ハイパフォーマンス
コンピューティング 93－3
（２００３．３．１１）

ユーザレベルでのマルチリンク利用による 高バンド幅/耐故障性を持つクラスタ向け結合ネットワーク RI2N

三浦信一[†] 朴泰祐[††]
佐藤三久[††] 高橋大介[††]

コモディティネットワークである Ethernet を複数並列で用いることで，PC クラスタ向けに高バンド幅/耐故障性のあるネットワークを提供する RI2N を提案し開発中である．RI2N では，並列リンクを正常時にはバンド幅向上に用い，障害発生時にはそれに対する代替リンクとして用いることで障害に対処する．現在の実装ではユーザレベルでの高バンド幅機能が提供されている．本稿ではその概念と実装方針及び現状での問題点を整理する．FastEthernet を同時に 4 本用いた本システムでは，最大で約 40[MB/s] のバンド幅を実現することが確認された．

RI2N - Interconnection network system for clusters with wide-bandwidth and fault-tolerancy based on multiple links

Shin'ichi Miura,[†] Taisuke Boku ,[††] Mitsuhisa Sato[††]
and Daisuke Takahashi[††]

In this paper, we propose an interconnection network system named RI2N for clusters based on parallel links with commodity Ethernet which provides both wide-bandwidth and fault tolerancy. In RI2N, multiple links are used to enhance the total bandwidth in ordinary mode or redundant connection in failure mode. In current prototype implementation at user-level library provides the former function, and it achieves the maximum bandwidth of 40[MB/s] with four FastEthernet links.

1. はじめに

近年パーソナルコンピュータ (PC) が安価で高性能になっており，これをコモディティネットワークで接続したクラスタシステムが多く用いられている．PC クラスタは，従来の大型計算機と比較して安価であるにもかかわらず，同等もしくはそれ以上の性能を持っており，ハイパフォーマンスコンピューティング (HPC) の分野での注目が集まっている．現在，これらのノードではスペースや処理能力の観点から 2CPU 程度の SMP 構成を用いることが多くなっている．その反面ノードの処理能力の増大と比較して，PC クラスタではコモディティ製品を用いていることから，ノード間でのネットワークの性能が相対的に低く，より高速なネットワークデバイスの開発が求められている．しかし専用の高速なネットワークデバイスである Myrinet[1] などは高価であり，その対価格性能がノードに対して相対的に低いことが大きな問題となっている．これに加えて，現在のネットワークインタフェースカード (NIC) は，通信経路 (Switch) の遮断や，NIC の故障といったことに対してロバストではないことも，今後の大規模化に対する大きな問題になっている．

そこで我々は，最もコモディティ性の高い Ethernet において複数のリンクで PC クラスタのノード間を結合し，これらの Ethernet と Switch を複数並列に用いて PC クラスタのノード間を結合し，その並列性を通信バンド幅増強と耐故障性の双方に活用するネットワークシステムである Redundant Interconnection with Inexpensive Network(以下 RI2N) を開発中である．RI2N では，通常は並列リンクをバンド幅増強手段として用い，いずれかのリンクに故障が発生した場合は，他のリンクを代替リンクとして用いることにより，高バンド幅と耐故障性をソフトウェア制御で実現することを目指す．これまで，並列ネットワークをバンド幅増強のために用いる，いわゆる trunking システムは提案・実装されているが[2][3]，バンド幅と耐故障性への動的な対応をソフトウェアレベルで実現しているものはない．また，MPI のような API レベルで耐

† 筑波大学 大学院 理工学研究科
Graduate School of Science and Engineering, University of Tsukuba
†† 筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

