2003－HPC－95　(17)
2003／8／5

GridMPI – 通信遅延を考慮したMPI通信ライブラリの設計

石川　裕[††,†]　松田元彦[††]　工藤知宏[††]
手塚宏史[††]　関口智嗣[††]

グリッド上での通信遅延を考慮した MPI 通信ライブラリである GridMPI の設計について述べる。実現するために解決しなければならない課題と、それに対する既存技術について考察を行なった後に、GridMPI のソフトウェアアーキテクチャについて述べる。既存 MPI 通信ライブラリと違い、ネットワークトポロジと通信遅延を考慮した通信層を導入し、グリッド上での通信遅延によるトポロジおよび低レベル通信ライブラリの違いを透過的に見せている。

GridMPI – The Design of a Latency-aware MPI Communication Library

Yutaka Ishikawa,[†] Motohiko Matsuda,[††] Tomohiro Kudoh,[††]
Hiroshi Tezuka[††] and Satoshi Sekiguchi[††]

This paper presents the design of an MPI library implementation, called *GridMPI*, which realizes a latency-aware interoperable communication in the Grid environment. After presenting the issues to realize such an MPI library implementation, the GridMPI software architecture is proposed. Unlike other existing MPI ipmlementations, the GridMPI introduces a layer, called the `latency-aware communication toplogy`, which realizes both the transparency of network topology in respect to communication latency and the transparency of lower-level communicarion libraries.

1. はじめに

グリッドにおける応用の一つであるインターネット上の計算資源を駆使した並列処理には、大きく 3 つのタイプがある。ジョブを計算資源に配布する、いわゆる分散コンピューティング、複数の依存関係のあるアプリケーションを計算資源に割り当てて処理を行なう、いわゆるワークフロー、並列アプリケーションを複数の計算資源上で並列計算するタイプ、である。

1990 年代初頭より、これらインターネット上の並列処理の可能性に関する研究が行なわれてきた。当時のインターネットのバンド幅は低く、通信遅延も高かった。このため、並列アプリケーションを複数の計算資源上で並列処理することは現実的でなかった [4]。現在では、つくば WAN のような 10 Gbps クラスのメトロポリタンネットワークが整備され、つくばと東京間の往復通信遅延も 4 msec 程度となっている。

このようなインターネットの性能向上により、従来非現実的とされてきた並列アプリケーションを複数の計算資源上で並列計算するタイプの処理が現実味を帯びてきた。論文 [1),2)] において、我々は、0〜20 msec の往復通信遅延を人工的に与えるルータにより同一構成のクラスタ 2 台を接続した環境において、NAS 並列ベンチマークによる性能評価を行なった。4 msec までの遅延では、クラスタ 1 台による実行とクラスタ 2 台による実行を比較すると、CG や LU で 1.2〜2 倍性能向上することが検証されている。しかし、20 msec まで遅延を大きくすると、性能は 0.5 倍程度にまで落ちてしまう。論文 [1),2)] では、さらに、既存 MPI 通信ライブラリである MPICH-G2, MPICH-SCore, MPICH-P4 の性能評価を行なっている。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、MPI 通信ライブラリが物理性能を引き出せていないことを明らかにした。

このような評価を通して、通信遅延を考慮した MPI 通信ライブラリを実現するには、TCP/IP の実装、カーネルインターフェイス、MPI ライブラリ実装レベル、の各レベルの再設計を行なう必要があると判断し、新しい MPI 通信ライブラリ実装である GridMPI を開発している。我々は、4 msec 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からメトロポリタンエリアネッ

† 産業技術総合研究所/東京大学大学院情報理工学系研究科
†† 産業技術総合研究所

トワーク上に位置する 1,024 台規模のクラスタやベンダの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。

本稿では、次章においてグリッド環境上で MPI 通信機構を実現するための課題として、トランスポート層、トポロジに依存した通信、耐故障機能、セキュリティ、標準化に関して議論を行ない、第 3 章で、我々が開発している GridMPI の設計について述べる。

2. 課題と既存技術

2.1 トランスポート層

広く利用されている Unix 系や Linux 等の OS における TCP/IP の実装では、通信遅延が大きいネットワークの物理性能を十分に引き出せない。これを解決するために幾つかの試みが行なわれている。

