2005－OS－100（6）
2005／8／3

シングルIPアドレスクラスタの設計

松葉浩也[†]　石川　裕[†]

単一 IP アドレス上で動作するクラスタの実現のため，TCP/IP のプロトコルハンドリングを I/O サーバと呼ばれるクラスタ内の一台のコンピュータで行い，他のクラスタノードでソケットインタフェースを提供する方法を提案する．この手法では I/O サーバが多数のコネクションを扱う．このように一台のコンピュータが多くの接続を持つ場合，ネットワークのインタフェースキューに多くのパケットが挿入された状態が続き，パケットが TCP 層を離れてから実際に送出されるまでに長い時間を要する．そのため TCP が長い遅延を観測し通信性能が低下する．この問題を解決するため，まずインタフェースキューに挿入するパケットの量を TCP レベルで制限する手法を提案し実装する．この改良した TCP を使用した上で，提案するシングル IP アドレスクラスタを実装し，通信性能を NAT を用いた通信と比較する．提案手法は NAT よりも低遅延，高速であり，複数の通信が平等に帯域を共有することを示す．

The Design of Single IP Address Cluster

Hiroya Matsuba[†] and Yutaka Ishikawa[†]

In order to realize a cluster which runs on a single IP address, we propose a method that a cluster node, called I/O server, performs TCP/IP protocol handling and other nodes, called application nodes, provide a socket network interface. When a single computer handles multiple connections, many packets are queued in the interface queue. Due the long length of the interface queue, packets are forced to wait for a long time until they are actually sent to the network. This results in the long round trip time and this reduce the performance of TCP/IP communication. To avoid this problem, we modify the implementation of TCP protocol so that a number of packets sent to the interface queue is limited. Using this improved TCP, we implement the single IP address cluster and evaluate its performance. In the comparison with communication using NAT, it is shown that our method achieves the short latency communication and equally shares the available bandwidth among the multiple streams.

1. はじめに

近年のパーソナルコンピュータの低価格化により，PC クラスタは並列計算，あるいは高い性能が要求されるサーバなどに広く使用されるようになってきている．プロセッサ単体の性能が以前ほど急速には向上しなくなり，並列化が高速化手法の主流となりつつある中，クラスタは今後も並列化の一つの方法として重要な役割を果たすことが予想される．

現在のクラスタは管理やプログラミングが煩雑である．例えば設定変更の必要が生じた際，管理者はすべてのノードにログインしてそれぞれのノードに同じ設定を行わなくてはならず，手間がかかる上にミスも犯しやすい．またプログラマがクラスタ上で動作するアプリケーションを作成する際には，明示的な通信を記述する必要があり，場合によってはノードの故障への対処も求められる．この煩雑さを軽減するため，我々はオペレーティングシステムのサポートにより単一システムイメージクラスタを提供し，扱いの容易なクラスタを構築することを目指している．本稿では特にネットワークに着目し，シングル IP アドレス上で動作するクラスタを設計する．これは Web Server など，外部にサービスを提供するクラスタ，あるいはグリッドノードとしてのクラスタのようにクラスタ間通信が多く発生するようなシステムで有効性を発揮する．

シングル IP アドレスクラスタはすでに Virtual Server[9] で部分的に実現されている．しかしこれは外部から参照される IP アドレスが単一であるのみで，クラスタ内部では個別の IP アドレスが割り当てられている．したがって管理やプログラミングの負担を軽減するものではない．また複数のコンピュータが同時に外部との通信を行う際には輻輳により通信性能が低下することが知られている [10]．

本研究では，TCP,UDP のプロトコル処理を I/O サーバと呼ばれる 1 台のコンピュータで行い，ソケッ

† 東京大学情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

トインタフェースのみをアプリケーションノードと呼ばれる I/O サーバ以外のクラスタノードで提供する方式を提案する．本方式ではプロトコル処理が単一のコンピュータで行われるためシングル IP アドレスが実現される．またクラスタ内のノード間通信と外部との通信を切り離すため，ノード間通信にはクラスタ向けのプロトコルが選択できオーバヘッドの少ない通信が可能である．提案手法は既存のプロトコルスタックの実装をクラスタ向けに拡張するものである．そこでこれを SAPS(Single-Address Protocol Stack for clusters) と呼ぶ．

一台のコンピュータが多くの通信を管理する場合，そのコンピュータのネットワークインタフェースが持つ送信キュー（インタフェースキュー）に多くの送信待ちデータが留まり，送信待ち時間が長くなる．そのため TCP は長い遅延を観測してしまい性能が上がらない．SAPS の I/O サーバがこの問題の影響を受けるのを避けるため，I/O サーバの TCP 処理にインタフェースキューの長さを制限する機構を加えた上で SAPS を実装している．複数ストリームの存在下で行った性能評価では，SAPS は NAT 用いた通信よりも低遅延であり，またストリーム間で平等に帯域を分け合うことを示す．

