2004－OS－97　(9)
2004／8／1

GridMPIのためのTCP/IP輻輳制御実装方式の検討

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

我々は，遅延がありクラスタ内に比べて帯域が小さいIP網を介して接続された計算機群上で，MPIアプリケーションを効率的に実行するGridMPIの開発を行っている．ところがこのような環境では，特に集合通信の性能が，帯域による制限以上に大幅に低下することがある．本稿では，特に全対全通信時の挙動について詳細に解析し，TCP通信の再送タイムアウトにより性能の低下がおきることを明らかにする．また，この問題を解決する手法について検討を行う．

A Consideration of TCP/IP Congestion Control Mechanisms for the GridMPI

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

We are developing the GridMPI, which is an MPI implementation for Grid environment, in which clusters are distributed in a wide area and connected by an IP network. When MPI collective communication is performed on such environment, communication performance is sometimes significantly low. In this report, we precisely analyze all-to-all collective communication, and make clear that the performance degradation is caused by Retransmission Time Out (RTO). We also discuss some ideas to avoid such performance degradation.

1. はじめに

クラスタ，グリッド技術が注目され，LANやインターネット上での並列処理のニーズが高まっている．我々は，グリッド環境上での並列処理を実現するために，広域に分散して配置されたクラスタ計算機群をつないだグリッド環境で，Message Passing Interface(MPI)[1]を用いて記述された並列アプリケーションを効率よく実行するGridMPI[3),4)]の開発を行っている．

GridMPIは，数ms程度以下の通信遅延で十分な通信バンド幅を有するキャンパスエリアやメトロポリタンエリアネットワーク上の1000台規模のクラスタや並列コンピュータ群から構成されるグリッド環境を想定している．このような環境における相互運用性を保つため，TCP/IPプロトコル標準を利用しつつ通信性能を改良していくアプローチをとっている．

我々は，並列アプリケーションにおける通信遅延の影響を解析した論文[5)]において，既存MPI通信ライブラリは通信遅延を考慮した実装になっていないために，クラスタ環境とは異なり通信遅延が大きなインターネット環境ではネットワークの物理性能を引き出せていないことを示した．また，論文[7)]では，遅延のあるネットワーク環境でホストOS内部のインタフェースキューあふれを輻輳とみなすことにより性能が低下していることを明らかにし，双方向通信時においてACKを優先的に返すことの重要性を示した．

一方，帯域遅延積が大きなネットワークにおけるTCP輻輳制御方式としては，HighSpeed TCP[9)]，Scalable TCP[8)]，FAST[10)]などが提案，実装されている．これらの方式は定常状態の通信が継続する場合には，十分な性能を得られることが知られている．しかし，MPIを用いて記述された並列アプリケーションのように，周期的に通信相手を切替えながら多対多通信を行う場合の振舞いについては今まで十分に研究されていない．

GridMPIが想定する環境では多数の計算ノードが集まったクラスタ等が広域網を介して相互に接続される．個々のノードはギガビットクラスの通信リンクを持つ．これに対して，広域網はギガビットから１０ギガビット程度のバンド幅であり，多数のノードが一斉に広域通信を行うと広域網が隘路（ボトルネック）となる．

本稿では，MPIおよびTCPプログラムによる全対全通信において，ノード数が増えた場合にバースト的なパケットロスによって再送タイムアウトが発生し，

† 産業技術総合研究所グリッド研究センター (National Institute of Advanced Industrial Science and Techinology)
†† 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 (University of Tokyo)

表 1　MPI 集合通信 API

通信パターン	関数名	説明
バリア同期	MPI_Barrier	全プロセスの実行同期
全対全通信	MPI_Alltoall	全プロセスのデータ交換
ブロードキャスト	MPI_Broadcast	ブロードキャスト
1 対全通信	MPI_Scatter	データ配布
全対 1 通信	MPI_Gather	データ集め
リダクション	MPI_Reduce	縮約

性能が低下していることを示す．この性能の低下は，ボトルネックリンクが存在する場合に顕著である．この問題の解決策として，再送タイムアウト時間の変更と，IFG 変更による疑似ペーシングの適用について議論する．

以下，2 節において MPI 通信ライブラリの集合通信と TCP による実装方法について述べる．続いて，3 節で MPI と TCP による全対全通信の性能測定について述べ，4 節で問題点に対する仮説を述べ，5 節でその仮説に対する解析を行う．6 節で問題点の解決策について考察を行う．最後に 7 節でまとめを行う．