故障性を実現しようという研究もある[4)6)]が，我々はこれをより一般的なレベルで実現したいと考えている．

2. RI2N: Redundant Interconnection with Inexpensive Network

近年では，CPU 性能の向上に伴う問題サイズの大規模化によって全実行時間において通信にかかる時間及び CPU 負荷も大きくなっている．クラスタでのネットワークではより高いバンド幅と，より低い通信遅延の両方が必要となっている．これに加え，クラスタを用いる計算ではたった一度のネットワークの切断が大きな問題となる．従って，耐故障性 (NIC およびその経路でのトラブルによる障害の回避) も重要な問題となっている．特に，コモディティ製品を用いる場合，従来の MPP のような並列計算機専用ネットワークの場合と異なり，強力なエラー検出・回避機能やハードウェア自体の高信頼性，パケット集中に対するスイッチのタフネス等が低い場合が多く，クラスタの大規模化に伴い，深刻な問題となってきている．

現在クラスタで多く用いられている Ethernet では，クラスタ専用のネットワーク (Myrinet 等) と比較して帯域幅・遅延時間ともに，貧弱である．また，ハードウェアとソフトウェア (TCP/IP) の両方が汎用通信用に設計されているため，もともとの用途である一般的な LAN のような環境と比べると，計算中に NIC やスイッチにかかる負荷も大きく，このため一時的なリンクの切断等が発生しやすい．これらは，クラスタシステムを大規模なメガスケールクラスまで拡張するにあたり，大きな問題になる．そこで，これを解決するために，check-point/recover といった方法も考えられているが[5)6)]，一時的なネットワークの不調のためだけにこれを行なうと大きなオーバヘッドが生じる．

現在のコンピュータでは性能向上に加え耐故障性といった面からも，マルチストリームを多用する傾向がある．代表例としてあげられるのが HDD の冗長手段として一般的に用いられている RAID (Redundant Array with Inexpensive Disk) システムである．RAID では，複数の HDD を同時にひとつの HDD と見立てて活用することにより，速度の向上と耐故障性を同時に提供している．我々はこの考えを基本に，クラスタ用ネットワークとしての特徴を加味し，高バンド幅と耐故障性を実現するネットワーク構築手段を提供することを考えている．このシステムを RI2N (Redundant Interconnection with Inexpensive Network) と呼ぶ．

RI2N では，RAID での複数 HDD に相当するものとして，NIC を複数束ね (以後，これを MP-NIC: Multiple-Port-NIC と呼ぶ)，高バンド幅と耐故障性を提供する (図 1)．最終的にはその環境をドライバレベルまたはプロトコルレベルで提供することで，低遅延を実現することを目指す．対価格性能比の良いコモディティ製品を対象とするという観点から，RI2N の現在のターゲットは Ethernet である．

RI2N ではまた，ハードウェア及びソフトウェアに関し，以下の点に留意する．

ハードウェア　NIC を複数用意するのは，最低条件である．それに加えて経路を複数用意することも必要となる．たとえばそれぞれの NIC に対し，それぞれ違う Switch につなげ，経路を複数用意し，耐故障性を高める必要がある．

ソフトウェア　通信を高速に安全に行なう環境を提供することが，RI2N の大きな目的である．不安定になってしまった機器について，当面は性能を落としながら運用を続け，さらにそれをユーザに通知して修理・復旧を促すシステムも必要となる．

今回は RI2N を実現するにあたり，まずはユーザレベルでの並列ネットワーク通信ライブラリを製作することで，高バンド幅の実現を行ない，現時点の問題点を収集する．その上で耐故障性の実装方針について検討することとする．

3. プロトタイプ

今回のプロトタイプでは RI2N を開発するにあたり，既存の socket 関数 (TCP) を元とした，高バンド幅を目的とする関数を用意し，その基本性能を調べることで今後の指標を得る．本節では，プロトタイプの実装手法，およびそれによって生じる利点と欠点についてまとめる．

なお，RI2N は現在まだ開発中であり，現段階では耐故障機能の実装が完了していない．今回のプロトタイプ実装及び性能評価は，高バンド幅化に関する機能及び性能について述べる．

3.1 実装レベル

実装のレベルには以下の 2 つが挙げられる．

ユーザレベル　TCP/IP を基本とし，multi-socket, multi-thread 等を用いた実装で，開発の容易さと高い移植性を持つことが可能になる．しかし，既存の single-stream 用 TCP/IP をベースにするため，全体の性能はこの基本性能に大きく左右される．また，システムレベルでの処理に (悪い意味での) 冗長性が生じる．