2.1.1 TCP バッファチューニング

TCP はスライディング・ウィンドウ・プロトコルと呼ばれる方式でデータ送信のフロー制御を行なっている。送信側は、受信側が公告したウィンドウサイズ分だけ受信側からの ACK を待たずに非同期にデータを送ることが可能である。受信側が公告するウィンドウサイズは、受信側の TCP バッファサイズに依存する。

ウィンドウサイズは非同期に送れるデータサイズに等しい。ACK が戻ってくるまでの間に非同期送信が完了してしまうと、それ以上のデータ送信は行なわれない。通信路にデータを満たすためには、ACK が戻ってくる時間、すなわち、RTT(Round Trip Time) 時間だけ送信できなければならない。

Linux 2.4 では、最大 170 KBytes の受信バッファが確保される。これは、1 Gbps の場合、RTT が 1.3 msec 以下でないと通信路が埋まらないことになる。このように、通信遅延とバンド幅に応じて送信、受信バッファを調整する必要がある。

2.1.2 輻輳制御の改良

送信側においてネットワーク上で輻輳が生じたと判断すると、スロースタートと呼ばれるモードになる。本モードでは、非同期に送信できるセグメントを 1 として、現在のウィンドウサイズ (cwnd と呼ぶことにする) の 1/2 になるまでは、ACK が戻ってくる毎に送信量を増やしていく。これにより 1/2*cwnd まで、非同期に送信できるセグメント数は指数的に増大する。その後、輻輳回避モードとなり、緩やかに送信側のウィンドウサイズは増加する。

輻輳時のスロースタートの影響で非同期に送信できるセグメント数が減少し、通信バンド幅の低下を招く。Scalable TCP[11] では、輻輳時に非同期送信量を 1 セグメントにまで下げるのではなく、0.875*cwnd にして、最初から輻輳回避モードにする手法が提案されている。

HighSpeed TCP では、輻輳発生時、輻輳回避時のそれぞれに対して、ウィンドウサイズの減少値、増加値をパケットの喪失度合で規定する方法を提案している [7]。

2.1.3 ストリームの多重化

輻輳によるスロースタートが頻繁に生じても通信ストリームの多重化でバンド幅を稼ぐことができる。この手法を用いているアプリケーションに、GridFTP[5] などがある。既存 TCP/IP の輻輳時のアルゴリズムによらないが、性能チューニングのためにユーザが多重度の調整を行う必要がある。

2.1.4 遠隔 Read/Write

RDMA コンソーシアム [10] により規格化されている RDMAP (Remote Direct Memory Access Protocol) は TCP 上に遠隔 read/write 機構を導入し、バッファサイズに依存したウィンドウ制御を回避することが可能となる。遠隔 read/write では、遠隔側のメモリアドレスを事前に知っておく必要がある。このために、遠隔操作の前に送受信側でネゴシエーションのフェーズが入る。通信遅延が大きいネットワークでは、このネゴシエーションのコストが健在化し、必ずしも遠隔 read/write により性能が向上するとは限らない [2]。

2.2 大規模クラスタとの接続

第 2.1.1 節で述べたように、TCP バッファの最適サイズは、通信バンド幅と通信遅延に依存する。1Gbps で往復遅延時間が 4 msec のネットワークでは、最低でも 524 KBytes の TCP バッファ領域が必要となる。このようなネットワークにおいて 1,024 ノード規模のコネクションを想定すると、各コネクション毎に送信および受信用の TCP バッファとして 524*2 KBytes となり、一台のノードが必要とする送受信バッファの総計は 1GByte にも及ぶ。

また、そのような台数を持つクラスタが TCP/IP V4 プロトコル上でグローバルな IP アドレスを持つ可能性は少なく、プライベート IP アドレスの運用が行なわれていることが多い。この場合、ゲートウェイ上で NAT を経由して通信することになり、ゲートウェイにおける輻輳が生じる可能性がある。

我々が知る限り、このような課題を解決した研究は存在しない。

2.3 カーネル API

2.3.1 非同期送信

集合通信のように全てのノードが送受信を行なう場合、送信バッファが一杯になり送信がブロックされることがある。単純な例を図 1 に示す。ここでは、各ノードは隣のノードにデータを送信し、もう一方の隣のノードからデータを受信している。各

