MPI ライブラリと協調する TCP 通信の実現

松田元彦[†] 高野了成[†,††] 石川裕[†††]
工藤知宏[†] 児玉祐悦[†]
岡崎史裕[†] 手塚宏史[†]

TCP は連続するストリーム通信で効率的な通信が行なえるように設計されている．一方，典型的な MPI アプリケーションは計算と通信のフェーズを繰り返し，その通信負荷は断続的であり TCP による通信は非効率的になる．そこで，断続する通信を連続に見せるため TCP に 3 点の変更を加える．通信開始時点でのペーシング，Retransmission Time Out 時間の短縮，通信フェーズ毎の通信パラメータの保存復元である．Linux 上の TCP スタックに実装し，その効果を NPB ベンチマークによって確認した．比較的定常的な通信を行なう FT，MG ベンチマークでは効果はなく，通信量の非常に少ない LU では逆に性能が低下した．しかし，それ以外の BT，CG，IS，SP ベンチマークでは，10%から 30%の性能向上が確認できた．

TCP Layer Adapted to MPI Library

Motohiko Matsuda ,[†] Ryousei Takano ,[†,††] Yutaka Ishikawa ,[†††]
Tomohiro Kudoh ,[†] Yuetsu Kodama ,[†] Fumihiro Okazaki [†]
and Hiroshi Tazuka[†]

Typical MPI applications work in phases of computation and communication, and messages are exchanged in relatively small chunks. This behavior is not optimal for TCP because it is designed only to handle a contiguous flow of messages efficiently. Modifications to make TCP observe these chunks as if it were a contiguous flow will be expected to improve performance of TCP. Three simple modifications are actually implemented in the Linux TCP stack and evaluated: pacing at start-up, cutting RTO (Retransmission Time Out) time, and saving/restoring TCP parameters at changes of computation phases. Evaluation using the NAS Parallel Benchmarks shows that FT and MG benchmarks get no merit because they have small but steady communication, LU benchmark gets worse because it has very small amount of communication, but other BT, CG, IS, and SP benchmarks get 10 to 30 percent of improvement.

1. はじめに

グリッド環境などで TCP/IP 通信を用いて MPI プログラムを実行する場合，ネットワークの物理バンド幅は広くても十分な通信性能が得られないことがある．高い実効性能を得ることを目的とした TCP 通信プロトコルは数多く開発されているが，これらのプロトコルは連続して通信が行われる場合を想定している．しかし，典型的な MPI プログラムは，計算フェーズと通信フェーズを交互に繰り返す．このような通信状況が断続的，周期的に変動する通信では TCP はうまく機能しない．

TCP は遅延やバンド幅を観測しており，そのパラメータに従って通信量を制御している[7]．特に重要なパラメータとして，輻輳ウィンドウサイズ（cwnd）がある．cwnd は TCP が利用する帯域遅延積を保持しており，ネットワーク中を cwnd 量のメッセージがインフライト中になるように自動的に調節される．MPI では，通信パターンにより利用するバンド幅が異なるため，cwnd 量もそれに従って異なる値になる．一対一通信では，ノードはバンド幅全体を利用できることが多いため cwnd が大きく，メッセージ送信量も大きい．一方，全対全通信では，ノード間でバンド幅が共有されるため一般に cwnd は小さく，メッセージ送信量も小さい．

通信フェーズが変わる時点で利用可能なバンド幅が急激に変化するが，TCP はその変化に追随できない．そのため，一対一通信から全対全通信に変わる時点で

† 産業技術総合研究所 グリッド研究センター
Grid Technology Research Center, National Institute of Advanced Industrial Science and Technology
†† 株式会社アックス
AXE, Inc.
††† 東京大学 大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

は，cwnd が大きい状態で全対全通信を行なうので輻輳が発生する．逆に，全対全通信から一対一通信に変わる時点では，cwnd が小さい状態で一対一通信を行なうのでバンド幅を有効に利用できない．

MPI のトラフィックは断続的であるが通信データ量自体は多い．断続する通信を連続しているかのように扱うことができれば，MPI のトラフィックに対しても TCP はうまく動作すると考えられる．そこで，MPI ライブラリと協調し，かつネットワークの実効性能を高めるための変更を TCP に実装し評価を行なった．

評価はシミュレーションによるものではなく，Linux 上に実装し，WAN エミュレータを使ったクラスタ環境で評価した．評価には MPI の代表的なベンチマークである NPB（NAS Parallel Benchmarks）を使用した．

TCP の変更は他の通信への影響を考慮しなければならない．ここでの TCP に対する変更は，標準 TCP に比べてバースト的なトラフィックの発生を抑えるので，他の通信への影響が小さくなるように機能すると考えられる．

この後，2 節で断続通信での TCP の挙動を示す．3 節で実現方法を説明し，4 節で単純なストリームでの評価と NPB ベンチマークによる評価を行なう．5 節はいくつかの問題を取り上げ，6 節で関連研究を述べ，最後にまとめを述べる．

2. MPI トラフィックと TCP の挙動

2.1 TCP の動作と問題点

TCP では，送信側が輻輳ウィンドウと呼ばれるバッファ領域のサイズを変化させることにより送信量を調整する．輻輳ウィンドウサイズ（cwnd）は往復遅延時間（RTT: Round Trip Time）内に送信できるデータ量を規定している．送信側は cwnd 量のパケットを送信することができる．

TCP 通信の定常状態では，cwnd 量のパケットがネットワークをインフライト中（送信済みで ACK が返ってきていない状態）であり，ACK の受信まで次の送信は行なわれない．送信側は，送信済みパケットの ACK を受け取ると，そのパケットを輻輳ウィンドウから削除する．その時点で次のパケットが送信可能になる．このように，送信は ACK の受け取りタイミングにより起動される．これを ACK クロッキングという．cwnd が帯域遅延積に等しくなった場合には ACK クッキングによりパケット送信間隔がほぼ均等に調整される．MPI の通信では，通信の断続による無通信期間がある．そこでは ACK クッキングが機能しないため，通信バーストが発生し輻輳が起こり易くなる．