システムレベル　single socket に対するドライバレベルでの multi-port network の実装で，高いユーザ透過性をもつ．また，プロトコルを独自に制御できるためクラスタ向けにチューニングも可能である．一方で開発にコストがかかり，使用可能な NIC に制約がかかる可能性がある．

今回はユーザレベルでの実装として，既存の socket 関数にラッピングする形で RI2N 用の API を用意した．アプリケーションからは従来の socket 関数とほぼ等価な呼び出しを行ない，内部で port の数に応じ

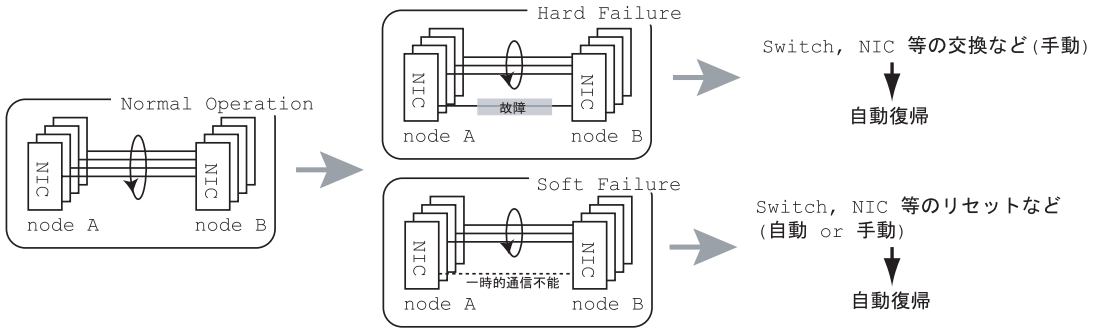

図 1 システムイメージ

て必要な socket 処理が並列に行なわれる．

送信 (send) アプリケーションから渡されたデータを，内部で一定のデータサイズ (chunk-size) に分割した後に管理用のヘッダを付加 (これらを分割パケットと呼ぶ)．その後並列化された各ポートに送信可能なポートから順に送信される．

受信 (recv) 受信可能なポートから分割パケットを受信しそれを一時バッファリングする．全データが到達した時点でヘッダ情報を元に再構成を行ない受信側のアプリケーションへ渡す．

3.2 データ分割手法

アプリケーションから受け取ったデータは，実際の RI2N 中での転送では，複数のポートを同時に用いるため分割パケットに変換する必要がある．この分割方法として**ブロック分割**と**サイクリック分割**の二種類が考えられる．ブロック分割では，データの送出時にパケット長を大きくできることで Ethernet の性能を生かすことができる．また，分割数が少なくなることで，管理用ヘッダ等の付加オーバヘッドの低減が期待できる．一方で障害発生時におけるデータ損失による再送ペナルティが大きくなり，また動的な負荷分散が行ないにくい．サイクリック分割ではデータの送出時における効率はブロック分割に比べて落ちるものの，障害時における適応性や，動的な負荷分散という点から優位にある．しかし管理用ヘッダがペイロードに比較して大きくなりオーバヘッドとなる．

これらを検討した結果，RI2N では多少のオーバヘッドは並列ストリームの持つバンド幅でカバーできると考え，故障時の対応と動的負荷分散の容易さから，データの分割手法としてサイクリック分割を用いることとした．

3.3 データ送信手法

データの送受信における複数 NIC ポートに対する並列ストリーム生成方法として，スレッドを用いる方法と，select 関数を用いる方法の 2 種類が考えられる．

Thread 法 一般にデータを同時に送受信するには，スレッドを用いた送受信がもっとも明快で簡単な手法といえる．しかしながら，スレッドを用いた場合，スレッド生成およびスレッド切り替えによるによるオーバヘッドが生じる．それに加えて，実際の使用時では RI2N 用のポートを使用するたびにスレッドを生成することなり，ラッパ関数としては適切ではない．特に，シングル CPU では SMP とは違い，同時に処理できるスレッドは 1 つであり，スレッドを用いるメリットは小さいと考えられる．