2. MPI 集合通信と評価プログラム

2.1 MPI と集合通信

流体力学，分子動力学などに代表される科学技術分野の並列アプリケーションでは，問題領域を分割し，各コンピュータは分割された一部分を計算し，その結果を他のコンピュータに送信する，という手順を繰り返す．すなわち，周期的に 1 対 1，全対全，全対 1，1 対全，などの通信でかつ一度に大量のデータが授受される．MPI では，1 対 1 を除くこのような通信パターンを集合通信と呼び，**表 1** に示すような専用 API を用意している．

2.2 全対全通信プログラム

集合通信時の性能と通信挙動を解析するために，もっとも通信帯域を必要とする全対全通信を取り上げる．

4 ノード通信時を例に，全対全通信の通信パターンを**図 1** に示す．図 1(a) のように，ノード 0 が持つデータ配列 {A, B, C, D} は全対全通信完了後は {A, E, I, M} となる．また，ノード間では図 1(b) に示す通りデータ送信が相互ノードで重ならないように通信し合い，3 フェーズで完了する．以下，1 フェーズで 1 ノードに転送するデータサイズをメッセージサイズと呼ぶ．

送受信に費す時間はノード毎にばらつきがあるので，フェーズの移行時刻はノード間でばらつきがあり，複数ノードから一つのノードへの通信が重なる場合が存在し得る．バリア同期によってフェーズからフェーズの移行時に同期する実装は可能であるが，グリッド環境では通信遅延が大きいため，オーバヘッドが大きく現実的でない．本稿で用いた評価プログラムはすべてフェーズのバリア同期を行っていない．

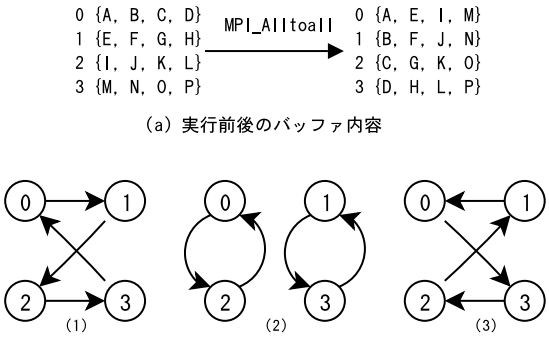

図 1　全対全通信パターン

また，MPI 通信ライブラリを用いた通信は，MPI におけるバッファ管理などが行われ，挙動が複雑になるため，全対全通信のより基本的な挙動を解析するために，`MPI_Alltoall` 関数を用いた MPI 版の他に同等の機能を TCP/IP ソケット関数を使って実装した TCP 版を作成した．

TCP 版はシングルスレッドプログラムであり，read，write 関数はノンブロッキングで実行し，select 関数によって送受信イベントを待つ．また，送信は図 1(b) のように隣り合ったノードから順に逐次的に write 関数を呼出し，受信はすべてのノードからの通信を select 関数によって待ち受け，read 関数を呼出す．

いずれのプログラムも，全ノードとの送受信が完了する (即ち図 1(b) の全フェーズを実行する) までを 1 イテレーションとし，これを 2 ノード間のデータ交換が累積 100MB に達するまで繰り返したときの全通信の平均帯域の和を測定した．

3. 実　　験

3.1 実験環境

実験環境のクラスタ構成を**図 2** に示す．グリッド環境に近い実験環境を作るため，8 台構成の PC クラスタを二組用意し，クラスタ内はそれぞれ単一のノンブロッキングスイッチで接続した．スイッチ間は，ギガビットの帯域と 2ms の往復通信遅延 (以下，通信遅延は双方向に均等に遅延がある往復通信遅延を指す) を持つネットワークをエミュレートするために，ハードウェアネットワークエミュレータ GNET-1[6] を介して接続した．GNET-1 はハードウェアロジックにより正確で細粒度な通信遅延をエミュレートでき，ワイヤレートで 100ns 単位に最大 268ms の往復通信遅延を設定することが可能である．さらに，通信挙動に影響をあたえることなく，ワイヤレートでパケットをキャプチャする機能も持つ．

Catalyst 3750G-24T の入力キューサイズは 75 パケット，出力キューサイズは 40 パケットであり，パ

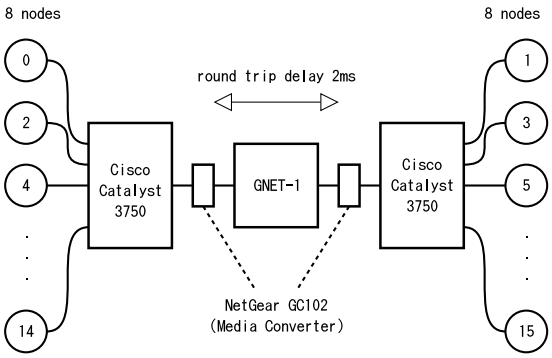

図 2 実験ネットワーク構成

表 2 ホストのハードウェア仕様

Processor	Pentium4 2.4GHz
Main Memory	FSB800 512MB
Mother Board	Intel D865GCL
NIC	Intel PRO/1000 (82547EI)
Bus Interface	CSA (Bus Bandwidth 2Gbps)