TCP 通信の通信開始時には，スロースタートと呼ばれる送出量制御機構が用いられる．標準 TCP のスロースタートは，cwnd を最小値から始めて ACK を受けとる毎に cwnd を 1 ずつ大きくしていく．スロースタートは，cwnd がスロースタート・スレッショルド値（ssthresh）に達するまで継続される．スロースタート中，cwnd は比較的急速に増加するように設計されている．しかし，ACK の受信には RTT 時間がかかるので，遅延が大きな環境では cwnd が小さい期間が長くなり有効バンド幅が小さくなる．ssthresh はスロースタート直前の cwnd の値を必要に応じて補正後，保存している変数である．MPI の通信では，スロースタートが高頻度で発生するためバンド幅の利用効率が低下する．

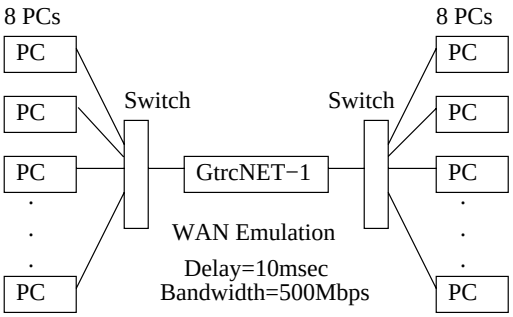


図 1 実験環境

表 1 PC クラスタ諸元

ノード PC	
CPU	Pentium4 2.4C（2.4 GHz）
マザーボード	Intel D865GLC
メモリ	512KB DDR400
NIC	Intel 82547EI（オンボード CSA 接続）
OS	Fedora Core 2（Linux-2.6.9-1.6_FC2）
ネットワークスイッチ	
	CISCO Catalyst 3750G-24T

TCP の高速再送[1] では，パケット・ロスを後続パケットの到着により検出する．受信側は，喪失したパケットの後続パケットが到着した場合，喪失以前のパケットに対する ACK を再送する．送信側では，その ACK（重複 ACK）の受信によりパケット・ロスを検出し，再送を行なう．しかし，通信の終端で後続パケットが存在しない場合，高速再送は機能しない．その場合，送信側は ACK を受け取っていないという受動的なタイムアウトによりパケット・ロスを検出することになる．これには RTO（Retransmission Time Out）と呼ばれるタイムアウト時間が設定されており，RTO 時間の経過後，再送が行なわれる．MPI の通信では，後続パケットの送信が起こらない状況が発生し易く，RTO タイムアウトによるパケット・ロスの検出頻度が高くなる．

まとめると，MPI のトラフィックは断続的で，かつ計算フェーズの移行によって特性が急激に変化する．このような断続する通信で TCP が効果的に機能するには，以下の問題がある．

- 通信フェーズよる通信パラメータの不整合

- スロースタートによる性能低下[2)]
- RTO 時間による無通信時間[2)]

通信パラメータの不整合については，1 節で述べた．一対一通信と全対全通信など通信パターンが変わる時点で，現在の cwnd が適当な値でないため輻輳が発生したりバンド幅の有効に利用できない現象が起こる可能性がある．

スロースタートによる性能低下と，RTO 時間による無通信時間については，この後，具体的な現象を確認する．

2.2 問題点の確認

2.2.1 実験環境

本節では，断続通信で問題になる TCP の挙動を，Linux による PC クラスタを WAN エミュレータで接続した環境上で確認する．同一の実験環境で 4 節の評価も行なう．

実験では TCP の実装として，Linux-2.6.9 の標準 TCP を使用した．これは，標準 TCP である NewReno[1),4)] をベースに輻輳回避に BIC-TCP[16)] を使用するものである．MPI ライブラリには，われわれが開発中のクラスタ用 MPI である YAMPII[6)] を使用した．コンパイラは，C 言語も Fortran も GCC（バージョン 3.3.3）で全て最適化-O4 でコンパイルした．

図 1 に 8 ノードの PC よりなるクラスタ 2 セットを WAN エミュレータを経由し接続した実験環境を示す．**表 1** に PC の諸元を示す．WAN エミュレーションでは，クラスタ間の通信に対して，遅延挿入，バンド幅制限，ルーターバッファの模擬を行なっている．WAN エミュレータには，ネットワーク・エミュレータ GtrcNET-1[9)] を使用した．ルーターバッファの模擬では，バッファ量を超える場合に後続データを破棄するドロップテールを行なうように設定した．

WAN エミュレーションには，以下の設定を使用した．

遅延	10msec
ボトルネック	500Mbps
ルーターバッファ	128KB（ドロップテール）

遅延時間の 10msec という値は，多くの MPI アプリケーションでメリットが得られる遅延時間の範囲として選択している．MPI ベンチマークによる評価では，遅延が 10msec 以上ではスケールしない結果が得られている[10)]．

一部の実験では WAN エミュレーションを行なっていない．その場合は実験結果にその旨を記した．この場合の設定はクラスタ環境を想定した実験であり，遅延 0msec，ボトルネック 1Gbps，ルーターバッファは 16MB である．

さらに，GtrcNET-1 を 1msec 単位のトラフィック観測にも使用した．計測は 1msec 間の平均バンド幅を 1msec 毎に取得している．GtrcNET-1 はハードウェア（FPGA）によって実装されており，トラフィックに影響を与えることなく精密な測定が可能である．トラフィックの観測は全て，MPI のランク（ノード番号）の小さい方のクラスタから大きい方のクラスタへの通信である．計測点はスイッチから GtrcNET-1 への入力である．入力点での計測であるので，バンド幅の制限に関わらず瞬間的にそれ以上の値が観測されることもある．