Select 法 対象となる複数のポートを select() システムコールを用いた方法により監視し，送信または受信が可能になり次第，送受信を行なう方法である．システムコールを多用するため CPU に対する負荷は大きくなるが，CPU 性能とネットワーク性能の格差が広がっていることを考慮するとトータルな性能が向上する見込みが高い．

今回の実装では両者の方法をそれぞれ試み，その性能評価を通して最終的な方法を決定する．直感的に Select 法の方が有利と思われるが，SMP マシン上では Thread 法，あるいは Thread 法と Select 法をミックスする方法も有効と考えられる．

3.4 実装上の問題点

プロトタイプでの実装では OS として Linux を用いたが，開発過程でいくつかの問題が発生した．

arp 問 題

ノード内の NIC をすべて同じスイッチングハブ (同一物理ネットワーク) に接続した場合，相手の MAC アドレスの結果が正しく引けない症状が発生した．これは，アドレス解決の際に，同一ノードにおける全 MP-NIC のアドレスが返されてしまうためである．これを解決するために，今回は MAC アドレスを静的に決定するようにした．

routing 問題

ノード内の NIC をすべて同じサブネットに接続した場合，送信を行なうポートがある一定のポートに限定されてしまう現象が発生した．Linux の場合，送信を行なうポートは，IP アドレス上で一番大きいアドレスのものになることが分かったが，これに関する解決策

表 1　実験環境

PC	DELL PowerEdge 1600SC 2 台 (Pentium4 Xeon 2.8GHz 2-way SMP)
NIC	Adaptec Quartet 64 (100base-TX Ethernet × 4 ports) Intel PRO/1000MT (1000base-T Ethernet × 2 ports)
Switch	PLANEX FMX-0248K (Layer2) (48 Port Fast Ethernet Switch) PLANEX FMG-24K (Layer2) (24 Port Gigabit Ethernet Switch)
OS	Linux 2.4.18

は現在見つかっていない☆．今回はこのため MP-NIC すべてを違うサブネットに所属させることで解決した．

複数の service port の確保

今回の実装はユーザレベルで複数の socket を用いているため，それらに個別の service port を割り振る必要がある．API 上で bind() システムコールと互換性を保つため，現在の実装では MP-NIC の bind() において空きポートをスキャンし，それを代表ポート (本来のユーザ指定ポート) を通じて client 側に渡し，最終的に並列リンクを確立する．RI2N ではクラスタを対象としているため，一般的にすべてのポートがアクセス可能であると仮定し，このような実装を取った．

4. 性 能 評 価

サイクリック分割において，ネットワークの転送性能は chunk-size に大きく影響されるものと考えられる．chunk-size が小さすぎると本来の Ethernet での MTU サイズの問題等で TCP の性能が発揮できず，RI2N でのヘッダ付加による，分割のためのオーバヘッドも大きくなる．一方で chunk-size が大きすぎるとユーザデータのサイズが小さい場合の並列転送の効果が期待できないばかりか，通信経路中 (特に Ethernet Switch) でのバッファサイズの影響を受けやすくなり，やはり性能が低下する恐れがある．よって，今回はこれらの影響と最適な chunk-size について調査を行なう．FastEthernet での Thread 法と Select 法での性能比較を行ない，最後に最近急速な低価格化が進んでいる GbE での評価を行なう．なお今回の実験では socket buffer size は固定である．(いくつか試みたが buffer size による影響はほとんどなかった．)

4.1 実 験 環 境

今回用いた実験の環境について，表 1 に示す．データの転送には 2 台の PC 間に 1 つのスイッチを挟み，MP-NIC のポート数に合わせた複数の Ethernet ケーブルを張り，複数リンクを同時に構成して用いる．FastEthernet を用いた場合 4 本のリンクを同時に用いると，理論最大性能は 50[MB/sec] になる．GbE の場合は 2 本のリンクで，理論最大性能は 240[MB/sec] となる．