2. 設　　計

SAPS の概要を**図 1** に示す．

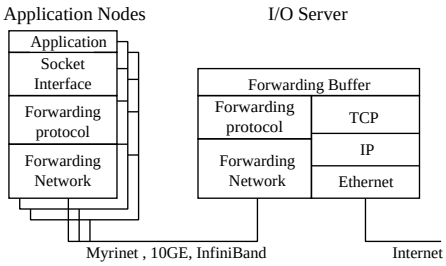

図 1　SAPS の全体図

上図にあるように SAPS ではクラスタを一台の I/O サーバと複数のアプリケーションノードに分割する．それぞれの役割は以下のようになる．

- I/O サーバ
  外部への物理的な接続を持ち，クラスタを代表する IP アドレスを持つ．クラスタ内のプロセスが行うすべての通信について，TCP あるいは UDP のプロトコル処理を行う．クラスタ外部からは I/O サーバ一台のみが見えるためシングルシステムイメージとなる．
- アプリケーションノード
  アプリケーションへのインタフェース（ソケット）を提供する．実際のデータ処理を行うアプリケーションが動作する．

I/O サーバとアプリケーションノードは「転送用ネットワーク」で接続される．このネットワークはクラスタ内のアプリケーションが送受信するデータ転送，あるいは I/O サーバとアプリケーションノード間の制御情報の伝達に使用される．具体的な転送用ネットワークとしては Myrinet, 10G Ethernet, InfiniBand などが想定されており，それぞれのデータリンクに適した「転送用プロトコル」が信頼性のあるコネクション指向通信を提供する．

2.1 SAPS の動作

SAPS の基本動作は次のようになる．

(1) ソケットの作成
ユーザアプリケーションがアプリケーションノード上で socket システムコールを発行すると，アプリケーションノードはソケットを管理するデータ構造を作成するとともに，I/O サーバにソケットの作成を伝える．I/O サーバは TCP あるいは UDP 通信の管理に必要なデータ構造を準備する．I/O サーバとアプリケーションノード双方で作成されたソケット管理のためのデータ構造は，転送用ネットワークのコネクションで接続される．

(2) bind
アプリケーションが bind システムコールを発行した場合，要求は I/O サーバに伝えられる．I/O サーバは bind 処理を行った上で結果を返す．

(3) listen
listen システムコールも I/O サーバに伝えられ，I/O サーバが結果を返す．

(4) connect
アクティブオープンの時はアプリケーションノードが connect 要求と接続先などのパラメータを I/O サーバに送信する．I/O サーバは接続処理を行い結果を返す．

(5) 接続の受け入れ
ユーザプロセスはアプリケーションノードで accept システムコールを発行し新しいコネクションの到着を待つ．クライアントからの接続要求はまず I/O サーバに到着する．I/O サーバはコネクション確立の TCP 処理を行った上で listen しているアプリケーションノードにコネクションの到達を通知する．それを受けたアプリケーションノードは accept の成功をユーザプロセスに通知する．socket システムコールと同じく，新しい接続を管理するデータ構造が I/O サーバ，アプリケーションノード双方で作成され転送用ネットワークのコネクションで接続される．

(6) データ送受信
外部のコンピュータからユーザプロセスに届いたデータは I/O サーバが TCP/IP プロトコル処理を行った上でアプリケーションノードに転送する．ユーザプロセスからの送信はアプリケーションノードが I/O サーバに転送し，I/O サーバが TCP パケットを生成した上でクライアントに送信する．

(7) その他システムコール

上記の他にも setsockopt など様々なシステムコールがあるが，それが TCP あるいは UDP に関するものであればI/O サーバに転送する．転送するものには例えば TCP での Nagle アルゴリズムを無効化する NODELAY オプションのセットが挙げられる．一方ノンブロッキング通信の設定など，ソケットインタフェースのみで処理が可能なものについてはアプリケーションノードが処理をする．

2.2 I/O サーバの設計

I/O サーバは TCP のプロトコル処理，アプリケーションノードからのシステムコール要求の処理およびアプリケーションノードとクライアント間のデータ通信の中継を行う．I/O サーバは転送用ネットワークのプロトコルスタックと TCP/IP,UDP/IP のプロトコルスタックの双方を持ち，その間のプロトコル変換を行う．

2.2.1 問 題 点

I/O サーバには多数の TCP/IP 接続を扱う上での問題が存在する．一般にネットワークインタフェースは送信待ちデータを一時的に保管するインターフェースキュー (IFQ) を持つ．一台のコンピュータが多くの TCP 接続を持つ場合，それぞれの接続が少量の送信を行うだけで IFQ には多くの送信待ちパケットが溜まってしまう（**図 2**）．