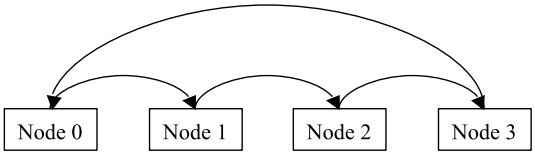

図 1 集合通信の例

表 1 MPI における受信メッセージ指定

タグ	送信プロセスのランク
特定の値	特定のランク
特定の値	指定せず (ANY)
指定せず (ANY)	特定のランク
指定せず (ANY)	指定せず (ANY)

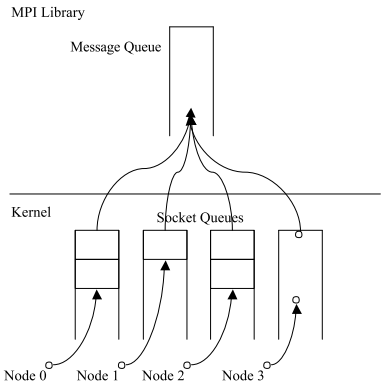

図 2 MPI vs TCP/IP

ノードが、一斉に送信を開始すると、各ノードのアプリケーションは到着したデータを受信バッファから取り出さないので受信バッファが一杯になる。これにより、送信プリミティブがブロックしてしまう。

図 1 で示す単純な例の場合には、例えば、奇数番号のノードは送信を先行し、偶数番号のノードは受信を先行する、という手法で、受信バッファの溢れを防ぐことが出来る。しかし、通信相手が動的に変わるようなプログラムでは、このような手法は使用できない。

このようなプログラムに対処するため、MPICH-G2 では、非同期 I/O を用いて、送信バッファが空いた時に送信するという手法を採用している。しかし、これは、select システムコールを使用しているために、次節で述べるスケーラビリティに関する問題が生じる。

2.3.2 select & read サイクルの最適化

まず、TCP/IP における通信モデルと MPI 通信ライブラリの通信モデルを比較する。MPI 通信ライブラリは、コミュニケータによる通信相手の集合上での送受信ととらえることができる。そして、受信側は、タグと送信プロセスのランクを指定してメッセージを受信するが、この指定の組合せには、表 1 に示す通り、4 通りある。送信側のランクを指

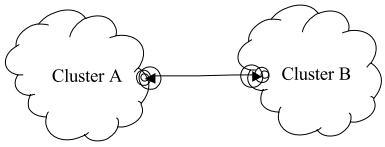

図 3 グリッド上のトポロジの例 1

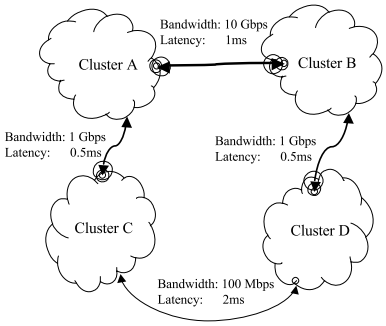

図 4 グリッド上のトポロジの例 2

定せずに任意の送信元からのメッセージを受信することが可能であるという特徴を持つ。さらに受信操作にはブロック、ノンブロックの 2 種類を提供している。一方、TCP/IP はポイント－ポイント間で信頼性のある通信を提供するコネクション型ストリーム通信である。データ受信はコネクション毎に行なわれる。

TCP/IP を使用して MPI 通信を実装するためには、各プロセス（ランク）は他のプロセス全てとコネクションを張る必要がある。MPI が提供している任意のランクからのメッセージ受信のためには、全てのコネクションからのメッセージ到着を待つ必要がある。

すなわち、図 2 に示す通り、TCP/IP レベルでは、各コネクション毎にメッセージキューが作られるが、MPI ランタイムライブラリでは、これらのキューからメッセージを取り出して一つあるいは複数のキューで管理することになる。実際、socket API の select システムコールで複数コネクションでの受信を待った後、read システムコールでメッセージを読むことになる。このために、select および read システムコールのオーバヘッドが生じる。Linux 2.4 の場合には、コネクション数 n に対して、$O(n)$ の処理時間がかかる。

2.4 トポロジに依存した通信機構

MPI 通信におけるバリア同期や集合通信のような全体通信に対して、各ノード間での通信遅延およびバンド幅を基準にしたネットワークトポロジに応じて最適な通信を実現する必要がある。例えば、図 3 において、インターネット上の 2 つのサイトのクラスタ A と B 上で MPI アプリケーションを動かすことを考える。クラスタ内のノードが 1 Gbps