表 3 OS 設定

OS	Fedora Core 2
Kernel	Linux 2.6.5-1.358 + Web100 2.3.8
Intel PRO/1000 Driver	5.2.39-k2
TCP Protocol	default(NewReno + SACK)
Socket Buffer	2MB

ケットは 12MB の共有メモリに格納される．

各ホストのハードウェアの諸元を**表 2** に，ソフトウェア構成を**表 3** に示す．

カーネルは Linux 2.6.5 をベースに Web100[12] パッチを適用した．Web100 は高性能ネットワーク環境における TCP バッファチューニングと性能解析ツールの提供を目的としたプロジェクトである．Web100 カーネルパッチはプロトコルスタックにフックを挿入することで，TCP コネクション単位での輻輳ウィンドウ，スロースタート閾値，広告ウィンドウなどの統計情報を取得し，proc ファイルシステムを介してユーザアプリケーションに提供する．

また，デフォルトカーネルの挙動と一致させるために，Web100 バッファオートチューニングを無効とし，ソケットバッファサイズを 2MB と設定した．これは通信遅延 2ms で性能を出すために十分な値である．

なお，MPI 通信ライブラリには YAMPII[2] を使用した．

3.2 全対全通信の性能

ノード数を 2，4，8，16，メッセージサイズを 1KB から 10MB まで変えながら解析用プログラムを実行した．MPI 版の結果を**図 3** に，TCP 版の結果を**図 4** に示す．

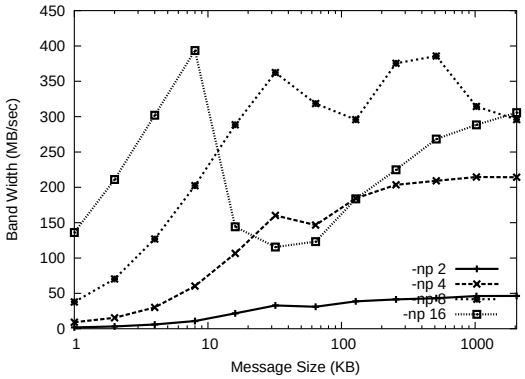

図 3 全対全通信 (MPI 版)

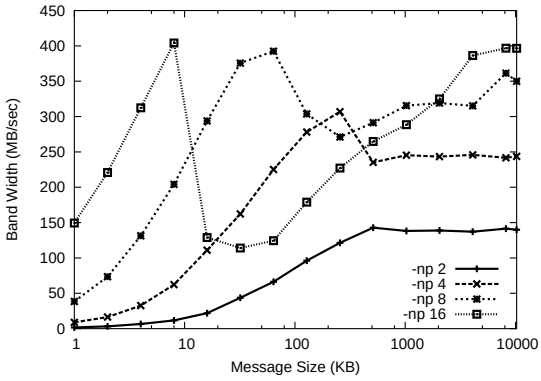

図 4 全対全通信 (TCP 版)

どのノード数でもメッセージサイズを大きくするにつれ，あるピーク値まで使用帯域が上昇した後，低下し，再び上昇する．低下の割合はノード数が多いほど大きく，ノード数が多い方が性能が低い場合があるという逆転現象が起きている．

MPI 版，TCP 版共にほぼ同じ傾向が見られるので，以下の解析は挙動が単純な TCP 版で行う．なお，GNET-1 によって遅延を付加しない環境でも，同様の現象が確認できた．

4. 仮　　説

ノード数が増えた場合に，上記のような性能劣化が発生する原因として，ボトルネックリンクの手前側のスイッチにおいてパケットロスが発生していると考えられる．パケットロスの原因としては，次の 2 点が考えられる．

- 宛先の重複
  1 イテレーションの全対全通信は，図 1(b) のような複数フェーズに分割できるが，各フェーズ毎に同期はしないので，ノード毎にフェーズ移行のタイミングのずれが生じる．このタイミングのずれがあると，一つのノードに対し同時に複数のノー

ドからの通信が行われる可能性がある．このような現象が発生する確率は，ノード数が増えるにしたがい高くなる．この際，重なった通信の帯域の合計が，その宛先ノードがつながっているリンクの帯域を超えると，スイッチにおいてパケットロスが発生する．