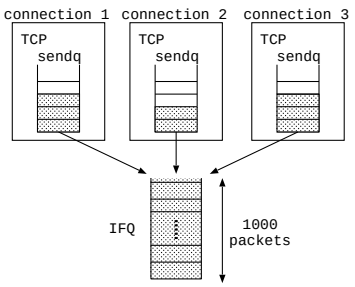

図 2 インタフェースキュー

現在の Linux では IFQ の長さの規定値は 1000 パケットである．Ethernet の標準的な最大サイズである 1500byte のパケットが 1000 個挿入されている場合，IFQ には 1.5Mbyte のデータが留まる．これは 1Gbps の速度で 12ms を要する量である．この状態で新たな接続が開始された場合，送信パケットが TCP を離れてから実際に送出されるまでに 12ms かかる．これは TCP にとっては通信路に 12ms の遅延が加わったのと同じ状況である．この遅延を疑似遅延と呼ぶことにする．TCP が大きな遅延を観測すると，スロースタートモードにおける送信量の増加速度が遅くなり，通信性能が低下する．これは特にスロースタートモードでの送信が大半を占めるような通信量の少ない接続で問題となる．

疑似遅延は SAPS 固有の問題ではなく，通常のコンピュータでも起こり得る．ただし SAPS の I/O サーバは多くの TCP 接続を管理するため IFQ にデータが留まりやすく，対策を講じることの重要性は高い．

2.2.2 プロトコルスタックの改良

IFQ に多くのパケットが留まることが問題であるので，最も簡単な解決は IFQ のサイズを小さくすることである．しかしこれは IFQ への挿入に失敗する可能性を上げることになる．TCP は IFQ の満杯状態を輻輳と見なすが，この動作による弊害は文献 11) で指摘されている．また同じ文献では IFQ の満杯状態を輻輳と見なさないことも必ずしも適切でないことが指摘されており，いずれにせよ IFQ への挿入の失敗は避けなくてはならない．

そこで TCP レベルで IFQ に挿入するパケットの数を制限する方法を提案する．TCP はパケット送信の際，輻輳ウィンドウや広告されたウィンドウサイズを確認し，送信の可否を判定している．この確認の最後に IFQ サイズを調べる仕組みを追加し，IFQ の長さが制限数以上の時はパケットを送出することなく送信キューに残すようにする．送信キューに残ったパケットの送信を試みるのは IFQ サイズが減じられたとき，つまり送信完了を伝える割り込みの発生時とする．さらに，IFQ サイズ制限のみでは単一のコネクションが規定の IFQ サイズを独占することによってスタベーションを発生させる危険性があるため，IFQ サイズの制限で送信待ちになった TCP 接続をキュー管理し，一定数以上連続で IP 層にパケットを送出した接続はキューの最後に回すようにする．

この IFQ サイズ制限の動作は，通常の TCP が IFQ が満杯の状態で見せる挙動と同じであり，アプリケーションに悪影響を与えるものではない．

2.3 アプリケーションノードの設計

図 1 にあるように，SAPS はアプリケーションノードのカーネルでソケットインタフェースを提供する．このソケットはアプリケーションに対しては通常の TCP あるいは UDP ソケットとまったく同じように振る舞うため，ソケットを使用する既存アプリケーションのバイナリが再コンパイル，再リンクなしでそのまま動作する．

2.4 転送用ネットワーク・転送用プロトコルの設計

転送用ネットワークの物理層，データリンク層には Myrinet[2], InifiniBand[3], 10G Ethernet などが想定されている．転送用プロトコルは信頼性のあるコネクション指向通信を提供するために，データリンク層が提供する機能に応じて設計，実装される．Myrinet や InfiniBand のように信頼できるデータリンクを提供するネットワークでは，転送用プロトコル部分はフローコントロールとコネクション管理機能を提供する．Ethernet のようにパケット消失の可能性のあるネットワークではフローコントロール，コネクション管理に加え，信頼性確保のための再送なども行う．いずれの場合も転送用ネットワークはカーネルから使用できるインタフェースを提供する．また，割り込みハンドラ内，つまりプロセスコンテキストが存在しない場合

にも使用されるので，送受信はすべてノンブロッキングであることが求められる．

3. 実　　装

SAPS はすべてカーネル機能として実装されている．現在の実装は Linux 2.6.12 を用いている．

3.1 I/O サーバ

I/O サーバでは TCP,UDP から SAPS 転送用プロトコル，あるいは SAPS 転送用プロトコルから TCP,UDP へのプロトコル変換が実装される．現時点では TCP のみが実装されている．TCP/IP のプロトコル処理は Linux オリジナルのものを変更することなく使用しており，ユーザインタフェース部分のみを SAPS 用に修正している．この修正はカーネルモジュールでは不可能であり，カーネル本体を書き換えている．