今回は，機器の都合上 Ethernet Switch は 1 台のみ用意し，対象となるリンクはすべて同一の Ethernet Switch に接続されている．

実験では，アプリケーションから渡されたデータについて，複数回のピンポン通信を行ない，そのときのアプリケーションデータサイズと各 chunk-size を変化させたときの転送性能の変化について調査を行なう．以後，性能評価グラフは横軸がアプリケーションレベルでのデータサイズ [KB]，縦軸が同レベルでの転送性能 [MB/s] である．パラメータとして，いくつかの chunk-size を取っている．

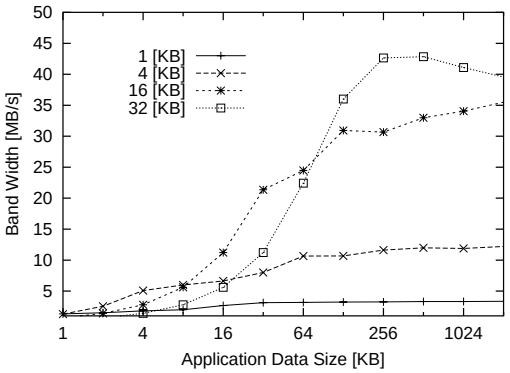

図 2　実験結果 [Thread] Fast Ethernet(100Mbps)

4.2 Thread 法 [Fast Ethernet]

FastEthernet において Thread 法による実装を行なった場合の送受信実験の結果を図 2 に示す．このプロトタイプでは，実際に送受信のたびにスレッドの生成を行なっている．そのためスレッド生成のコストに対してアプリケーションデータサイズが大きくならなければ，実際に性能は向上していない．結果からもわかるとおり，ここでは，chunk-size が 32[KB] のとき性能が最もよい．chunk-size が小さい場合の性能が出ていないのは，各 Thread が送受信のたびに他のスレッドとの同期をとるなどのオーバヘッドがあるためである．一方，アプリケーションデータサイズが小さいときは性能は出ていない．使用できるポート数が使いきれていないのに加えて，chunk-size に比べてアプリケーションサイズが小さいため無駄なデータ量を送っているためである．また当然ながら chunk-size 以下のデータサイズに対しては並列化されていない．これを解消するには，chunk-size をアプリケーションデータサイズに対して動的に変更するか，使用するポート数を制限することで性能を上げるしかない．

明らかに今回の実装ではスレッドの生成が大きなネックとなっている．今後の実装ではスレッド pool を用いた方法をとらなくてはいけない．

☆ MPP 用の OS，たとえば日立 HI-UX/MPP では socket option により送出ポートを正しく指定できるが，MP-NIC をもつ Linux ではうまく動作しなかった．

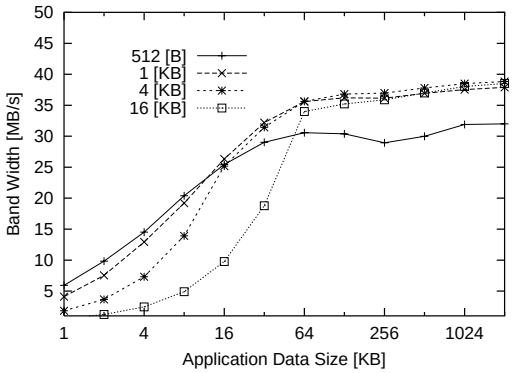

図 3 実験結果 [Select] Fast Ethernet(100Mbps)

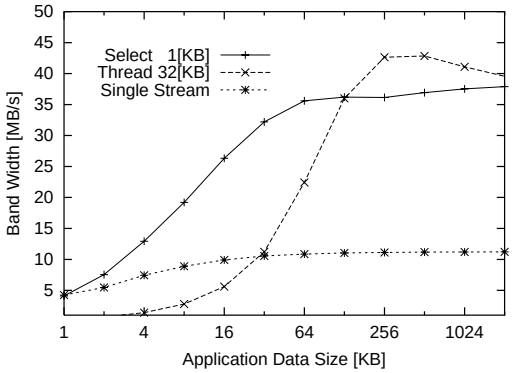

図 4 実験結果 比較 Fast Ethernet(100Mbps)