- バースト転送の発生
  ボトルネックリンクが存在する場合，バースト転送がパケットロスの要因になる．TCP は ACK によるセルフクロッキングによって，ボトルネックリンクに合わせてパケット送信間隔を調整し，バースト性の軽減，レート制御を行っている．
  しかし，メッセージサイズが小さい場合は，セルフクロッキングによる定常状態にまで至らないので，メッセージ送信時には常にバースト転送が起きる．

5. 解　　析

本節では，前節でノード増加時の性能劣化の原因と考えた (1) 宛先の重複，(2) バースト転送の発生について解析し，仮説が正しいこと示す．

5.1 ノンブロッキング構成の場合

ボトルネックリンクに対するバースト転送の影響を排除し，宛先が重複した場合の挙動を調べるために，16 ノードすべてを単一のスイッチに接続し，ノンブロッキング構成による全対全通信を実行した．

ノンブロッキング構成の場合，宛先の重複がなければ，転送のバースト性にかかわらずスイッチがパケットロスを起こすことはない．したがって，ノンブロッキング構成でパケットロスが発生すれば，宛先の重複が発生していることを意味する．

測定結果を**図 5** に示す．ボトルネックリンクがないので，図 4 に比べて性能は大きく向上しているが，やはり性能の落ちこみが見られる．また，スイッチの統計情報から，メッセージサイズ × ノード数が 1MB を超えると，パケットロスが発生していることがわかった．これよりフェーズ間で同期がずれ，宛先が重複していることがわかる．また，ノード数が多いと，より小さなメッセージサイズでパケットロスが発生する．これは宛先の重複が発生する確率が高くなるためと考えられる．

5.2 ボトルネックリンクが存在する場合

5.2.1 半対半通信

全対全通信からクラスタ内の通信を省き，ボトルネックリンクを通るデータ授受だけを行った際の性能を測定した．以下，これを半対半通信と呼ぶ．ボトルネックリンクが 1Gbps なので，全通信帯域のピーク値は両方向合わせて 250MB/s になる．

測定結果を**図 6** に示す．メッセージサイズが 16KB の場合，もっとも性能が低下しており，帯域のピーク

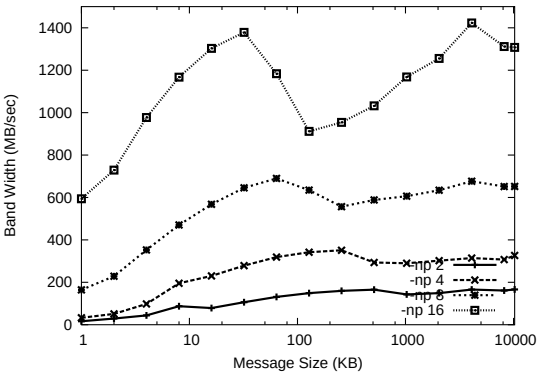

図 5　全対全通信 (ノンブロッキング構成)

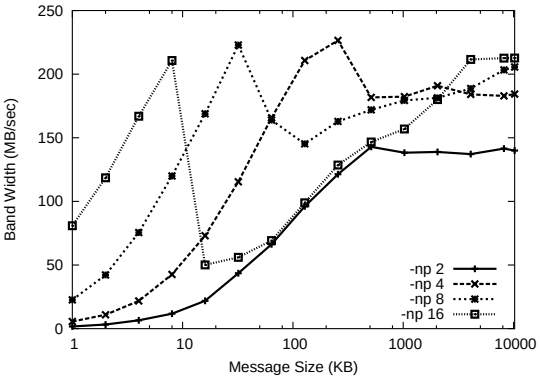

図 6　半対半通信 (ボトルネックリンク 1Gbps)

値の 1/5 である．以下，ボトルネックリンクを介した半対半通信で，メッセージサイズが 16KB の場合についての解析結果を述べる．

5.2.2 バースト転送の発生

送信帯域と輻輳ウィンドウを Web100 にて調べた結果を**図 7** に示す．送信の立ち上がり毎にバースト転送によるパケットロスが発生し，輻輳ウィンドウが減少している．これは単一ノードでの測定結果であるが，すべてのノードにおいてほぼ同じタイミングで送信と中断が発生している．

図 7 に加え，GNET-1 を用いて 1ms の解像度で帯域の変動を測定したところ，約 10 イテレーションごとに，送受信がまったく行われていない時間帯が約 200ms あることがわかった．