2.2.2 断続通信時の TCP の挙動

スロースタートによる性能低下は，断続通信を行なう場合に起こる．**図 2** に断続通信時の挙動を示す．一対一のデータ転送を 2 秒間隔で繰り返す実験を行なっている．転送データサイズは 1 回あたり 10MB を連続に送信している．グラフの X 軸は時刻，Y 軸は 1msec あたりのデータ転送速度を示す．グラフではトラフィックが 60〜80MB/s 程度に抑えられているが，これは WAN エミュレータでバンド幅を 500Mbps に制限しているためである．

図 2 右のグラフは，左のグラフ中 4 秒付近の 1 回分の断続を拡大したものである．RTT 時間（20msec）毎に徐々にトラフィックが増えていくスロースタートが行なわれていることが観察できる．各断続通信の開始時点では，バンド幅を有効に利用できていないことが分かる．

TCP では無通信期間が RTO 時間以上続くと，スロースタートに移行する．これは，ネットワークの状態が変化している可能性があるので，スロースタートを使って利用可能バンド幅を観測し直すためである．また，無通信期間が続くと ACK クロッキングが働かなくなり通信バーストが起こるので，通信バーストを回避する目的もある．

このように，断続的な通信ではバンド幅利用に不利なスロースタートが高頻度で観察されることになる．

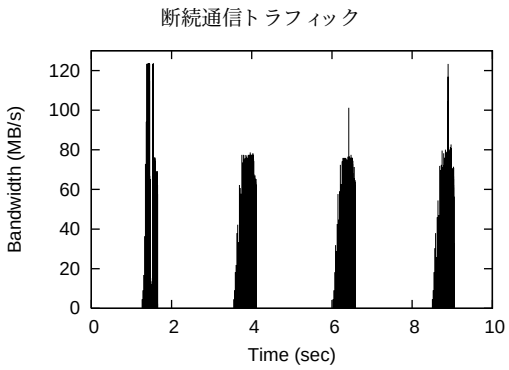
2.2.3 全対全通信時の TCP の挙動

RTO 時間による無通信時間は，全対全通信時に発生する[14)]．**図 3** に全対全通信時の挙動を示す．**MPI_Alltoall** を連続して呼び出す実験を行なっている．この実験では，WAN エミュレータは遅延挿入もバンド幅制限も行なっていない．この設定はクラスタ環境に相当する．

図 3 左のグラフはデータサイズを変えた時のバンド幅を示す．X 軸はデータサイズ，Y 軸は 1 ノード当たりのバンド幅である．バンド幅としては，一つのノードから送信したメッセージのバイト数を通信時間で割ったものを使っている．ここでは自分自身への通信も通信バイト数に含めている．

図 3 右のグラフは，データサイズ 32KB の場合のトラフィックである．連続した **MPI_Alltoall** 呼び出しにも関わらず，通信は不連続になっておりバンド幅を有効に利用できていないことが分かる．

実験環境ではクラスタ間の接続がボトルネックになっているため，全対全通信を行なう場合ボトルネッ



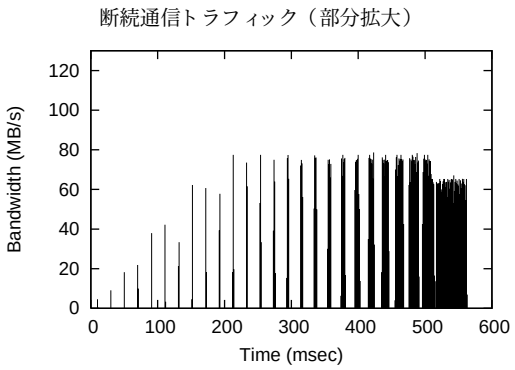
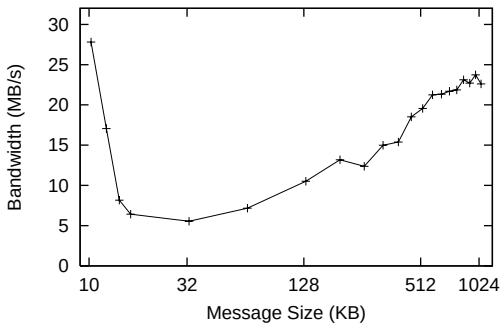


図 2 断続通信時の挙動



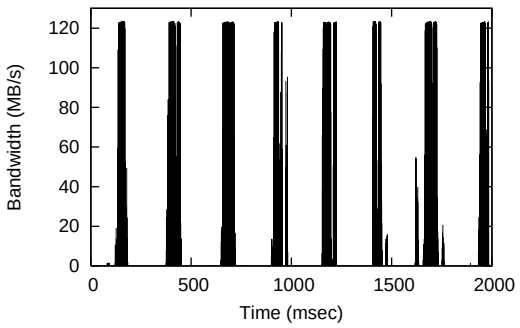


図 3 全対全通信時の挙動

クへ接続するスイッチ上でパケット・ロスが発生する．全対全通信では，他のノードに送信すると同時に他のノードから受信も行なっている．受信データが届くまで全対全通信は終了しないので，パケット・ロス以降の送信が行なわれない状況が発生する．

Linux では RTO 時間に 200msec 以上という長い時間が設定されているため，グラフ上でも 200msec 程度の無通信期間が確認できる．さらに，RTO タイムアウト後はスロースタートに移行するためその分のバンド幅低下もあるが，この実験では遅延を挿入していないのでその影響は小さい．スイッチやルーターでのバッファ溢れでは，後続パケットが失われることが多いことに注意して欲しい．