スイッチで接続されているとすると、各クラスタ内の Bisection バンド幅は 16 Gbps となる。一方、クラスタ間も 1 Gbps でつながっているとすると、全クラスタにおける Bisection バンド幅は 2 Gbps しかなく、さらに通信遅延がクラスタ内よりも高くなる。

このような環境において、例えば、リダクションなどのような集合通信を効率良く実現するために、MPICH-G2 では、各クラスタ内でリダクションを行ない、各クラスタの代表ノードに結果を送付し、クラスタ内で全体のリダクションを行なう、という手順をとっている。

図 4 に示すような通信遅延とバンド幅を考える。Cluster A と Cluster B は高速専用通信路でつながり、その性能は 1msec の通信遅延、10 Gbps のバンド幅である。Cluster A と Cluster C は同一サイト内にあり、通信遅延 0.5msec、バンド幅 1Gbps である。Cluster B と Cluster D は同一サイト内にあり、通信遅延 0.5msec、バンド幅 1Gbps である。また、Cluster C と Cluster D 一般インターネット経由でのルーティングとなっていて、その通信遅延が 2msec、バンド幅は 100Mbps である。

このようなネットワークトポロジでは、Cluster C と Cluster D の通信を Cluster A および Cluster B を中継した時の通信性能と比較して、最適通信パターンを求めるべきである。しかし、MPICH-G2 では、クラスタ内かクラスタ間のような単純なトポロジを想定した通信の最適化は行なわれているが、図 4 のようなトポロジと通信遅延に基づく通信の最適化は行なわれていない。

2.5 ゲートウェイ

第 2.2 節で述べたように大規模クラスタとの接続では、ゲートウェイ上の NAT 経由となる可能性を指摘したが、大規模クラスタに限らずサイト間の接続では、ファイアーウォールを実現しているゲートウェイを経由することが多い。このような場合、ゲートウェイの処理能力がボトルネックになる可能性が高い。

2.6 耐故障機能

グリッド環境下においては、サイト毎の運用ポリシにより、長時間にわたるジョブを連続して実行することが不可能なことが想定される。このために、グリッド環境下でのチェックポイント機能を提供する必要がある。グリッド環境下でのチェックポイントファイルの効率的かつ効果的保存方法は未解決の課題である。

2.7 セキュリティ

以下の 3 つのセキュリティに関する対応が必要である。

- ジョブ起動に関して
  管理が異なる複数のサイト上でのジョブ起動時のユーザ認証であり、Globus における OGSI[9] に相当する機能である。
- ユーザプログラムに関して
  ユーザプログラムがウィルスに感染している等により、システムに被害を及ぼす可能性がある。プログラムは sandbox 技術等により隔離されて実行されるべきである。
- ネットワーク
  ネットワーク上のデータ通信において、データの改竄がないようにする必要がある。MPICH-G2 では SSL によるメッセージダイジェスト機能が提供されている。

これらセキュリティに関しては運用形態によっては、考慮する必要のない項目もある。例えば、ネットワークが専用線でつながり物理的にデータの盗聴や改竄が起こらないことが保証されている場合には、ソフトウェアレベルでのセキュリティ対応は必要ないだろう。

2.8 標 準 化

MPI 規格は、MPI 関数がエラーを起こした時のエラーの種類を詳細に規定していない。また、エラーが生じた時の挙動についても正確に規定されていないことが多い。実際、我々は、MPICH の処理系と LAM/MPI の処理系では、エラー時の振舞いが異なることを確認している [6]。現状の MPI の規格は、正常に動作するプログラムの動作は規定されているが、プログラムにバグが潜んでいる場合のプログラムの挙動はシステムにより異なっていると言える。

実装の異なる MPI 通信ライブラリを使用したプログラム同士をつなげるためのネットワークプロトコル規約として、IPMI[8] がある。IPMI は、プロセス生成時のプロトコル、一対一通信のプロトコル、集合通信時の通信手順を決めている。

しかし、Grid 上で使われている MPICH-G2 は IPMI 規約に準拠していない。また、IPMI の集合通信における通信手順は通信遅延を考慮していないという問題がある。