送信は**図 3** に示すように，本来システムコールを通じてユーザープログラムからコピーされるデータを転送用ネットワークから受け取る．受け取ったパケットは転送プロトコルのヘッダを持つため，それを TCP/IP ヘッダに書き換え，IP 層に送出する．受信は**図 4** のように，IP 層から受け取ったパケットを TCP レイヤーが処理した上で，本来はユーザープログラムにコピーするデータを転送用ネットワークに送出する．この場合も受信したパケットの TCP/IP ヘッダ部分を転送プロトコルのものに書き換える．

転送用ネットワーク，IP ネットワークともにパケットは共通の構造体 sk_buff で表現されるため，プロトコル変換に際してデータ構造の変更や送受信データのコピーは必要なく，ヘッダの書き換えのみが発生する．

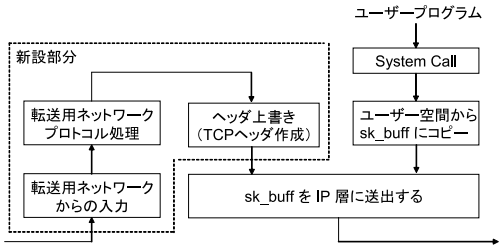

図 3　I/O サーバの送信動作

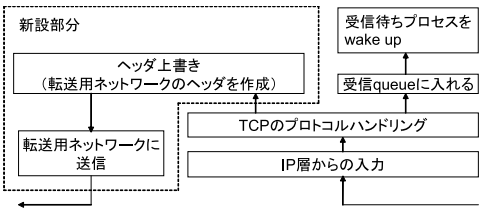

図 4　I/O サーバの受信動作

3.1.1 I/O サーバ実装の問題

I/O サーバの実装にはプロセスコンテキストが存在しないことによる問題がある．システムコールを介してカーネルに送信要求が届いた場合，そのシステムコールハンドラはプロセスコンテキストを持った状態で動作する．これは「待ち」が必要な際にはスリープできる（スケジューラを呼び出し他のプロセスに実行を譲る）ことを意味する．例えば TCP の connect システムコールハンドラは SYN パケットを送出した後 SYN,ACK パケットを受信するまでスリープする．ところが SAPS のために追加実装した部分は転送用ネットワークからの受信を知らせる割り込みハンドラで動作しているため，プロセスコンテキストがなく，スケジューラを呼び出すことはできない．したがって，オリジナルの TCP 実装のうち，プロセスコンテキストの存在を仮定して作られている部分はそのままでは動作しない．

最も簡単な解決策は各ソケットに対し一個のカーネルスレッドを割り当てることである．カーネルスレッドはプロセスと同様のコンテキストを持つため，スケジューラを呼び出すことによるスリープが可能である．しかし，これではスケジューラのオーバヘッドが大きくなることが容易に予想されるのでこの方法は用いない．現在実装されているスリープの回避方法を**図 5** に示す．

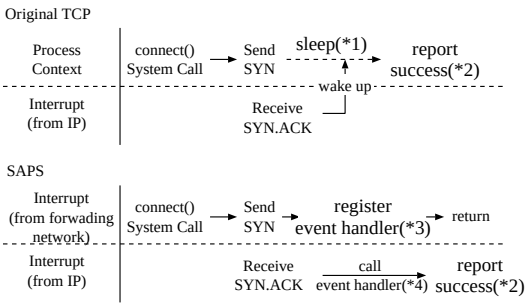

図 5　sleep の処理

これは先に挙げた connect システムコールのハンドラである．図の上段にあるように，プロセスコンテキストが存在する場合は SYN,ACK の到着を待つことができる（図中*1）．割り込みハンドラ内ではそれが不可能であるため，スケジューラを呼び出す代わりにスリープ後に相当する部分（ここでは完了をユーザープログラムに報告する処理，図中*2）をイベントハンドラとして登録し（図中*3），割り込みハンドラ自体は終了する．そして SYN,ACK を受信した際にはスリープ中のプロセスを wake up する代わりに登録されているイベントハンドラを呼び出す（図中*4）．具体的な実装としてはプロセスの wait queue の代わりにイベントハンドラの queue を準備し，wait queue から dequeue し wake up する部分をイベントハン

表 1　評価環境

	I/O server, delay router	Application Node	Client
CPU	AMD Opteron 248 (2200MHz) × 2	Xeon 3.4GHz × 2	Xeon 2.8GHz × 2
Chipset	AMD 8131	Intel E7520	ServerWorks CG-LE
Memory	4Gbytes	4Gbytes	2Gbytes
PCI	PCI-X (64bit 133MHz)	PCI-X (64bit 100MHz)	PCI-X (64bit 100MHz)
Ethernet	Broadcom BCM5703X GE	Intel Pro/1000 Server	Intel Pro/1000 Server
Myrinet	Myrinet XP	Myrinet XP	

