．移送完了前にデータが送られると，そのデータはクライアントではなく，移送元へと向かってしまう．これを防ぐため，移送元が送信する TCP セグメントはパケットフィルタにおいてウィンドウサイズを 0 に書き換える．これによって移送先の TCP プロトコルスタックはたとえ送信すべきデータがあったとしても送信を行わなくなる．

移送元は必要なデータの送信を終えると，移送先との接続を閉じる．この際移送元から移送先に FIN ビットの立った SYN セグメントが送られる．移送元は FIN ビットを検出すると，移送元から必要なデータを全て受け取ったと判断し，クライアントから直接データを受信するようになる．この時点で，移送先の TCP プロトコルスタックがクライアントから次に受け取るセグメントのシーケンス番号 (`RCV.NXT`) はクライアントが次に送信してくるセグメントのシーケンス番号と等しくなっている．

2.7 TCP オプションに関する議論

現在の設計及び実装には含まれていないが，提案機構の実現にあたって考慮すべき TCP オプションがいくつか存在する．

まず，Selective ACK(SACK) オプション[7] が有効になっている場合，TCP ヘッダ中の確認応答番号フィールドだけでなく，TCP オプションとしても確認応答番号が含まれている．このため SACK オプション中の確認応答番号もヘッダと同様に調整する必要がある．

また，タイムスタンプオプション[8] が有効になっている場合，このオプション中には単調増加するタイムスタンプ値が含まれることが期待される．タイムスタンプの増え方とある瞬間における値は当然計算機ごとに異なるため，タイムスタンプオプションが使われている場合にはクライアントが混乱しないようにタイムスタンプの増減が必要である．

同様に最大セグメント長 (MSS) オプション[9] やウィンドウスケールオプション[8] についても，最初にクライアントとプロキシとの間に開いた接続と，接続移送元から移送先に開い

Vol.2009-ARC-183 No.1
Vol.2009-OS-111 No.1
2009/4/22

た接続とで矛盾することがないようにする必要がある．

2.8 OS カーネル依存性に関する議論

提案機構のうち，`request_handover` システムコールの実装は明らかに OS カーネルの改変が必要である．このシステムコールは TCP 接続の状態 (次に送信するシーケンス番号など) を取得したり，ソケットの受信キューを操作する必要があり，TCP プロトコルスタックの実装について多少の知識を必要とする．一方で TCP 接続の移送を受ける側では，操作するのは TCP セグメントだけであり，プロトコルスタックの実装に関する知識は必要ない．また，移送元から移送先に送られる情報は 2.5 節で述べたように，IP アドレス，ポート番号，シーケンス番号といった情報のみである．従って，提案手法は移送元と移送先が同じ OS・同じバージョンでなくても動作するものと考えられる．

3. 実　　装

現在我々は提案機構のプロトタイプを Linux カーネル 2.6.28 上に実装中である．本節では，プロトタイプの実装方針について述べる．このプロトタイプはモジュールとして実装されており，Linux カーネル本体に静的なパッチは必要ない．

3.1 パケットフィルタの実装

DTS で最も重要な役割を果たすのは入出力される TCP セグメントを書き換える部分である．Linux には Netfilter[10] というパケットフィルタリングのための機構があり，プロトタイプ実装ではこれを利用する．

パケットフィルタをカーネルに登録すると，システムに出入りする全てのパケットがパケットフィルタを通過するようになる．このため，各パケットについて書き換えが必要かどうか判断しなければならない．DTS では，論理的に 1 つの TCP 接続であっても 2 つより多い端点 (IP アドレスとポート番号の組) を持ち得る．例えば，接続の移送先においては，クライアントと自ノードとの間の本来の接続の他に，移送元と自ノードの間の移送用の接続もあり，これら 2 つの TCP 接続は本来同じ TCP 接続を表している．このため，**図 6** に示すようなデータ構造を用いる．

パケットフィルタにパケットが入ってくると，ローカル IP アドレス，ローカルポート番号，リモート IP アドレス，リモートポート番号の 4 つ組をキーとしてハッシュ表の探索を行う．一致するエントリが見つかったら，そこにはそれぞれの端点の種類 (type) が記されている (**表 2**)．

