グリッド環境におけるクラスタ間データ転送の評価

小倉章嗣[†1] 松岡聡[†2,†3] 中田秀基[†4]

大規模ストレージシステムを構築する際に、クラスタ計算機同士でグリッドを構成し、より大規模なストレージを構築する試みがなされている。グリッド上のストレージを構成するクラスタ間でデータ転送を行う場合、$RTT \times bandwidth$ の大きい環境では通信路の性能を生かし切れないことが問題になる。また、各クラスタノード間での通信のように同時に多対多で通信を行う場合は、複数の通信で通信路を共有しなければならないため、通信路のバンド幅以上にデータが流れ込んでしまうことが問題になる。本研究では、これらの問題を解決するために、同時に通信を行うノード数、ストライプ数等のパラメータについて、自動的に通信路に合ったパラメータを決定するシステムを提案する。本稿では、いくつかの環境を想定し、その上でクラスタ間データ転送のシミュレーションを行ない、それぞれの環境に応じた最適なパラメータについて考察する。

Evaluation of the inter-cluster data transfer on Grid environment

Shoji Ogura ,[†1] Satoshi Matsuoka [†2,†3]
and Hidetomo Nakada[†2,†4]

Large-scale storage systems to be utilized in DataGrid settings implemented by interconnecting and federating large-scale storage clusters is being proposed and constructed. On peer-to-peer data transfer between two large clusters, two major factors are involved: on one hand network pipes with large $RTT \times bandwidth$ typically become data-starved, resulting in bandwidth loss whereas when multiple nodes on the clusters attempt simultaneous transfer, the network pipe could become saturated, resulting in packet loss which again may result in bandwidth degradation in large $RTT \times bandwidth$ networks. By dynamically and automatically adjusting transfer parameters between the two clusters, such as the number of network nodes, number of socket stripes, we could achieve optimal bandwidth even when the network is under heavy contention. We have conducted several simulations on a few environments to evaluate and determine the appropriate transfer parameters for this purpose.

1. はじめに

グリッドを用いてクラスタ計算機同士を組み合わせることにより、大規模な計算が可能になっている。また、大規模なストレージシステムとしてのグリッドの利用も進みつつある。

例えば、CERN における LHC 加速器と ATLAS 検出器を用いた実験では、計 4 台の検出器が 1 台あたり年間約 1 ペタバイトのデータを生成する。このような大規模なデータを解析するために、グリッドを用いた解析システムがいくつか検討されている[1)]。これらのシステムでは遠隔地での大規模なデータを転送するために、高速なネットワークを利用する。また、大規模データを扱う例として、クラスタノード全体で動いている MPI プロセスのチェックポイントをとり、このチェックポイントファイルを他のクラスタへ移送する場面も考えられる。この際のチェックポイントファイルサイズは、クラスタ全体では物理メモリ × ノード数になる。そのような大容量のデータを転送する際には、高速なネットワークを有効に活用し、ファイルを高速に転送できることが望まれる。

しかし、グリッド環境においてデータ転送を行う場合、TCP のウィンドウサイズの上限により通信路上にデータが常に流れている状態にすることができず、高速なネットワーク基盤を有効に利用しているとはいえない。現在この問題点を改善する方法はいくつか提案されているが、それらは主に一対一での利用を想定している。多対多の通信でそれらの方法を用いた場合には、ウィンドウサイズが大きくなりすぎてしまい転

†1 東京工業大学 情報理工学研究科 数理・計算科学専攻
Tokyo Institute of Technology dept. of Mathematical and Computing Science
†2 東京工業大学学術国際センター
Tokyo Institute of Technology GSIC
†3 国立情報学研究所
National Institute of Information
†4 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology

送効率が落ちてしまう。そのためグリッド環境においてクラスタ間データ転送を行うことを想定し、ネットワークをより有効に使う方法を考える必要がある。本稿では様々なネットワーク環境での最適な方法を探るため、Network Simulator を用いてクラスタ間データ転送をシミュレートし、転送実験を行なった。シミュレーションの結果、リンクの静的なパラメータから最適なストライプ数を予測することは難しいため、動的にノード数、ストライプ数を変えるようなシステム、もしくは転送時にシミュレートを行ない、その結果によってパラメータを決定するシステムを考える必要がある。

2 章ではクラスタ間データ転送で生じる問題点について考察する。3 章では本研究で提案する手法の有効性を確認するために、シミュレーションを行なう。その際の設定等について述べる。4 章ではシミュレーションの結果を述べ、考察を行なう。5 章ではまとめと今後の課題を述べる。

2. クラスタ間データ転送の問題点

2.1 TCP

TCP は上位のアプリケーションに対して信頼性のある通信を提供する通信プロトコルである[2)]。TCP では複数のコネクションがリンクを共有し輻輳 (ふくそう) が起きるのを防ぐため、ウィンドウサイズを調整して転送を行うデータ量を決定する。このときウィンドウサイズを調整するアルゴリズムは輻輳制御アルゴリズムと呼ばれ、基本的にはネットワークが混んでいるときにはウィンドウサイズを減らし、空いているときにはウィンドウサイズを増やしていく。輻輳制御アルゴリズムにはいくつかのバージョンが存在し、実用化されているものとしては、Tahoe、Reno、Vegas、などがある。以下では現在最も使われている輻輳制御アルゴリズム TCP Reno について概略を述べる。