このように、MPI アプリケーションがグリッド上で使われるようになるためには、既存 MPI の規格をさらに詳細化した規格を標準化していく必要があると考えている。

3. 設 計

3.1 ソフトウェアアーキテクチャ

GridMPI のソフトウェアアーキテクチャを図 5 に示す。GridMPI は、東大で開発されている YAMPII[12] をベースとし、プロセス生成、通信機構、のそれぞれに対して、様々な既存ライブラリが利用可能となるようなモジュラな構造としている。

<table>
<tr><td colspan="8">MPI Core</td></tr>
<tr><td colspan="4">RIM</td><td colspan="4">Grid ADI</td></tr>
<tr><td rowspan="3">SSH</td><td rowspan="3">RSH</td><td rowspan="3">DUROC</td><td rowspan="3">Vendor MPI</td><td colspan="3">Latency-aware Communication Topology</td><td rowspan="3">Other Comm. Library</td></tr>
<tr><td colspan="2">P-to-P Communication</td><td rowspan="2">Vendor MPI</td></tr>
<tr><td>TCP/IP</td><td>PMv2</td><td>Others</td></tr>
</table>

図 5 GridMPI のソフトウェアアーキテクチャ

3.1.1 MPI Core

MPI Core 部は、YAMPII で提供される部分であり、グループ、コミュニケータ、トポロジなどの機能が実現される。MPI Core 部の下位レイヤに RIM(Remote Invocation Mechanism) 層と Grid ADI 層を定義し、これらの層が提供するサービスを利用して MPI Core 部が実現される。

3.1.2 RIM(Remote Invocation Mechanism)

本モジュールは、MPI Core 部に対してリモートプロセス生成機能およびプロセス間交信に必要なコネクションの初期化手段を提供する。さらに、Latency-aware Communication Topology 層が必要とする通信遅延に関する情報収集用の API が定義される。

RIM は、rsh や ssh、Globus ツールキットが提供する DUROC などで実現される。

3.1.3 Grid ADI

Grid ADI (Abstract Device Interface) は、MPI Core が使用する基本通信機構の API を規定する。Grid ADI の下位層には、通信遅延を考慮したネットワーク層および他の通信ライブラリがある。

Grid ADI は、一対一の非同期送信・受信、集合通信が定義される。一対一の非同期送信・受信機構は Grid ADI の下位層で実現されていなければならない。下位層が集合通信を提供しない場合には、Grid ADI 層において、一対一の非同期送信・受信を使用した集合通信が実現される。

Grid ADI の下位層に位置する Latency-aware Communication Topology は、集合通信も実現する。これは、通信遅延によって集合通信の実現方法を適応的に実現しなければならないからである。

3.1.4 Latency-aware Communication Topology

Latency-aware Communication Topology 層は、GridMPI における通信遅延に基づく通信の最適化を実現している層である。Grid ADI 上での一対一通信は、送信バイト数と通信遅延および通信バンド幅から最適な通信経路を探す。第 2.4 節で述べたように、この経路は、ネットワーク上はルーティングされていなくても、ノードを経由して MPI 通信パケットを送信する。すなわち、Latency-aware Communication Topology 層でメッセージのルーティングを行なうことになる。ルーティング情報は、MPI のヘッダに格納される。

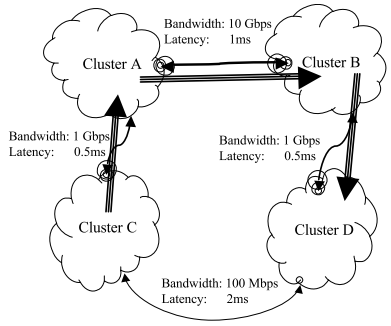

図 6 MPI_Bcast 実現例

集合通信は、大きく以下のように分類されて実現される。

(1) 全対全通信
MPI_Alltoall や MPI_Allgather などの通信。それぞれの MPI 関数に応じてトポロジに応じた通信の最適化の余地がある。

(2) 一対全通信
MPI_Bcast や MPI_Scatter などの通信。MPI_Scatter は異なるデータが送信されるために、1 対 1 通信に簡約できる。
MPI_Bcast は同一データが全てのプロセッサにブロードキャストされるために、MPI_Bcast 用にトポロジに応じた通信の最適化の余地がある。例えば、図 4 で示したようなトポロジにおいて Cluster C 上のあるノードがルートとなるブロードキャストでは、図 6 に示す通り、メッセージは Cluster A に送り、Cluster A が Cluster A 内にブロードキャストするとともに、Cluster B にメッセージをフォワード、Cluster B は Cluster B 内にブロードキャストするとともに、Cluster D にメッセージをフォワード、というふうにブロードキャストすべきである。