5.3 パケットレベルの挙動

前節において，宛先の重複やバースト転送のためにパケットロスが発生していることを示した．本節では，パケットロスによりどのような基準で性能が低下しているかを調べる．

まず，任意の二つのノード (以下，ノード A，B と呼ぶ) とスイッチの間に GNET-1 を挟み，これらのノードが授受するパケットのヘッダをキャプチャすることで，パケット単位の挙動を解析した．

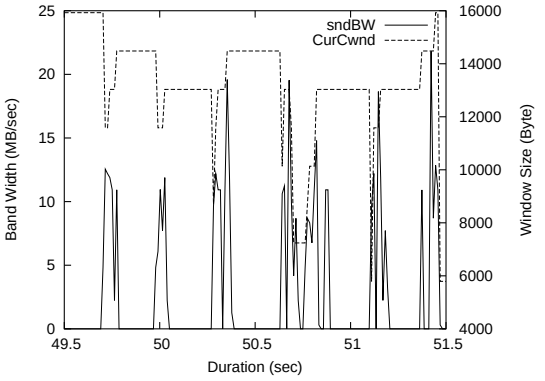

図 7 バースト転送の発生

その結果，約 200ms の通信がまったく行われない期間の後，通信断の前に送ったパケットの再送が行われている場合が見られた．通信断後，他のノードからパケットを受け取ることなく再送を行っていることから，再送タイムアウト (RTO) が発生していることがわかる．Web100 によって各コネクションの RTO 時間を調べたところ，約 200ms であり，この中断時間と一致する．

RTO は，送信側がパケットを送った後，タイムアウト時間たっても受信側から ACK や再送要求を受け取らなかったときに，送信側が再度パケットの送信を行うものである．TCP では，パケットがロスした場合，受信側が後続のパケットを受け取った時点でパケットロスの発生を知り，重複 ACK や SACK(Selective ACK) により再送を促す．したがって，通常のパケットロスでは RTO がおきることはほとんど無い．

キャプチャしたパケットヘッダを解析した結果，次の条件が満たされているために RTO が発生していることがわかった．

- メッセージの末尾のデータパケット，もしくはその ACK パケットが欠損する．
- 上記の現象が，依存性のループ (例えば，図 1(B) のフェーズ 1，3 では 0 から 4 の全ノード，フェーズ 2 では 0 と 2，または 1 と 3 がループになっている) に属するすべてのノードで発生する．

全対全および半対半通信では，全ノードからのデータ受信が完了するのを待って，次のイテレーションに進む．ところが，上記の条件が満たされると，ループ中の全ノードが次のイテレーションに進むことができない．したがってロスしたパケットに後続するパケットを送信しないため，受信側はパケットロスが発生したことを知ることができず，SACK や重複 ACK が送信側に返されないため，RTO がおきるまで通信は再開されない．

図 8，**図 9** に，RTO が発生する場合と，SACK により高速再送される場合のキャプチャデータに基づくシーケンス図を示す．ここでは単純化のため，片方向

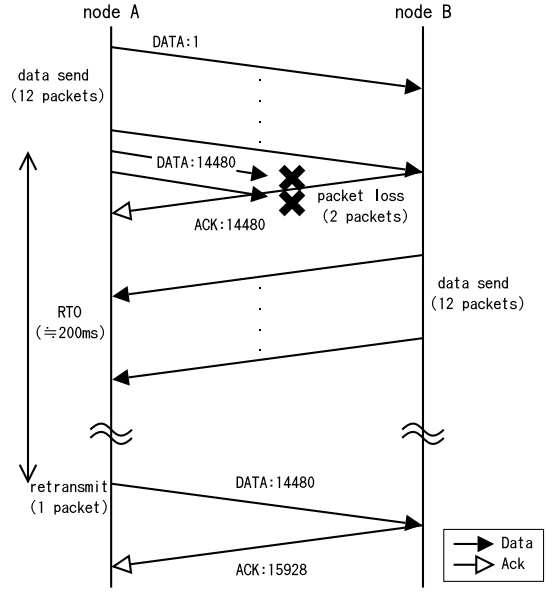

図 8 再送タイムアウト (RTO) の発生

の RTO についてだけ述べる．図中，DATA:および ACK:に続く数字はそれぞれシーケンス番号，ACK 番号を示す．さらに，シーケンス番号は 1 から始まるように実データを加工した．

図 8 は RTO が起きる場合である．ノード A からノード B 宛のメッセージは，シーケンス番号 1 から始まる 12 個のパケットにフラグメンテーションされる．そして，メッセージの末尾から二つのパケットがスイッチで欠損している．ノード B は受信分までの ACK として，ACK:14480 をノード A に返信する．