次に，このエントリから Filter Control Block (FCB) と呼ぶ構造体を辿る．FCB は論理

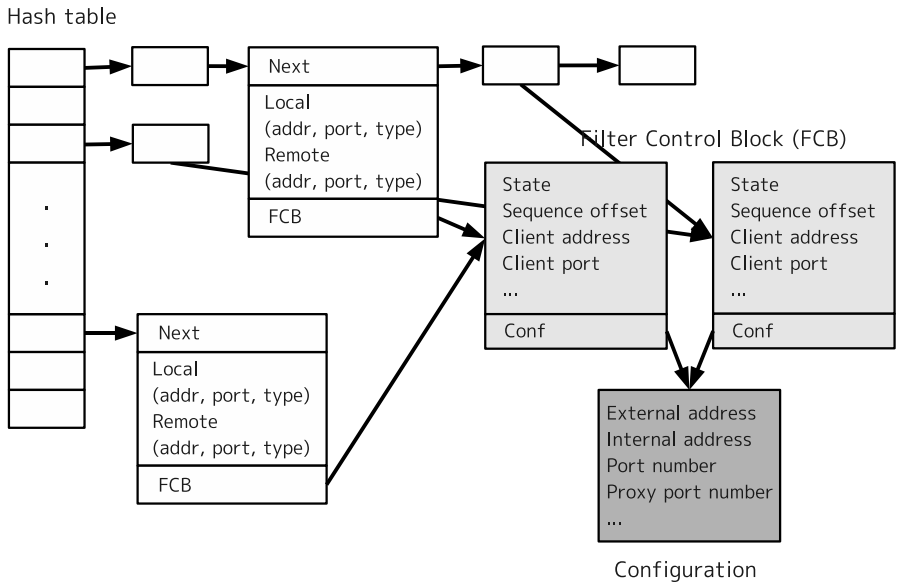

図 6　パケットフィルタリングのためのデータ構造

表 2　端点の種類

端点の種類	ローカル/リモート
外部向けアドレス・アプリケーションポート	ローカル
外部向けアドレス・プロキシポート	ローカル
内部向けアドレス・アプリケーションポート	ローカル
クライアントアドレス・不特定のポート	リモート
接続移送先の内部向けアドレス・アプリケーションポート	リモート
接続移送元の内部向けアドレス・不特定のポート	リモート

的な TCP 接続 1 つにつき 1 つ作成される構造体で，接続の状態やクライアントのアドレスとポート番号，加減算すべきシーケンス番号の量などが保存されている．さらに FCB からは，Configuration と呼ぶ構造体を辿ることができる．これは 1 つのインターネットサービスにつき 1 つ生成される構造体で，あるサービスが使うアドレス，ポート番号，プロキシのポート番号などを保存する．Configuration 構造体はユーザーの指定によって作成される．一方 FCB は，ノードが新しい TCP 接続を受け付ける際や新しい接続を作成する際に作成されハッシュ表に登録される．

3.2 パケットの取捨判断

ブロードキャスト型の単一 IP アドレスクラスタにおいては，クラスタ宛の全てのパケットが全ノードに届く．これは，本来自ノードに関係のないパケットであっても入ってくるということを意味し，各ノードはそれぞれのパケットについて自分が処理すべきかどうか自律的に判断する必要がある．

3.2.1 SYN セグメント

クライアントが新しい TCP 接続を開こうとするとき，SYN ビットの立ったセグメントを送ってくる．これに応答するノードはクラスタ内でただ 1 つでなければならない．現在のプロトタイプ実装では，あらかじめ TCP 接続を受付リクエストを解析するノードを指定しているが，例えば既存のブロードキャスト型単一 IP アドレスクラスタのようにハッシュ値を用いる[11)–13)] ことも可能である．

3.2.2 SYN 以外のセグメント

SYN 以外のセグメントについては，各ノードが独立に判断を行う．具体的には，Configuration 構造体が存在して FCB が存在しない場合，それは他のノード宛であると判断してパケットを捨てる．

3.3 システムコールの実装

プロキシからの TCP 接続移送要求を受け付けるために，`request_handover` システムコールを追加する．ただし，Linux で本来の意味でいうシステムコールを実装するためには，カーネル内に静的に定義されたシステムコールテーブルを変更せねばならず，モジュールでの実装が難しいという点で実装の敷居が高い．ここでは，ユーザー空間からカーネル空間に何らかの手段でデータを渡せれば十分である．そこで，Netfilter の機能の一つである，`setsockopt` システムコールのハンドラを定義する機能を使って実装を行った．