(3) 全対一通信
MPI_Gather などの通信。各プロセッサが同一プロセッサにデータを送信する。1 対 1 通信に簡約できるが、経路選択に関する最適化

の可能性がある。

(4) リダクション
MPI_Reduction などの通信。MPI_Reduction 用にトポロジに応じた通信の最適化の余地がある。

(5) バリア
MPI_Barrier は通信遅延センシティブな関数である。通信遅延を指標としたネットワークトポロジに基づき最適なバリア同期を実現する。

3.2 ベンダ MPI

商用並列コンピュータは、固有の高速ネットワーク用に独自の MPI 通信ライブラリ（以降、ベンダ MPI ライブラリと呼ぶ）を持つ。このような商用並列コンピュータ上では、並列コンピュータ内はベンダ MPI ライブラリを使用し、他のコンピュータ間では TCP/IP を使用した通信を行なう。ベンダ MPI の集合通信が、一対一通信で実現するよりも効率良く実現されている場合には、商用並列コンピュータ上ではベンダ MPI の集合通信が使用される。

ベンダ MPI の集合通信機構は用いずに、一対一通信機構を使用する、すなわち、ベンダ MPI を高性能な通信通信デバイスとして使用する方法がある。本方式では、使用するコンピュータ全てが TCP/IP あるいはベンダ MPI による一対一通信機構を使用するため、全体の通信遅延と通信バンド幅を考慮して集合通信を実現できる利点がある。

3.3 TCP/IP と MPI の新しいインターフェイスの提供

第 2.1.4 節および第 2.3.2 節で述べた課題については、論文 [3] において、新しい API を提案し解決している。

4. おわりに

本稿では、最初にグリッド環境上で高性能な MPI 通信機構を実現する際の課題および既存技術について述べた後、我々が現在設計中の GridMPI の概要を述べた。通信遅延を考慮した高性能通信を実現するために、GridMPI では、トポロジに応じた通信パターンを実現する枠組を提供する。また、socket API と MPI 通信モデルのミスマッチを解決する機構も実現される [3]。さらに、商用並列コンピュータ上で提供されているベンダ MPI を柔軟に使用する機構も提供されている。

GridMPI は、東大で開発している YAMPII[12] をベースにして、現在、開発を進めている。2003 年秋には GridMPI の最初のバージョンをリリースする予定でいる。

謝　辞

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) 松田、石川、工藤、「WAN 上の複数クラスタによる単一 MPI アプリケーションの性能評価」、情報処理学会、HOKKE'03, 2003.
2) M. Matsuda, Y. Ishikawa, and T. Kudoh, "Evaluation of MPI Implementations on Grid-connected Clusters using an Emulated WAN Environment," CCGRID'03, 2003.
3) 松田、石川、工藤、手塚「MPI 通信モデルに適した通信 API の設計と実装」、情報処理学会、SWOPP'93, 2003.
4) Atsuko Takefusa, Satoshi Matsuoka, Hirotaka Ogawa, Hidemoto Nakada, Hiromitsu Takagi, Mitsuhisa Sato, Satoshi Sekiguchi, and Umpei Nagashima, "Multi-client LAN/WAN Performance Analysis of Ninf: a High-Performance Global Computing," SC'97, IEEE, 1997.
5) "GridFTP: Universal Data Transfer for the Grid," The Globus Project, http://www.globus.org/datagrid/gridftp.html
6) GridMPI ホームページ、http://www.gridmpi.org
7) Sally Floyd, "HighSpeed TCP for Large Congestion Windows," IETF, INTERNET-DRAFT, draft-floyd-tcp-highspeed-03.txt, 29 June 2003.
8) IPMI, http://impi.nist.gov/IMPI/
9) OGSI, http://www.gridforum.org/ogsi-wg/
10) RDMA Consortium、http://www.rdmaconsortium.org/
11) Scalable TCP, http://www-lce.eng.cam.ac.uk/ctk21/scalable/
12) YAMPII, Yet Another MPI Implementation, http://www.ilab.is.s.u-tokyo.ac.jp/yampii/