続いて，ノード B からノード A へのデータ送信が始まるが，ACK:14480 送信以降，新たにパケットを受け取っていないので，ノード B はパケットロスの発生を知らず，重複 ACK や SACK をノード A に送ることは無い．したがって，RTO 発生するまで，送受信が止まってしまう．RTO 後は，スロースタートから通信が再開される．

一方，図 9 は，SACK による再送が起きる場合である．この場合のパケットロスはメッセージの中間に位置する，シーケンス番号が 10136 から始まる 4 パケットである．パケットロス後，ノード B から後続するパケット DATA:16928 がノード A に到着し，そのパケットに対する ACK:10136 が返信される．この ACK は SACK であり，重複 ACK の到着を待たず，速やかに再送が実行される．

5.3.1 パケットロスおよび再送タイムアウト発生数

性能劣化に対する RTO の影響を調べるために，スイッチでのパケットロスの発生回数と，ノード 0 での RTO の発生回数を調べた．この結果を，**図 10** に示す．パケットロスは，メッセージサイズが 16KB 以上

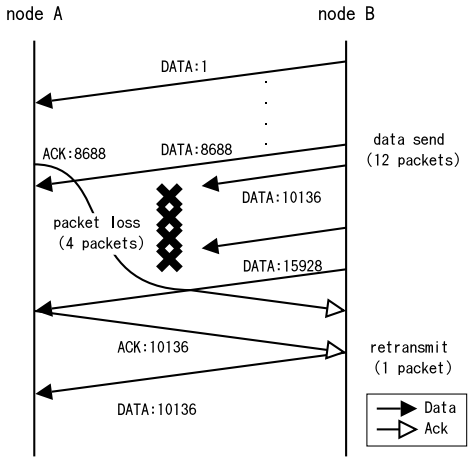

図 9 SACK による再送

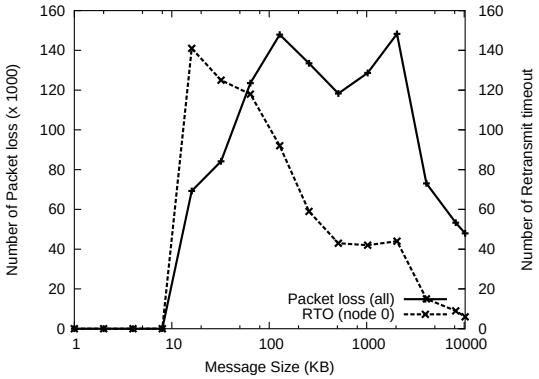

図 10 図 6 におけるパケットロスおよび再送タイムアウト発生数

のときに発生し，128KB〜2MB で多く，2MB 以上では減少する．この実験では，同一のデータ量を転送しているので，メッセージをパケットへ分割する時の影響はあるものの，転送されるパケット数（再送等を除く）はほぼ等しい．

メッセージサイズが小さい場合は，バースト転送がおきても 1 回あたりの転送量が少なく，スイッチのバッファが吸収できるため，パケットロスは起きていない．

メッセージサイズが大きいと，パケットロスの回数が大きくなる．しかし，メッセージサイズが 2MB を超えると，1 回のメッセージ転送中にセルフクロッキングが働き，パケットロスの発生数が減少すると考えられる．

一方，RTO 回数は 16KB をピークに単調減少する．これはバースト転送をスイッチのバッファが吸収できないぎりぎりの大きさのときに，メッセージの最後の部分がロスする場合が多いためと考えられる．よりメッセージサイズが大きくなると，メッセージの中間がロスしても同一メッセージの後続するパケットが到着する確立が高くなり，この場合受信側から送信側へ再送要求が送られるため RTO は発生しない．

RTO の発生回数の傾向は，図 6 における性能の劣化の傾向と一致しており，性能低下の主原因が RTO であることがわかる．

6. 議　　論

6.1 再送タイムアウト問題

RTO の発生に対する単純な解決策としては，RTO 時間を短くして，速やかな再送を促すことであるが，不要な再送を行う可能性が高くなる．RTO の時間は，通信の往復遅延時間 (RTT) の実測値に基づき決められるが，最小値が定められており，これより小さくなることは無い．これは，TCP の動作はカーネルタイマによって制御されるため，あまり細かい時間の制御はできないことや，ACK パケットの数を減らすために，受信側がある程度パケットを受け取ったあとにまとめて ACK を返す遅延 ACK が用いられることがあり，RTT のみに基づき小さな RTO を設定すると，不要な再送が頻発する恐れがあるためである．異機種が相互に接続することを想定した TCP の実装では，RTO の最小値をむやみに小さくすることはできない．