3. 実 現 方 法

3.1 通信パラメータの保存復元

まず，通信フェーズよる通信パラメータの不整合を回避するため，通信パラメータの保存復元を行なう．その実現として，集団通信の前後で ssthresh 値の保存と復元を行った．今回の実現では，保存復元しているのは ssthresh 値のみである．ssthresh は輻輳が起こった場合などに，その直前の cwnd の値を保持するパラメータであり，通信の帯域遅延積を良く反映している．また，cwnd は常に変動しているので，cwnd よりも安定した値が保持されている ssthresh を使用した．ssthresh はソケット毎の TCP 構造体中に保持されている．

実装では，通信を一対一通信と集団通信への 2 群に分け，それぞれの群で ssthresh を独立に持つようにした．集団通信の呼び出しの前後で，パラメータの保存復元を行なっている．集団通信の入口では，一対一通信用の ssthresh を保存し集団通信用の ssthresh の復元を行なう．集団通信の出口では，集団通信用の ssthresh を保存し一対一通信用の ssthresh の復元を行なう．ただし，**MPI_Barrier** や **MPI_Bcast** など一部の集団通信はバンド幅を共有しないので一対一通信に含めている．バンド幅を共有する集団通信としては，**MPI_Alltoall**，**MPI_Alltoallv**，**MPI_Reduce** 等がある．

通信を一対一通信と集団通信への 2 群に分けるのは，実際の MPI 通信ライブラリとしては単純過ぎる．ただし，評価に用いるベンチマークでは単純に特定の集団通信を使用するだけなので評価目的にはこれで十分である．

実装では，パラメータ変更時点で TCP をスロースタートの状態に移行させている．TCP をスロースタートの状態に移行させると，次に述べるペーシングが機能し，正確に ssthresh の値に従って通信が開始される．

3.2 通信再開時のペーシング

スロースタートによる性能低下を回避するため，スロースタートを行なわず，代わりにタイマ割り込み

により生成したクロックによるペーシングを用いる．ペーシングとは，ACK クロッキングに関係なく，一定の間隔でパケットを送信する機構である[15]．

ペーシングには，RTT/ssthresh をペーシング周期の目標値としている．バンド幅に置き換えると，(ssthresh∗MSS)/RTT をバンド幅の目標値として使用していることになる（MSS は Maximum Segment Size, Linux では cwnd や ssthresh はバイト数ではなくパケット数で数える）．ペーシングに用いる ssthresh は，通信フェーズに合わせて適切な値になるように保存復元が行なわれていることを仮定している．通常，ssthresh が十分正確な帯域遅延積の値を保持しているので，それを目標値とした．

通信層において，通信がスロースタート状態に入ったことが分かると，ペーシングを起動する．ペーシングは，以下の条件で開始する:

インフライト中のパケットがなく，かつ，
cwnd < ssthresh

このチェックのために TCP 層にフック関数を入れ，パケットを送信する毎に条件を検査している．

ペーシングは ACK を受信した場合に，ただちに停止している．そして全てのペーシングが停止した時点で，タイマの割り込みも停止している．よって，ペーシングの実装は，通信再開時点以外では TCP の挙動に影響を与えない．また，余分なオーバーヘッドもない．

コネクションの作成直後は ssthresh が無限大に設定されているので，その場合，あらかじめ設定されている目標値を使用する．今回はボトルネックが 500Mbps であるので，その半分の 250Mbps を ssthresh の初期値として使用した．このような静的な環境情報は，グリッド環境などではネットワーク情報サービス等で管理されており，そういったサービスから取得できるものと想定している．

今回の実装では，タイマ割り込みに 10µsec 周期のタイマを使用している．タイマ割り込み毎にペーシング周期目標値をチェックし，ペーシング周期毎に cwnd を増加させている．割り込み周期としては，バーストを抑えルーターのバッファ溢れを起こさない程度に設定すれば良い．1Gbps Ethernet では MTU サイズのパケット送信に 12µsec 程度かかるので，10µsec は十分細かい周期と言える．また，10µsec 周期なら 10Gbps になっても 10 パケット単位でペーシングできるので，ルーターのバッファを溢れさせないようにできると考えられる．1Gbps のネットワークだけを考慮するなら 100µsec でも十分であったが，10Gbps での利用の検証の意味で 10µsec という速い周期を選んでいる．

IA32 システムでは，1µsec 以下の精度を持つ高速タイマとして APIC タイマ，HPET が備わっている．今回は，利用が簡単な APIC タイマを利用した．Linux-2.6 には，双方のタイマをサポートするコードが入っているが，今回はそれらを使用せず，割り込みベクタを直接書き換えて実装している．

Linux 上の実装の詳細であるが，割り込みハンドラは cwnd を増加させた後，パケット送信をスケジュールしソフトウェア割り込みを起動している．ソフトウェア割り込みとは，割り込みハンドラ処理の一部であり，時間のかかる処理を行なうために用意された OS の仕組みである．ハードウェア割り込みを許可した状態で動作するようになっている．処理はコールバック関数のキューとしてスケジュールされる．ハードウェアに対する割り込み処理が終った時点でキューのエントリが順次実行される．Linux では，ネットワークの送受信を行なうソフトウェア割り込みにはネットワーク・デバイス（NETDEVICE）が必要である．そのため，疑似デバイスを用意しそれをスケジュールに使っている．

3.3 RTO 時間の短縮

RTO 時間による無通信時間の発生を軽減するためには，RTO 時間の短縮を行う．RTO 時間は RTT 時間を元に RFC[13] の記述に従って計算されている:

$$\mathrm{RTO} = \mathrm{SRTT} + 4 * \mathrm{RTTVAR}$$

ここで，SRTT は smoothed round-trip time，RTTVAR は round-trip time variation であり別途計測されているものとする．