システムコールハンドラはプロセスコンテキストで動作するため，`socket`，`connect`，`sendmsg` といったソケット API のカーネル内実装を利用することができる．これらを用いて，移送先のホストに接続を行い，リクエストデータを送信する．

ここで，プロキシが `read` あるいは `recv` システムコールで読み取ったデータ以上のデータが既に到着し，カーネル内の受信キューに入っている可能性を考慮する必要がある．Linux の TCP 実装では，受信キューに入っているセグメントについては既に応答確認が返されている可能性があるため，これらのデータも併せて送信する．

4. 関 連 研 究

本節では，単一 IP アドレスクラスタにおいてレイヤー 7 負荷分散を行うための方式に関する既存の研究について述べる．

4.1 代表型単一 IP アドレスクラスタを対象としたもの

TCP splicing[5)] は本研究と同様にシーケンス番号の書き換えによってプロトコルスタックを変更することなしに TCP 接続の移送を行う．TCP splicing では，代表ノード上のプロキシがクライアントとの後に接続を開き，リクエストを受信し解析した後，リクエストを受け付けるべきバックエンドサーバーに対して新しく TCP 接続を確立する．そしてこの接続上にリクエスト内容を送信したのち，OS カーネルに対して対クライアント接続と対バックエンド接続を結合 (splice) するよう依頼する．これ以降，OS カーネルはクライアントからセグメントを受信すると，アドレスとポート番号，シーケンス番号を書き換えてそのままバックエンドに送信する．逆も同様である．この仕組みによって，ユーザー空間のプロキシが全てのデータを中継する場合に比べて，TCP スタックとコンテキストスイッチのオーバーヘッドを削減できる．TCP splicing においては，ヘッダ書き変えの整合性を取るため，バックエンドサーバーからクライアントへ向かうパケットも必ず再度代表ノードを経由する必要がある．TCP splicing は代表ノード型のため，代表ノードが故障すると全サーバーノードがクライアントと通信できなくなる．一方本研究の提案機構は，このような代表ノードを持たず，TCP ヘッダの書き換えは各サーバーノードが個別に行うため，他のサーバーノードの故障に影響されずに通信が継続可能である．

KNITS(Knowledgeable Node Initiated TCP Splicing)[14)] は TCP splicing を応用した手法であり，リクエストの解析を代表ノードではなくそれぞれのバックエンドノードで行う．その点で KNITS は本研究の提案手法に近い．リクエストを処理すべきノードが明らかになると，バックエンドノードは代表ノードにメッセージを送り，それ以降にクライアントから到着したパケットが接続移送先のノードに送られるようにする．

TCP handoff[15)] は TCP splicing と異なり，バックエンドノードの TCP プロトコルス

Vol.2009-ARC-183 No.1
Vol.2009-OS-111 No.1
2009/4/22

タックに手を加え，TCP 接続の移送時に TCP ソケットの状態を書き換える．これによって，シーケンス番号等の変換なしにクライアントとバックエンドノードが通信できるようになる．このためバックエンドサーバーからクライアントに向かうパケットは代表ノードに中継される必要がない．TCP handoff はバックエンドサーバーの TCP プロトコルスタックを書き換える必要があるため，実装のコストが高い．また OS によっては TCP プロトコルスタックを書き換えるためのインタフェースが公開されていないこともあり，この手法を適用不能な場合が存在する．

4.2 ブロードキャスト型単一 IP アドレスクラスタを対象としたもの

Kerdlapanan らは本研究と同様にマルチキャストベースの単一 IP アドレスクラスタにおいて TCP handoff を用いてリクエスト振り分けを行う手法を提案している[13]．この手法では，サーバークラスタに対する新しい TCP 接続要求がくると，クライアントの IP アドレスによるハッシュ値を取って接続を受け入れるサーバーノードを決定し，リクエストの解析を行う．しかし，この研究では TCP 接続の移送の実現方式については TCP handoff の既存実装を用いると述べているにとどまり，踏み込んだ議論は行われていない．