RFC2988[13] では，RTO の最小値を 1s と定めている．近年の計算機はタイマ粒度が細かくなったこともあり，Linux では，RTO の最小値を 200ms に設定している．しかし，通信する相手が既知であり，RTO を小さくしても問題がおきないことがわかっていれば最小値を変更することができる．そこで，再送タイムアウト時間の最小値を 200ms から 5ms に変更して測定を行った．

また，再送タイムアウトが発生する根本的な原因は，スイッチにおいてパケットロスが発生することである．したがって，ペーシング[11] によってバースト転送を抑制すれば，パケットロスを軽減することができ，全対全通信の性能が向上すると期待できる．

そこで，ネットワークインタフェースデバイスドライバに対して，実パケット間にスイッチで破棄されるダミーパケットを挿入することで疑似的に IFG(Inter Frame Gap) を拡大する変更を行ない，性能を測定した．IFG を制御すれば，最も精密なペーシングを行うことができる．本方式についての詳細は節 A.1 で述べる．

IFG によるペーシングを行った場合および RTO の最小値を 5ms にした場合の通信性能を**図 11** に示す．RTO の最小値は，カーネルのヘッダファイルにリテラルとして記述されているため，これを直接変更した．

IFG によるペーシングは，メッセージサイズと IFG の値の組によっては効果があり，メッセージサイズにより適切な IFG を選定する必要があるといえる．しかし，完全に性能劣化を防ぐことはできない．ボトルネックリンクがあるネットワークで全対全通信を行う

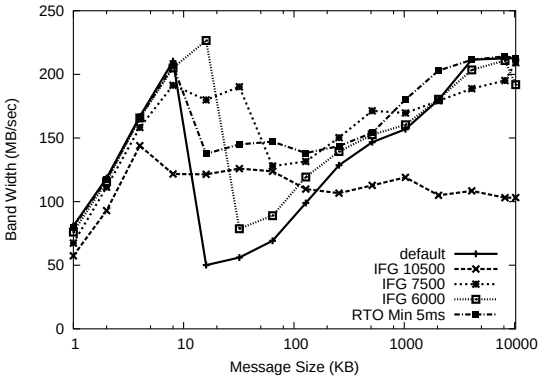

図 11 変更の効果

場合にパケットロスの原因になるのは，バースト転送の発生と宛先アドレスの重複である．バースト転送は，ペーシングによって軽減できるが，宛先の重複によるパケットロスは防げない．宛先の重複を回避するには全ノードでフェーズ移行の同期を取ることが考えられるが，遅延のある環境では同期のオーバヘッドが大きいし，遅延が異なるパスを通って到着するメッセージの到着が重なることは避けられない．

一方，RTO の最小値を 5ms にすると，RTO 発生時の通信断の時間が減少するため，性能が向上している．しかし，伝送路に遅延が存在するため，実際の RTO は 10ms 程度になっており，通信性能への影響は残っている．あらゆる通信の RTO の最小値をこのように小さくすることは適当ではないので，コネクションごとに最小値を設定できるようにすることが必要と考えられる．

6.2 MPI 通信ライブラリとの関係

RTO やペーシングによって通信性能の向上を図る場合，コネクションやメッセージサイズごとに設定を変更する必要がある．

また，MPI の 1 対 1 通信と各種の集合通信時では，通信パターンが異なり 1 ノードで使用可能な帯域が大きく異なる．

MPI レベルの通信パターンを TCP レベルで暗黙的に知ることは困難であるので，通信パターンが切り替わる時に，TCP レイヤに状況の変化を適切に通知する必要がある．

そこで，ネットワークの状況をアプリケーションに通知するとともに，RTO やペーシングの設定をアプリケーションから行える API を用意することが考えられる．

7. ま と め

本稿では，ボトルネックリンクで接続されたクラスタ間での全対全通信時の挙動について詳細に解析した．その結果，宛先の重複やバースト転送によって発生するメッセージの末尾のパケットロスによって TCP 通信の再送タイムアウトが起こり，性能が低下していることを明らかになった．この問題に対し，タイムアウト時間を短縮することによりオーバヘッドの削減を図る手法と，ペーシングによりパケットロスの発生を減らす手法が有効であることを示した．

今後，さらに効率を向上する手法を検討するとともに，本稿で述べた知見を基に TCP/IP 実装の改良を行ない，既研究成果とともに高性能 GridMPI 環境を構築していく予定である．