RFC では，RTO の最小値を 1sec と定義しており，1sec 未満の場合は 1sec に切り上げるべき（SHOULD be）と定義されている．Linux の TCP では，少し異なり，4∗RTTVAR の最小値が 200msec になっている．そのため，Linux では RTO の値はほぼ RTT + 200msec になっている．RTO 時間の短縮では，RTTVAR の値として（最小値の制約を外し）本来の観測された値を使用するように変更した．

ここで，RFC では，RTO の最小値を大きく取っている理由として，古い TCP では遅延 ACK 処理などに精度の悪いタイマを使っているため，RTO が小さいと無駄に再送が発生する点を挙げている．しかし，遅延 ACK などによる無駄な再送は数パケットである．Gbps クラスの高バンド幅環境では大きな問題にならないと考えられる．

4. 評価実験

4.1 実験環境

実験環境はすでに 2 節に示した．

ここでは，RTO 時間短縮についての評価以外は，人工的に遅延を入れた WAN 環境で評価を行なうこととする．遅延が小さい場合には，ACK の到着が早く cwnd が速やかに大きくなる．そのため，不適切な cwnd の値で通信を再開しても影響は小さく，性能上無視できるためである．RTO 時間短縮については，タイムアウト時間自体が影響するので遅延のない環境で評価を行なう．ただし，RTO 時間短縮についても，スロースタートが影響するため遅延がある方が影響は大きく

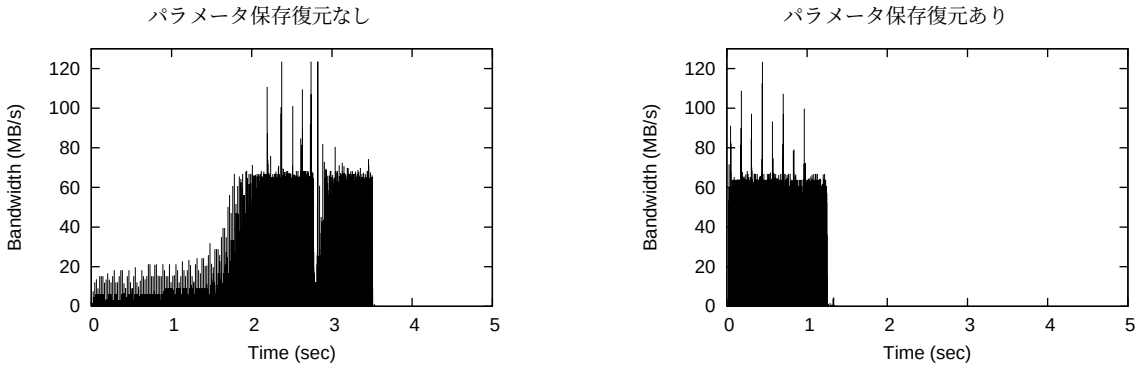


図 4 通信パラメータ保存復元の効果

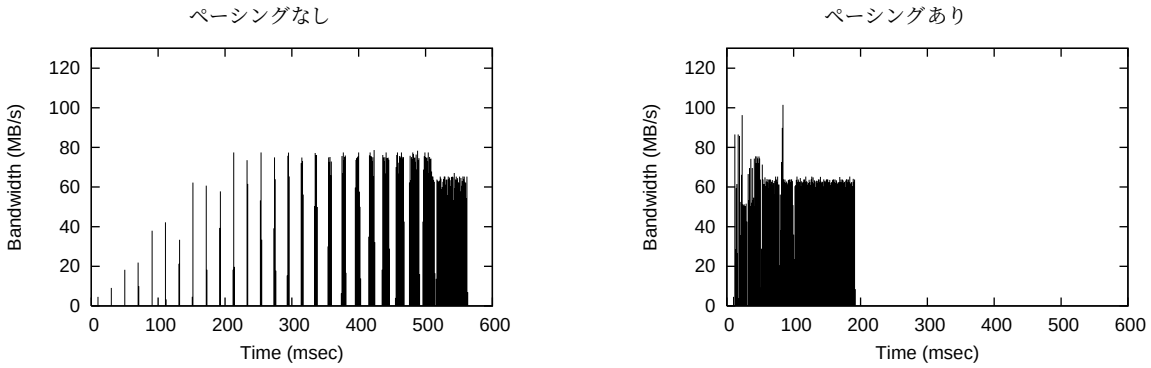


図 5 通信再開時のペーシングの効果

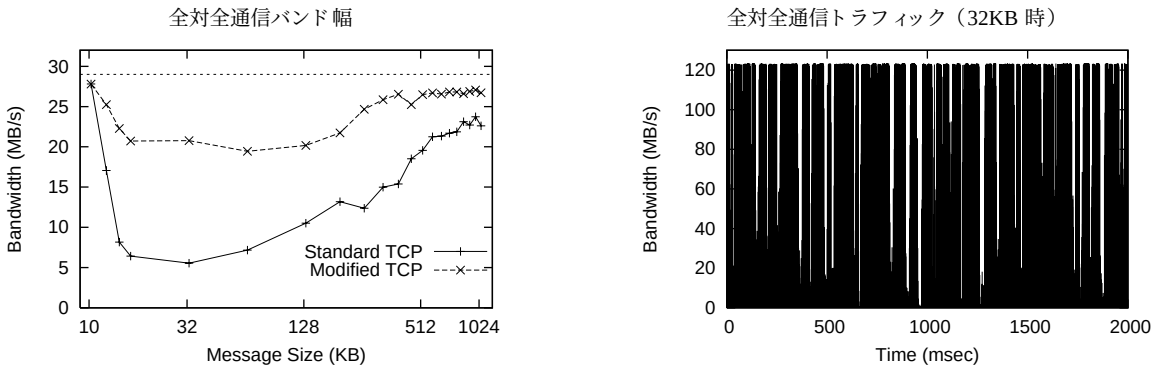


図 6 RTO 時間短縮の効果

なる．

実験では，ソケットに対して NODELAY オプションを指定している．NODELAY は，Nagle 処理を無効化するものである．Nagle 処理は小さいパケットの送信時に複数パケットを一つにまとめる処理を行なう．また，ソケットバッファは 10MB に設定している（Linux ではバッファの設定値として倍の 20MB が取られる）．これは帯域遅延積に対して十分なバッファ量である．