ドラの dequeue および呼び出しに置き換えることになる．

3.2 アプリケーションノード

アプリケーションサーバでは TCP ソケットと同じ動作をするソケットインタフェースの新しい実装を提供する．Linux カーネルには新たなソケットの追加のためのインタフェースが準備されているため，カーネルを書き換えることなくモジュールのみで実装可能である．

3.3 転送用ネットワーク

転送用ネットワークの物理ネットワークとして Myrinet を用いる．通信ライブラリは PM/Myrinet[8] であるが，PM/Myrinet はハイパフォーマンスコンピューティングをターゲットに設計されてあり，以下の理由で SAPS の転送用ネットワークとしてそのまま使用することができない．

- ユーザモード通信
  ハイパフォーマンスコンピューティングではシステムコールのオーバヘッドを削減することが重要であるため，PM/Myrinet は OS をバイパスしてユーザプログラムが直接通信を制御している．そのため OS 内部から使用できるインタフェースを持たない．
- Busy Wait による受信
  科学技術計算ではではメッセージの受信待ち時に他の有用な処理が存在することは少ない．したがって受信待ち時には計算資源を解放することよりも割り込みのオーバヘッドをなくすことが重要であるため，Busy Wait をするように設計されている．SAPS の転送用ネットワークはカーネル内での通信なので Busy Wait はシステム全体をロックする危険性がある．

PM/Myrinet を SAPS の転送用ネットワークとして使用するため，ライブラリをカーネルレベルに移植した上でファームウェアも変更し割り込みベースの受信を可能にした Kernel Mode PM/Myrinet を作成した．

Kernel Mode PM/Myrinet は信頼性のある通信を提供するがコネクションを持たない．そのため SAPS で使用するコネクションは SAPS 用転送プロトコルが提供する．この転送用プロトコルでは各コネクションが Stop-Go 方式のフローコントロールを行う．

4. 性能評価

本節では SAPS を Web Server として使用することを想定し，クラスタから外部へのデータを送信の性能を測定する．

4.1 評価環境

以降ではすべて**図 6** に示す環境で評価を行う．それぞれのコンピュータの仕様を**表 1** に示す．SAPS は Web Server やグリッドのように遅延のあるネットワークで使用されることを想定しているため，図に Delay と示された遅延ルータで往復 2ms の遅延を発生させている．遅延の付加には Linux カーネルに標準で実装されている netem[6] を用いた．遅延時間の 2ms は，環境の良好なインターネットにおける遅延を目安に選択した．実際には例えば学術ネットワーク (SINET) のみを通過する東京都内の通信で 2ms から 5ms 程度のラウンドトリップタイムを観測する．

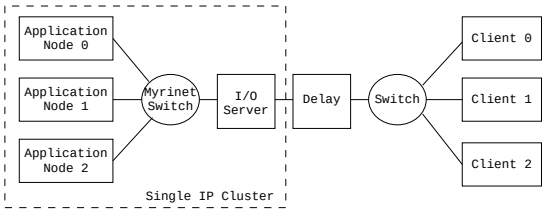

図 6　評価用ネットワーク

4.2 性能比較対象

以下のすべての評価において比較対象として NAT の入る IP ネットワークを用いる．これは外部のコンピュータに対し仮想的にシングル IP クラスタを提供する Virtual Server[9] が NAT をベースに設計されているため比較対象として選択した．NAT を用いたネットワークは図 6 の Myrinet を Ethernet に置き換え，I/O Server を NAT ルーターに置き換えた構成となる．このとき用いた Ethernet スイッチの使用を**表 2** に示す．なお Myrinet の Ethernet エミュレーション機能を用いることで性能比較を行うことも考えられるが，この場合 2Gpbs の Myrinet から入力されるパケットを 1Gpbs の Ethernet にルーティングすることになり，ルーターで大量のパケットが失われることが予想される．これでは公平な比較は不可能なので NAT に関しては 1Gpbs の Ethernet を用いる．

4.3 基本性能

基本性能として Point-to-Point のラウンドトリップタイムとバンド幅を計測する．ラウンドトリップタイムはアプリケーションノード 0 のユーザープログラ

表 2 Ethnernet Switch

Model	Cisco Catalyst 3750 24-T-S
Queue	75 for input, 40 for output
Buffer	12MBytes shared
Flow Control	On for RX

ムとクライアント 0 のユーザープログラムが 4byte のメッセージを往復させるのに要する時間を測定する．バンド幅はアプリケーションノード 0 がクライアント 0 に特定量のデータを送信し，クライアント 0 からの受信確認メッセージが届くまでの時間を測定することにより計測している．この評価のみ遅延ルーターは通過するのみの設定とし，人工的な遅延は加えていない．