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) Message Passing Interface Forum, “MPI: A Message-Pasing Interface Standard”, `http://www.mpi-forum.org/`, June 1995.
2) 石川: “YAMPII もう一つの MPI 実装”, 情報処理学会，SWoPP 2004, HPC-20, 2004.
3) 石川，松田，工藤，手塚，関口: “GridMPI - 通信遅延を考慮した MPI 通信ライブラリの設計”, 情報処理学会，SWoPP 2003, pp.95-100, 2003.
4) 松田，石川，鐘尾，枝元，岡崎，工藤，児玉，手塚: “GridMPI の性能評価”, 情報処理学会，SWoPP 2004, HPC-22, 2004.
5) 松田，石川，工藤，手塚: “MPI 通信に適した通信 API の設計と実装”, 情報処理学会，SWoPP 2003, pp.101-106, 2003.
6) 児玉，工藤，佐藤，関口: “ハードウェアネットワークエミュレータを用いた TCP/IP 通信の評価”, 情報処理学会，SWoPP 2003, pp.47-52, 2003.
7) 高野，石川，工藤，松田，児玉，手塚: “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, 2003.
8) T. Kelly: “Scalable TCP: Improving Performance in Highspeed Wide Area Networks”, Computer Communication Review, Vol.32, No.2, April 2003.
9) S. Floyd: “HighSpeed TCP for Large Congestion Windows”, Internet Draft, `http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-highspeed-00.txt`, Work in progress, July 2003.
10) C.Jin, D.X.Wei and S.H.Low: “FAST TCP: Motivation, Architecture, Algorithms, Performance”, IEEE Infocom 2004, March 2004.
11) A.Aggarwal, S.Savage, and T.Anderson: “Understanding the performance of TCP pacing”, IEEE INFOCOM, 2000.
12) The Web100 Project, `http://www.web100.`

org/.
13) V. Paxson, M. Allman: “Computing TCP’s Retransmission Timer”, RFC 2988, November 2000.

付　　録

A.1 ギャップパケットを用いた疑似ペーシング

IFG はイーサフレーム間の隙間のことであり，実際のデータ転送に用いられることはない．IFG のデフォルト値は 8 バイトであるが，これを増減させることでトラヒックシェイピングのようなレート制御を実現することができる．ギガビットイーサ，かつ MTU サイズが 1500 バイトの場合は，IFG の制御によって次式のようにレート制御が可能になる．

$$\frac{1448}{(1526 + IFG)} \times 125MB/s \qquad (1)$$

一部のネットワークインタフェースカード (以下，NIC と呼ぶ) では，IFG 値をソフトウェアから変更可能である．例えば，Intel PRO/1000 は IFG 設定用に 10 ビットのレジスタを持ち，IFG を 1023 バイトまで設定することができる．しかし，それ以上に送信を絞ったり，ジャンボフレームを利用する場合には不十分な長さである．

そこで，NIC ドライバにおいて，スイッチやネットワークに副作用のないダミーのパケットを実パケット間に挿入することで，疑似的に任意の長さの IFG を挿入できるようにした．これにより任意の帯域に送信量を絞ることを可能にした．ダミーのパケットをギャップパケットと呼ぶ．ただし，本稿の実験環境のようにリンクが 1 本の場合では問題ないが，利用可能帯域や遅延が異なる複数リンクが存在する場合は，このように単純に制御できないという制限がある．

ギャップパケットのフォーマットは副作用のないことが必要であるが，現在は，IEEE 802.3x PAUSE フレームの pause time = 0，すなわち XON をギャップパケットとして用いている．PAUSE フレームは特定のマルチキャストアドレスへのイーサネットフレームであるが，スイッチのフロー制御が無効の場合，PAUSE フレームはスイッチで破棄される．

iperf による単方向通信によってギャップパケットによるレート制御の評価を行った．**図 12** より，式 1 の理論値とほど等しくレート制御できていることがわかる．

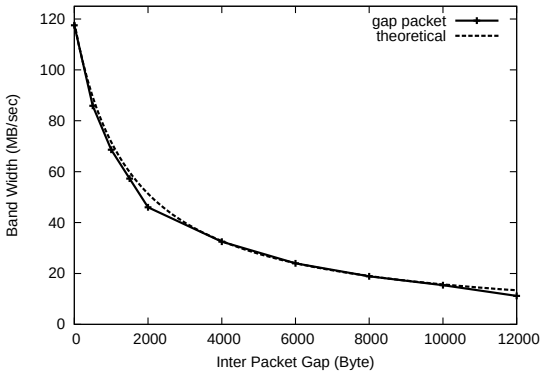

図 12 ギャップパケットによるレート制御