実験では，RTO 時間の短縮は，全てのコネクションで有効とした．これは，遅延の小さいクラスタ内の通信でも RTO は発生するためである．また，通信再開時ペーシングもクラスタ内通信でも有効にしている．ただし，遅延の小さいクラスタ内では ACK は短時間（数 10μsec）で返って来るため，ペーシングの効果はほとんどないと考えられる．

4.2 通信パラメータの保存復元

図 4 に通信パラメータ保存復元の効果を示す．グラフは，集団通信を行なった後に続けて一対一通信を行なった場合のトラフィックを表示している．左のグラフは，集団通信を行なった後，そのまま一対一通信を開始した場合である．右のグラフは，パラメータを復元してから一対一通信を開始した場合である．復元に使うパラメータは，先に一対一通信を行ない，その時に保存したものを使用している．グラフから，パラ

メータの復元によりトラフィックが定常状態に近い状態でスタートしていることが分かる．一方，パラメータを復元しない場合，輻輳回避フェーズにより緩やかにバンド幅が大きくなっていくのが観測される．

一対一通信の通信データ量は，50MB である．また，通信再開時のペーシングと RTO 時間の短縮も有効にしている．この実験では，パラメータを復元しない場合，直前の集団通信により小さくなっている ssthresh=49 という値で通信がスタートしているのが観測された．一方，復元する場合，一対一通信により大きくなっている ssthresh=582 という値で通信がスタートしているのが観測された．

ここでの集団通信には，**MPI_Alltoall** のうちボトルネックを通過する通信のみを行なうように変更したものを使用した．これは，**MPI_Alltoall** ではタイミングにより結果が大きく異なり再現性がなくなるためである．**MPI_Alltoall** はボトルネックを通過しない通信を含むため，ボトルネックを通過するトラフィック量が時間とともに大きく変動する．

4.3 通信再開時のペーシング

図 5 に通信再開時のペーシングの効果を示す．実験は，既出図 2 と同一の実験であり，10MB のデータ転送を 2 秒間隔で繰り返す通信を行なっている．グラフは，2 秒間隔で繰り返したうちの 1 回分を取り出したものである．グラフの X 軸は時刻（msec），Y 軸は 1msec あたりのデータ転送速度を示す．

図 5 左のグラフ（ペーシングなし）は既出図 2 と同一である．図 5 右のグラフは，ペーシングを行なっている場合であり，通信がすみやかに開始されていることが分かる．

標準 TCP のスロースタートでは，RTT 時間毎に急速にバンド幅が上がるが，Gbps クラスのバンド幅に達するには 10MB 程度のデータ量が必要であることが分かる．スロースタート以降は十分なバンド幅が出るので，データ量が十分大きければ性能上の問題は相対的に小さくなる．しかし，10MB 程度の通信量では，スロースタートの性能への影響は大きい．スロースタート期間の長さは遅延 D に対して D log(D) に比例するので，遅延が大きくなるに従って性能への影響も大きくなる．

4.4 RTO 時間の短縮

図 6 左のグラフに **MPI_Alltoall** をメッセージサイズを変えて呼び出した時のバンド幅の変化を示す．設定は図 3 の時と同一である．ラベル「Modified TCP」が RTO 時間の短縮を行なったものであり，性能が大きく改善されていることが分かる．

この実験環境で理想的な通信が行なわれた場合，スイッチを経由する通信速度は 1Gbps，ボトルネックを経由する通信速度は 1/8Gbps になる．この構成では，理想的な通信速度は約 29 MB/s になる．グラフ中に破線としてこの値を示した．

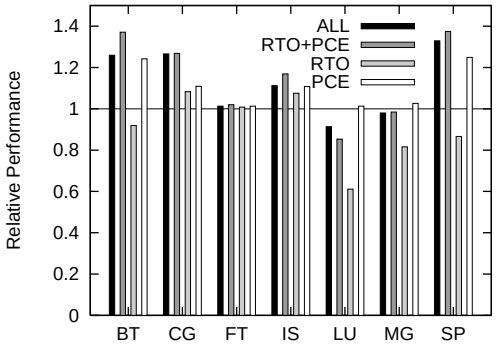


図 7 NPB ベンチマーク性能比較

表 2 NPB ベンチマーク絶対性能

	STD	ALL	RTO+PCE	RTO	PCE
BT	2835.48	3571.02	3887.74	2606.81	3520.48
CG	293.86	371.99	372.48	318.37	325.94
EP	82.23	82.23	82.22	82.23	82.22
FT	1168.01	1182.69	1191.58	1177.80	1182.74
IS	23.61	26.25	27.60	25.39	26.15
LU	4143.90	3781.36	3535.71	2531.91	4198.00
MG	1130.55	1107.41	1112.02	922.44	1160.35
SP	995.72	1323.11	1368.42	862.46	1243.75

図 6 右のグラフは，データサイズ 32KB の場合のトラフィックである．図 3 右のグラフと比較することで効果は明らかである．

ここでの **MPI_Alltoall** 通信のアルゴリズムには，全ての **MPI_Irecv** と **MPI_Isend** を発行し，続いて **MPI_Waitall** を発行する単純ものを使用している．

4.5 NPB ベンチマーク性能比較

次に，アプリケーション・ベンチマークを使って評価する．ベースになる TCP 実装の輻輳回避部分は一切変更していないので，定常通信を行なうベンチマークでは挙動の違いを示すことができない．そこで，実際に断続的な通信が起こるアプリケーション・ベンチマークを使って評価を行なうことにする．