5. まとめと今後の課題

本研究報告では，TCP 接続を移送する機構 DTS(Distributed TCP Splicing) の提案を行った．この機構は，パケットフィルタによってパケットの送信元アドレス，ポート番号，シーケンス番号を書き換えることによって，他のサーバーノードから移送されてきた TCP 接続があたかも外部のクライアントから来たものであるかのように TCP プロトコルスタックに見せることによって TCP 接続の移送を実現する．DTS はクライアント，サーバーアプリケーション，TCP プロトコルスタックの全てにとって透過的であり，これらのどれも変更する必要がない．

今後の課題は，一度移送された TCP 接続の後続リクエストを再度移送先のプロキシで解析できるようにすることである．例えば HTTP/1.1[16] では，1 つの TCP 接続に複数のリクエストが含まれ得る．このため，TCP 接続の途中から別のサーバーノードが処理する必要が生じる可能性がある[3),17]．さらなる課題としては，提案機構を実際のサーバーアプリケーションに適用し，効果を確認する必要がある．

謝辞 本研究の一部は，科学技術振興機構 戦略的創造研究推進事業 (CREST) (領域名：実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム) 技術課題：「高信頼組込みシングルシステムイメージ OS」による．

参 考 文 献

1) Cardellini, V., Casalicchio, E., Colajanni, M. and Yu, P.S.: The State of the Art in Locally Distributed Web-Server Systems, *ACM Computing Surveys*, Vol.34, No.2, pp.263–311 (2002).
2) Pai, V.S., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W. and Nahum, E.: Locality-Aware Request Distribution in Cluster-based Network Servers, *In Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems* (1998).
3) Aron, M., Druschel, P. and Zwaenepoel, W.: Efficient Support for P-HTTP in Cluster-based Web Servers, *In Proceedings of the USENIX 1999 Annual Technical Conference* (1999).
4) Fujita, H., Matsuba, H. and Ishikawa, Y.: TCP Connection Scheduler in Single IP Cluster, *8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'08)*, pp.366–375 (2008).
5) Spatscheck, O., Hansen, J.S., Hartman, J.H. and Peterson, L.L.: Optimizing TCP forwarder performance, *IEEE/ACM Transactions on Networking*, Vol.8, pp.146–157 (2000).
6) Postel, J.B.: SIMPLE MAIL TRANSFER PROTOCOL, RFC 821 (1982).
7) Mathis, M., Mahdavi, J., Floyd, S. and Romanow, A.: TCP Selective Acknowledgment Options, RFC 2018 (1996).
8) Jacobson, V., Braden, R. and Borman, D.: TCP Extensions for High Performance, RFC 1323 (1992).
9) Postel, J.: The TCP Maximum Segment Size and Related Topics, RFC 879 (1983).
10) The Netfilter.org project: netfilter/iptables project homepage, `http://www.netfilter.org/`.
11) Damani, O.P., Chung, P.E., Huang, Y., Kintala, C. and Wang, Y.-M.: ONE-IP: techniques for hosting a service on a cluster of machines, *Selected papers from the sixth international conference on World Wide Web*, Essex, UK, Elsevier Science Publishers, Ltd., pp.1019–1027 (1997).
12) Microsoft Corporation: Network Load Balancing Technical Overview, `http://www.microsoft.com/technet/prodtechnol/windows2000serv/deploy/confeat/nlbovw.mspx`.
13) Kerdlapanan, D. and Khunkitti, A.: Content-based load balancing with multicast and TCP-handoff, *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, Vol.2, pp.II–348–II–351 vol.2 (2003).
14) Papathanasiou, A. and Van Hensbergen, E.: KNITS: Switch-based Connection Hand-off, *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE*

Vol.2009-ARC-183 No.1
Vol.2009-OS-111 No.1
2009/4/22

Computer and Communications Societies. Proceedings. IEEE, Vol. 1, pp. 332–341 vol.1 (2002).

15) Aron, M., Sanders, D., Druschel, P. and Zwaenepoel, W.: Scalable Content-aware Request Distribution in Cluster-based Network Servers, *In Proceedings of the USENIX 2000 Annual Technical Conference* (2000).

16) Fielding, R., Gettys, J., Mogul, J., Frystyk, H. and Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1, RFC 2068 (1997).

17) Luo, M.-Y.: On the Effectiveness of Content-aware Load Distribution for Web Clusters, *2006 IEEE International Conference on Cluster Computing*, pp. 1–10 (2006).