ラウンドトリップタイムの測定結果を**表 3** に示す．5 回の測定の最大，最小，平均を示した．測定毎のばらつきが大きいが，おおむね SAPS の方が高速である．これは一部に低遅延の Myrinet を使用していることが影響していると考えられる．**図 7** がバンド幅の測定結果である．3 回の測定の平均値をプロットした．データ転送量の少ないコネクションはスロースタートが大部分を占めるのでバンド幅も遅延の影響を受けやすい．そのため SAPS の方が高速になっている．最大バンド幅はほぼ同じである．

表 3 ラウンドトリップタイム (μs)

	最小	最大	平均
SAPS	165	243	217.6
NAT	210	334	255.6

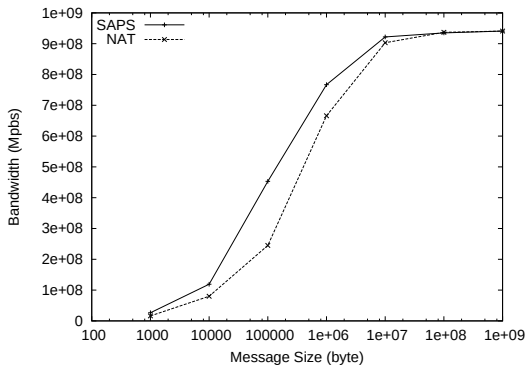

図 7 Point-to-Point バンド幅

4.4 複数ストリーム時の性能

4.5 測 定 方 法

アプリケーションノード 1,2 がそれぞれクライアント 1,2 にメッセージを送信し続ける中，アプリケーションノード 0 からクライアント 0 に対し 100Kbyte または 100Mbyte のデータを 1 回送信し，性能を測定する．今後，測定対象の通信をフォアグランド通信，送信を続ける 2 本の通信をバックグラウンド通信と呼ぶ．100Kbyte の送信量は，すべての通信が TCP のスロースタートモードの間に終了する場合の評価のため選択した．また 100Mbyte はスロースタートモードの時間が無視できるほど短く，フォアグランド通信とバックグラウンド通信が対等である場合の評価のために選択した．

結果は，常にほぼ一定量のトラフィックがあると見なすことのできる程度に多数のアクセスがある Web Server に新たなコネクションを張った場合の性能，あるいは巨大なファイルを提供している ftp サーバで，あるクライアントのダウンロード中に別のクライアントが観測する性能に相当する．

性能評価はパケットの送信時刻を記録したデータを解析することにより行う．パケットの通過ログは以下の 2 カ所で測定する．

- 遅延ルーターの IP 層
  実験環境中に示した遅延ルーターの IP 層に通過した TCP パケットのシーケンス番号と時刻を記録する機構を加え，フォアグラウンド通信の SYN パケットから FIN パケットの間に通過した全パケットを記録する．
- 送信元コンピュータの TCP 層
  送信元コンピュータの TCP 層（SAPS に関しては I/O サーバになる）が IP 層にデータを送出した時刻および ACK を受け取った時刻をそれぞれ送信シーケンス番号および受信 ACK 番号とともに記録する．

これらデータ取得にかかるオーバヘッドは無視できるほど小さい．

4.6 結　　果

4.6.1 100Kbyte 送信

フォアグランド通信として 100Kbyte を送信した場合のパケット送受信時刻および ACK 受信時刻をを**図 8** に示す．

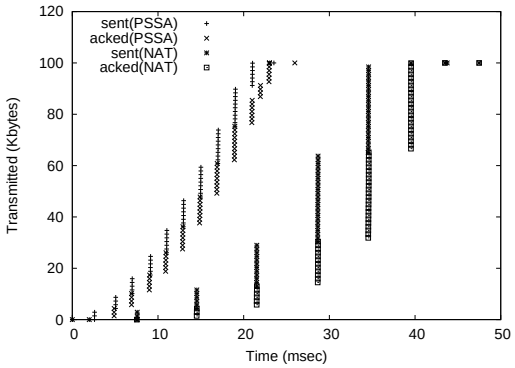

図 8 100Kbyte 送信

SAPS, NAT とも数回に分けて不連続な送信になっており，それに伴い不連続に ACK が受信されている．

これはスロースタートモードの動作であり，輻輳ウィンドウに達するまで連続的にパケットを送出し，その後 ACK を待つ動作は正常である．また，複数のパケットが 1Gbps を超えるスピードで送信されているが，これはインタフェースキューへの挿入のみで TCP/IP の送信操作が完了するためである．

スロースタートモード時の性能を左右する遅延時間は SAPS が 2.5ms，NAT が 6ms であり，この差のために SAPS が NAT を上回る結果となっている．NAT 時に大きな観測する理由は，スイッチ内に長いキューができていることだと推測されるが，詳細は調査中である．Ethernet のフローコントロール機能を有効にしているため NAT でもパケットロスは発生していない．フローコントロールを無効にした場合はパケットロスが発生することが確認されている．