4.3 Select 法 [Fast Ethernet]

FastEthernet において Select 法による実装を行なった場合の送受信実験の結果を図 3 に示す． 一般的には，chunk-size が細かくできたほうが，動的な負荷分散や故障発生での再送ペナルティが小さくなる点で優位である．しかしその一方でヘッダの付加によるオーバヘッドや，TCP/IP や Ethernet の特性などで一概に小さい chunk-size が性能を出せるとは限らない．今回の結果では 1[KB] の時が，最大性能の点やアプリケーションデータが小さい時の性能から見てよい結果となっている．この 1[KB] という値は，Ethernet の MTU が 1500[B] 程度という点からみても適切値と考えられる．

4.4 Thread 法と Select 法の比較

Thread 法と Select 法でそれぞれ最も性能が優れている chunk-size における性能比較を図 4 に示す．Thread 法では 32[KB]，Select 法では 1[KB] を参考データして選択した．また，参考のために 1 つのポートを用いた single stream 転送 (通常の TCP/IP) の性能も併せて示す． 性能的に優れているこれら 2 つを比較した場合，最大性能ではやや Thread には及ばないものの，全体的に Select 法が優れている．全体的に安定した高性能を得ることを考え select 法を採用する．ただし thread 法にスレッド pool を適用するという改良の余地は残されている．また両方法ともにアプリケーションデータサイズが大きくなった際は，chunk-size が大きいほうが性能がよくなる．今後は chunk-size をスイッチする方法を考慮しなくてはいけない．

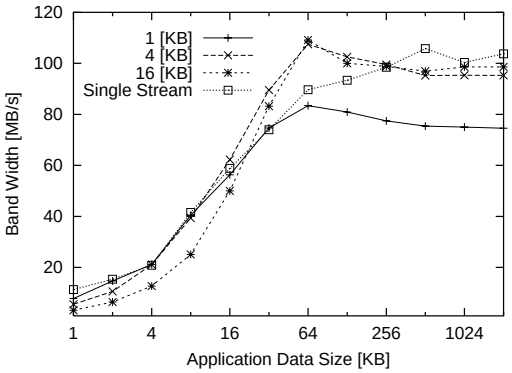

図 5 実験結果 [Select] Gigabit Ethernet(1000Mbps)

4.5 Gigabit Ethernet

GbE を用いた RI2N での結果を図 5 に示す．GbE では Select 法だけを用いて計測を行なった． 結果では 2port での理論最大性能 240[MB/s] に対して，1port 分程度の速度となり，性能が思った以上に出ていない．使用したバスは PCI-X(64bit,100MHz) であるため，このバンド幅が不足しているとは考えられない．GbE では，ネットワークの処理速度に対して，send()/recv() のシステムコールの処理に時間がかかり，chunk-size の大きさによっては，該当ポートへのデータ供給が追いつかない．それに加え，送受信前後での分割パケットの生成や再構成に時間がかかり，相対的にその部分が大きくなりがちとなる．GbE の潜在能力を考えるならば，まだまだチューニングが必要である．なお今回の実験では MTU はあえて変更はしなかった．これは，使用できる機材の制限を与えるのに加えて，安定性という面から考えてまだ使用できないと判断したためである．

今回の GbE での実験では，本来の性能は出し切れてはいないにもかかわらずアプリケーションサイズが小さいときなどでも，全体的な性能は FastEthernet に比べて大変よくなっており，現在の GbE の低価格化 (特にスイッチ) が進むことで有望なことがこの結果でもわかる．このことにより，今後は GbE を大きく意識する必要があると考えている．また GbE MP-NIC を用いることによる対故障性は重要であり，チューニングを進めてある程度の性能が得られれば有望なシステムとなる．