ベンチマークには，NPB ベンチマーク（NPB2.3）を使用した．問題サイズは CLASS=B，プロセス数は NPROCS=16 で行なった．測定は，標準 TCP の場合に 10%程度遅くなる揺らぎがみられるので，それぞれ 3 回ずつ計測しその最大値を取っている．

図 7 に標準 TCP を 1 にした時の相対性能を示す．**表 2** にベンチマークの絶対値を示した．STD は標準 TCP である．RTO は RTO 時間の短縮，PCE は通信再開時ペーシングである．RTO+PCE は 2 つの組合せ，ALL は通信パラメータ保存復元を含めた 3 つの変更を全て組み合わせたものである．

BT，CG，IS，SP で 10%から 30%の性能向上がみられた．FT と MG では効果はほとんどなく，LU では逆に性能が低下している．

詳細にトラフィックを観測すると，BT ベンチマークでは，200msec を超える計算フェーズがありスロースタートが発生している．そのため効果があったと推測できる．

CG ベンチマークでは，短い周期で断続通信を行なっている．いくつかの点で通信バーストとスロースタートが発生しており変更に効果があったと推測できる．

FT ベンチマークでは，連続的に通信が起こっており性能差が出なかったと推測できる．

IS ベンチマークでは，通信フェーズ開始時点でスロースタートしており変更に効果があったと推測できる．

LU ベンチマークでは，ごく小量の通信が持続している．RTO もパケット・ロスも発生しないので，標準 TCP で問題なく通信が行なわれている．今回の変更により，RTO 時間を短縮したことで高い頻度でスロースタートが発生し，ペーシングしていても有効なバンド幅が減り性能が低下したと推測できる．

MG ベンチマークでは，断続的に通信を行なっておりバーストも観察できるが，CG に比べて小さい．標準 TCP でもパケット・ロスは起こっていないと考えられ，性能差が出なかったと推測できる．

SP ベンチマークでは，まれにパケット・ロスを起こしスロースタートの状態で通信を行なっていると考えられ，変更により性能向上したと推測できる．

RTO 時間の短縮，通信再開時ペーシング，通信パラメータ保存復元の変更の間には依存関係があるのに加えて，トラフィックの性質によっては性能を悪化させる場合もある．例えば，RTO 時間短縮はスロースタートの頻度を高くするため性能を悪化させる原因になり得る．通信パラメータ保存復元についても，保存復元したパラメータがより適している保証はない．実際，RTO 時間短縮を単独で使用しペーシングを行なわない場合，性能が低下した．また，通信パラメータ保存復元には効果がみられず，逆に多くの場合性能が低下した．

4.6 タイマ割り込みの影響

10μsec のタイマ割り込みのオーバーヘッドの影響を見る．10μsec のタイマ割り込みをかけると，空ループを実行するプログラムの実行速度が 1.8%から 2.0%程度低下する．ペーシングのために割り込みを使用している期間も短いので，この程度のオーバーヘッドは許容範囲だと言える．

5. 議　　論

5.1 TCP 公平性

TCP の研究では複数ストリーム間の公平性の議論が必要である．本論文の TCP の変更は通信再開時（RTT 時間内の現象）に対するものであり，公平性に影響する輻輳回避フェーズについては全く無変更である．つまり，公平性についてはベースになる TCP 実装に従っている．また，通信再開時に起こる通信バーストは他のストリームに悪影響を与えるが，ペーシングにより通信バーストを抑制しているので，その点でも公平性に与える悪影響は小さくなっている．

5.2 高速タイマ割り込み

今回の実装では，APIC タイマという IA32 CPU に内蔵されるタイマを使用した．APIC タイマは CPU に内蔵されているため，SMP 計算機では割り込みの負荷分散が行なわれない．IA32 では，他にも HPET（High Precision Event Timer）が用意されている．HPET は I/O ハブに内蔵されており，マルチメディア等のソフト実時間処理用に設けられたものである．SMP 計算機におけるタイマ割り込みの負荷分散を考えると，I/O 処理である TCP のペーシングには HPET の方が適していると考えられる．

5.3 Linux の TCP スタックの問題

図 5 右のグラフ（ペーシングあり）において，時刻 0msec 付近にトラフィックの乱れがある．これはペーシング処理がうまく動作しなかったためである．通信初期時には，ユーザーデータのソケット・バッファへのコピーが行なわれる．Linux の実装では，ソケット・バッファへのコピー中，数 msec に渡ってソケット構造体が排他的にロックされる．そのため，割り込みによりペーシング処理が呼び出されても，ロック期間中は処理が行なえなくなる．実際，実験では割り込み回数の半分程度でソケットがロックされているのが観測された．

図 5 右のグラフで RTT 時間（20msec）以降にもトラフィックの乱れが見られるが，RTT 時間以降はペーシングは機能していないのでこれは標準 TCP の挙動である．

6. 関 連 研 究

無通信期間後の通信開始（スタートアップ期間）の挙動については，WEB トラフィックへの対応として数多くの提案がなされている[5]．

論文 2) では，RTO 時間と，通信再開時のペーシングについて議論している．RTO 時間について，古い BSD Unix 実装で使用されている低精度タイマ（200msec や 500msec）の代わりに，比較的高精度のタイマ（10msec）を使用することで性能が向上することを示している．これは RTO を実質短縮することになっている．また，ペーシングについては，バンド幅 128Mbps，遅延 5msec の環境で一般的な OS のタイマ間隔である 10msec でペーシングしたシミュレーション結果が良好であることを示している．

多くの研究では，ペーシング用のタイマとして，1msec や 10msec のタイマを使用している．1Mbps 程度までのトラフィックであれば，10msec のタイマでも