一方，IFQ サイズを制限する機構を無効にして測定した結果は**図 9** のようになる．NAT に関しては上で示したものの再掲である．

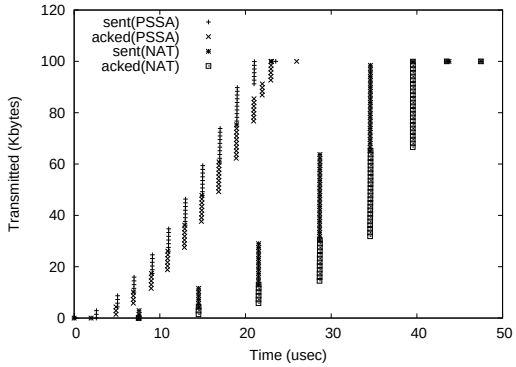

図 9 100Kbyte 送信（IFQ サイズ制限なし）

IFQ サイズ制限を行わない場合，SAPS は 20ms の遅延を観測し，結果として 100Kbyte の送信に NAT の 2.5 倍ほどの時間を要している．この結果から IFQ サイズ制限が有効に機能していることが確認できる．

4.6.2 100Mbyte 送信

フォアグランド通信として 100Mbyte を送信した場合について，その動作を 2 本のバックグラウンド通信の動作と共に示す．**図 10** に SAPS 使用時に遅延ルーターで取得したパケットの通過情報を示す．3 本のストリームが対等な関係で帯域を共有していることがわかる．

IFQ サイズ制限を行わないときの SAPS の通信状況を**図 11** に示す．帯域共有に不公平が生じていることがわかる．これは通常の TCP が IFQ への挿入時に公平性を考慮しないのに対し，IFQ サイズ制限では飢餓状態を防止するため，公平な IFQ への挿入機構を加えたためである．

図 12 に NAT 使用時の通信状況を示す．通信開始から 1.6 秒ほどの間，3 本のストリームの使用帯域が

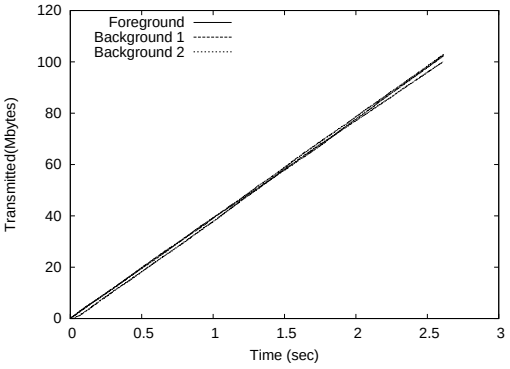

図 10 100Mbyte 送信 (SAPS)

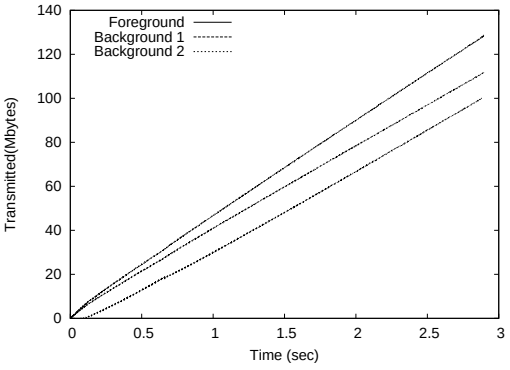

図 11 100MByte 送信 (SAPS,IFQ サイズ制限なし)

不平等な状態が続いていることがわかる．NAT 使用時は TCP の輻輳回避アルゴリズムによりバンド幅が調整される．この場合，3 本のストリームがパケットロスを起こさないバンド幅に収束すると，それが不平等な帯域共有であってもその状態で安定してしまう．1.6 秒程度で速度変化が起こっているのは，そこで最高速のストリームにパケットの消失が発生したためである．

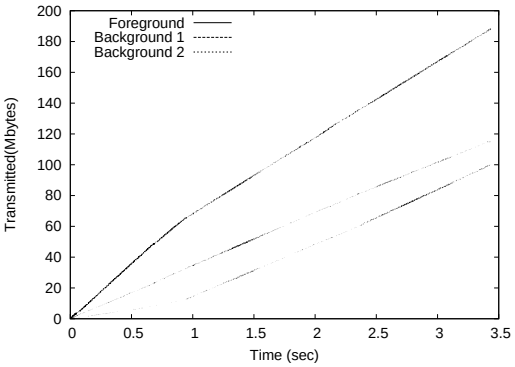

図 12 100MByte の送信 (NAT,3stream)