5. 対故障性 実装方針

現在検討を進めている耐故障性機能について，以下のように考えている．

5.1 障害の検知

障害検知方法として以下の方法を考えている．

- **バッファの調査 (他のリンクとの比較)**
  RI2N で用いているそれぞれのポートについてバッファを確認し，長時間変化が無い場合故障と断定する．
- **タイムアウト**
  特定のポートからある一定の時間送受信ができなかったとき，故障と検知する．
- 送信数の確認
  RI2N で用いているポートで送受信した分割パケット数を確認し，他のポートでの送受信数と差が大きくなってきたときに，NIC または経路に異常が生じたと判断し，故障と検知する．

5.2 対 処 方 法

- **バッファリング アプローチ**
  送受信双方で一定サイズのバッファ(ウインドウ)を用意し，リンクの failure 発生時には，他のリンクで再送を行なう．
- **RAID5 アプローチ**
  送信に常に冗長データを含め送信を行なう．これを用いると故障の検知に伴う再送要求が減るため，一時的な極端な速度低下は発生しない．しかし，オーバヘッドが増大するため，通常の通信時においては広帯域の確保が難しくなると考えられる．また，ある程度のポート数 (3port 以上) が必要になる．

現在のユーザレベル実装では，RAID5 アプローチは大きな負荷とオーバヘッドを伴う可能性があり，また，Ethernet で用いられる TCP/IP では，ある程度の信頼性が保たれている．そのことにより，バッファリングアプローチが最適であると考えられる．今後は予備実験に基づく比較検討を行なう．

6. 終 わ り に

本稿では，コモディティEthernet を用いた冗長/高バンド幅ネットワーク RI2N の概要と，予備実験の結果について述べた．今後はこの RI2N を以下の内容を考慮した上で拡張していく予定である．

耐故障性の検討と実装

RI2N に耐故障性機能について，前節で挙げたいくつかの実装方法について予備評価を行ない，実装方法を決定する．

実アプリケーションへの適用

RI2N を MPICH 等に適用したうえで，ベンチマーキングを行ない，アプリケーションの特性及び各種条件に対する性能を評価する．

スレッド pool の実装

今回評価を行なった Thread 法の実装ではスレッド pool を用いておらず，毎回スレッドを生成・消去している．pool を用い，このオーバヘッドを削減した場合の両手法の検討を再度行なう必要がある．

遅延時間の考慮

今回はバンド幅のみに注意を行ない，遅延時間については考慮に入れなかったが，遅延時間の影響に関する実験を行なう．

他のシステムソフトウェアとの連携

クラスタ管理システムなどのシステムとの連携について検討する．

謝　　辞

本研究を行なうにあたり，貴重な助言・提言を頂いた CREST「メガスケールクラスタ研究チーム」のメンバーである，豊橋技術科学大学中島浩教授，東京大学中村宏助教授，東京工業大学松岡聡教授に深く感謝します．本研究の一部は科学技術振興事業団「戦略的創造研究推進事業 (CREST) — 情報社会を支える新しい高性能情報処理技術 —『超低電力技術によるディペンダブルメガスケールコンピューティング』」および文部科学省 科学研究費補助 (基礎研究 (C),12680327) による．

参 考 文 献

1) Myricom, Inc.
http://www.myri.com/myrinet/.
2) 住元 真司, 堀 敦史, 手塚 宏史, 原田 浩, 高橋 俊行, 石川 裕
高速通信機構 PM2 の設計と評価, 情報処理学会論文誌，Vol. 41, No. SIG 5(HPS 1), pp. 80–90, 2000.
3) IEEE. 802.1ad, http://www.ieee.org/.
4) G. Bosilca, A. Bouteillier, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, A. Selhikov, MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes, SC 2002, 2002.
5) 西岡利博, 堀敦史, 手塚浩史, 石川裕.
クラスタにおけるコンシステントチェックポイントの実現, JSPP2002 論文集, pp. 229-236, 1999.
6) 高宮 安仁, 松岡 聡. ユーザー透過な耐故障製を実現する MPI へ向けて, JSPP2002 論文集, pp. 217-224, 2002.
7) Andrew S.Tanenbaum.
水野 忠則, 相田 仁, 東野 輝夫, 太田 賢 訳
COMPUTER NETWORKS, Third Edition, ピアソン・エデュケーション, 1999.