ペーシング可能であるが，1Gbps 以上では難しいと考えられる．一方，IA32 システムでは，高速タイマとして APIC タイマ，HPET が備わっている．論文 11) では，APIC タイマとそれを利用した UDP パケットの μsec 単位でのペーシングの実験結果が述べられている．論文 8) では，高精度タイマを使用し，通信再開時のバーストを均一に分割するペーシング手法が提案され，日米間の高速ネットワークを使用した実験結果が述べられている．

また，論文 2)，12)，15) では，ペーシングの目標値の決定方法として，より正確な RTT の計測値を用いる方法や，ssthresh が時間の経過によって不正確になっていることを想定し一定のファクターをかける方法なども提案されている．

Ensemble-TCP[3] では，通信相手が同じ複数のストリーム間では挙動が似ているという点に注目している．その仮定のもと，TCP のパラメータをストリーム間で共有することによって性能を向上させている．一方，本論文の TCP の変更は，単一ストリームであっても計算フェーズによってストリームの利用性質が異なる点に注目している．そして，フェーズにより違うパラメータ・セットを利用することで性能向上を目指している．これはある意味，逆のアプローチになっている．複数ストリームの利用を前提にすると，計算フェーズ毎に個別のストリームを割り当てることでパラメータ保存復元の効果を得る実装も考えられる．

7. おわりに

TCP の通信断続部分の処理に対して簡単な変更を行なうことにより，MPI 通信負荷での性能向上が可能であることを示した．変更を加えたのは通信断続部分のみであり，定常通信が行なわれる輻輳回避フェーズ等は無変更である．通信断続部分の変更だけでも，いくつかのベンチマークに対して 10%から 30%の性能向上が得られた．

変更を加えた点については，それぞれ以下の状況に対して効果が期待される．RTO 時間の短縮は，輻輳時のリカバリー時間の短縮．通信再開時ペーシングは，通信再開時のスロースタートによるバンド幅抑制の回避．通信パラメータ保存復元は，トラフィックの性質の変化によるバンド幅抑制の回避．

他の TCP のトラフィックへの影響という点では，バースト抑止が重要であると考えられる．今回の変更は，バーストを抑止する方向に変更しているのでその点でも問題はないと考えられる．NPB ベンチマークの性能向上に効果はなかったが，通信パラメータの保存復元もバースト抑止に効果があると考えられる．

NPB ベンチマークのトラフィックをサンプルした結果，集団通信を行なう FT と IS を除くベンチマークでは利用バンド幅はかなり小さいことが観測された．広域ネットワークでの MPI の利用を考えた場合，標準 TCP がバースト的な通信を行なう点を考慮すると，バンド幅と同様にルーターの持つバッファ量が重要なファクターになってくると考えられる．

大規模クラスタ同士を接続し数百のコネクションが存在する状況で TCP がうまく動作するかは課題である．今回の変更は定常状態での通信には変更を行なっておらず，また，バースト抑制の効果がある．そのため，標準 TCP に比較して性能が悪化することはないと予想されるが，実際の挙動の解析は今後の課題である．

謝　辞

なお，本研究の一部は文部科学省「経済活性化のための重点技術開発プロジェクト」の一環として実施している超高速コンピュータ網形成プロジェクト（NAREGI：National Research Grid Initiative）による．

参 考 文 献

1) M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, 1999.
2) M. Aron and P. Durschel. TCP: Improving Startup Dynamics by Adaptive Timers and Congestion Control. Tech. Rep. TR98-318, Rice Univ., 1998.
3) L. Eggert, J. Heidemann, and J. Touch. Effects of Ensemble-TCP. ACM Computer Communication Review, 30(1), 2000.
4) S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, 1999.
5) A. S. Hughes, J. Touch, J. Heidemann. Issues in TCP Slow-Start Restart After Idle. Expired Internet Draft, draft-hughes-restart-00.txt, 2001.
6) 石川. YAMPII もう一つの MPI 実装. 情報処理学会, SWoPP'04, 2004.
7) V. Jacobson. Congestion Avoidance and Control. Proc. SIGCOMM'88, in ACM Computer Communication Review, 18(4), 1988.
8) H. Kamezawa, M. Nakamura, J. Tamatsukuri, N. Aoshima, M. Inaba, K. Hiraki, J. Shitami, A. Jinzaki, R. Kurusu, M. Sakamoto, and Y. Ikuta. Inter-layer Coordination for Parallel TCP Streams on Long Fat Pipe Networks. SC2004, 2004.
9) Y.Kodama, T.Kudoh, R.Takano, H.Sato, O.Tatebe, and S. Sekiguchi. GNET-1: Gigabit Ethernet Network Testbed. IEEE Intl. Conf. on Cluster Computing (Cluster2004), 2004.
10) M. Matsuda, Y. Ishikawa, and T. Kudoh. Evaluation of MPI Implementations on Grid-connected Clusters using an Emulated WAN Environment. CCGrid2003, 2003.
11) V. Oberle and U. Walter. Micro-second Precision Timer Support for the Linux Kernel. IBM Linux

Challenge, 2001.
http://www.tm.uka.de/itm/publications.php?id=61
12) V. N. Padmanabhan and R. H. Katz. TCP Fast Start: A Technique for Speeding Up Web Transfers. Proc. IEEE GLOBECOM Internet Mini-Conference, 1998.
13) V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988, 2000.
14) 高野, 工藤, 児玉, 松田, 手塚, 石川. 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.
15) V. Visweswaraiah and J. Heidemann. Improving Restart of Idle TCP Connections. Tech. Rep. 97-661, Univ. of Southern California, 1997.
16) L. Xu, K. Harfoush, and I. Rhee. Binary Increase Congestion Control for Fast Long-Distance Networks. INFOCOM 2004, 2004.