上に示したいずれの場合も，3 本のストリームは合計でギガビットイーサネットのバンド幅を余すことなく使用している．ただし通信の平等性の観点から，IFQ サイズ制限を設けた SAPS の動作が最も望ましいと言える．

5. 関連研究

シングルシステムイメージクラスタを提供する研究として進行中のものに Kerrighed[5] がある．このシステムは分散共有メモリ，プロセスマイグレーションなどを提供しているが，ネットワークに関してはシングルシステムイメージを提供していない．同様にシングルシステムイメージクラスタを実現するオープンソースプロジェクトとして OpenSSI[7] がある．このプロジェクトはネットワークのシングルシステムイメージ化のために Virtual Server[9] を使用しているので，本論文で示した NAT による通信と同じ問題を持つ．

TCP Splicing[4] はサーバ，クライアント間のコネクションを途中のコンピュータで分割し，認証やロードバランスなどを行うプロキシの高速化技術である．TCP Splicing は TCP のコネクションを管理し，他のコンピュータへの転送をカーネル内で行う点で SAPS と共通の特徴を持つが，SAPS の I/O サーバはネットワークを流れるパケットのアプリケーション層を調査することを目的としたものではない．

IFQ サイズの制限は，バンド幅を適切な速度に制限するものと考えることができ，一種のペーシング [1] と考えることもできる．ただし，広く研究されているペーシングは複数台のコンピュータが 1 本のボトルネックリンクを共有している環境をターゲットにしているのに対し，本論文で提案した IFQ サイズ制限は，単一コンピュータ内で発生する問題に対処する手法である．両者は相反するものでなく，IFQ サイズ制限を行った上で，外部でのパケットロスを軽減するためにペーシングを行うことも可能である．

6. おわりに

本稿ではシングルシステムイメージクラスタのためのネットワークシステムとしてシングル IP クラスタの実現方法を述べた．提案した SAPS は，TCP プロトコルハンドリングを I/O サーバと呼ばれる一台のコンピュータで行い，ユーザプログラムが動作するアプリケーションノードでは TCP ソケットと互換性のあるソケットインタフェースを提供する．そして I/O サーバとアプリケーションノードはクラスタ専用ネットワークで接続する．

一台のコンピュータに多数の TCP コネクションが存在する場合，IFQ に多くのパケットが留まり性能を悪化させる．I/O サーバでこの問題が発生しないよう，インタフェースキューに挿入するパケット数を TCP レベルで制限する方法を提案し実装した．この改良した TCP を用いて SAPS を実装し，性能評価を行ったところ，SAPS は NAT を用いた通信と比較して低遅延であり，複数ストリーム間で平等に帯域を分け合うことが示された．

今後の課題としてはアプリケーションレベルでの評価を行い，SAPS の有効性あるいは問題点を明らかにする必要がある．また IFQ サイズ制限については適切なアルゴリズムとパラメーターを詳しく調査する必要がある．

謝辞 本研究の一部は，科学研究費補助金・特別研究員奨励費 17-11878 の援助および文部科学省「eSociety 基盤ソフトウェアの総合開発」の委託による．

参考文献

1) Amit Aggarwal, Stefan Savage, and Thomas Anderson. Understanding the performance of TCP pacing. In *INFOCOM (3)*, pp. 1157–1165, 2000.
2) Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, Vol. 15, No. 1, pp. 29–36, 1995.
3) InfiniBand. http://www.infinibandta.org.
4) David A. Maltz and Pravin Bhagwat. TCP splicing for application layer proxy performance. Research Report RC 21139, IBM, March 1998.
5) Christine Morin, Pascal Gallard, Renaud Lottiaux, and Geoffroy Vallée. Towards an efficient single system image cluster operating system. *Future Gener. Comput. Syst.*, Vol. 20, No. 4, pp. 505–521, 2004.
6) Netem. http://developer.osdl.org/shemminger/netem/.
7) OpenSSI. http://openssi.org/.
8) Toshiyuki Takahashi, Shinji Sumimoto, Atsushi Hori, Hiroshi Harada, and Yutaka Ishikawa. PM2: High performance communication middleware for heterogeneous network environments. 2000.
9) Wensong Zhang. Linux Virtual Servers for Scalable Network Services. *Linux Simposium*, 2000.
10) 高野了成, 工藤知宏, 児玉祐悦, 松田元彦, 手塚宏史, 石川裕. GridMPI のための TCP/IP 輻輳制御実装方式の検討. 情報処理学会 SWoPP'04, 2004.
11) 高野了成, 石川裕, 工藤知宏, 松田元彦, 児玉祐悦, 手塚宏史. 並列アプリケーション実行における TCP/IP 通信挙動の解析. インターネットコンファレンス 2003 論文集、日本ソフトウェア科学会研究会資料シリーズ No. 26, 2003.