TCP Reno では 2 つのフェイズに分けて輻輳ウィンドウを変化させる。図 1 が TCP Reno での典型的な輻輳ウィンドウの変化である。転送開始時のスロースタートフェイズではパケットロスが起こるまで輻輳ウィンドウを指数的に増やしていく。パケットロスが起こるかウィンドウサイズがある一定値 (設定されていれば) になったところでウィンドウサイズを半分にし、輻輳回避フェイズに入る。輻輳回避フェイズではウィンドウサイズを 1 パケットずつ増やしていき、パケットロスが起きた時点でウィンドウサイズを半分にし輻輳回避フェイズを繰り返す。ロスしてしまったパケットは再送を行なうが、再送に失敗してタイムアウトしたらウィンドウサイズを最小にしスロースタートフェイズに入る。

2.2 TCP の問題点とその解決方法

高レイテンシ・高バンド幅な通信路において、TCP の問題点はウィンドウサイズの上限にある。初期の TCP のバージョンでは、ノードのバッファサイズなどを考慮しウィンドウサイズの上限は 64KB に制限されていた。そのため、高レイテンシ・高バンド幅な通信路では通信路を満たすだけの十分なパケットを送り出すことができず、通信路の性能を引き出せない。この問題を解決する 3 つの方法が知られている。

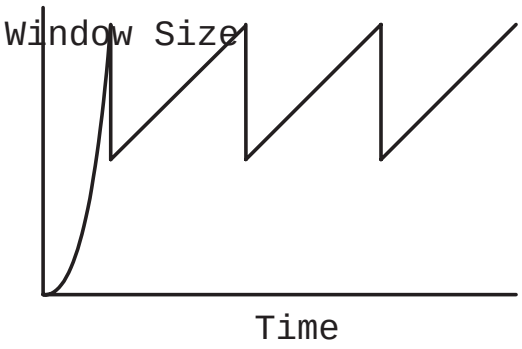

図 1 TCP Reno における輻輳ウィンドウの時間変化

1 つはウィンドウサイズの上限を変更する方法である。TCP ではウィンドウスケールオプションというものが定められており、ウィンドウスケールオプションを指定すると通信を行う両ノードでウィンドウサイズの上限をより大きな値にすることができる。しかし、この方法を用いるには両ノードでウィンドウスケールオプションに対応している必要があり、もし対応していないときには一般的にカーネルレベルでの変更が必要になる。また、最適なウィンドウサイズが大きい場合、スロースタートのフェイズでウインドウサイズが最適な値になるまでには時間がかかってしまう。2 つ目の方法としては、TCP を使わずに UDP を使って通信を行なうことである。UDP では輻輳制御を行なっていないため、ウィンドウサイズの上限を気にする必要はない。しかし、UDP で通信を行なう際にはアプリケーション自身で輻輳制御を行なう必要があり、現実的ではない。

3 つ目の方法は通信を行うノード間で複数のソケットを開き、データを分割して並列に通信を行う、ネットワークストライピングといわれる方法である。ウィンドウサイズの上限は一つのソケットに対するものなので、複数のソケットを同時に使うことで、その上限を克服できる。この方法ではシステムレベルでの変更が必要ないため、アプリケーションレベルでバンド幅の向上が期待できる。

2.3 クラスタ間転送における TCP の問題点

上記の方法は一対一での転送を想定しており、クラスタ間データ転送でそのまま用いると問題がある。Feng らは、グリッド環境上でクラスタ間データ転送を行なったときの問題点について言及している[3)]。この論文で

は NS[4] を使って LANL(Los Alamos National Laboratory) と SNL(Sandia National Laboratory) にあるクラスタ、及びその通信環境をシミュレートした。その上で、TCP Reno と TCP Vegas を使って転送をシミュレートし比較を行なった。その結果、同時に転送を行なうノード数を増やしたとき、TCP Vegas では輻輳制御アルゴリズムがうまく働きそれぞれのノードで十分な転送効率が実現できたものの、TCP Reno ではスロースタート時に輻輳ウィンドウが大きくなりすぎ、パケットロスが増えて転送効率が下ってしまうということが確認された。また、通信路のバンド幅が大きくなれば、この問題点はより顕著になる。

しかし、TCP Vegas を TCP Reno との混在環境で用いると TCP Vegas がスループットの面で非常に不利になることが知られており、[5],[6] TCP Vegas をグリッド環境において用いるのは現実的でない。

3. グリッド上クラスタ間データ転送の最適化手法提案

本研究ではクラスタ間データ転送の問題を改善するために、クラスタ間データ転送にネットワークストライピングを用い、その際に同時に転送を行うノード数、ストライプ数を最適な値に調整することを提案する。クラスタ間通信においては、通信路の途中でデータが溢れることが問題となる。ネットワークストライピングでは、主に一対一での利用を想定しており、クラスタ間通信に用いた場合には多くのコネクションで通信路を共有することになり、パケットロスの増加などが起きてしまう。同時に転送を行うノード数を制限することで、多くのノードを使用した際のパケットロスの増加を抑えることが期待できる。また、通信路の途中でデータが溢れることのない状況に対応するため、そのような状況ではネットワークストライピングのストライプ数を調整し、より最適なデータ転送を実現する。最終的な目標として、これら 2 つのパラメータ、ノード数、ストライプ数を、最適な値に自動チューニングするシステムを目指す。

ノード数、ストライプ数を最適な値にチューニングすることにより、スロースタートアルゴリズムの問題、ウィンドウサイズの大きさの問題が共に改善されることが期待できる。

4. クラスタ間データ転送のシミュレーション

本稿では、さまざまな環境における最適なノード数、ストライプ数を同定するために、ネットワークシミュレータを用いてグリッド環境におけるクラスタ間データ転送をシミュレートする。その上で、レイテンシ、パケットロス率などを変更することにより、さまざまなグリッド環境をシミュレートし、その環境における最適なノード数、ストライプ数を、実験を行い推定する。

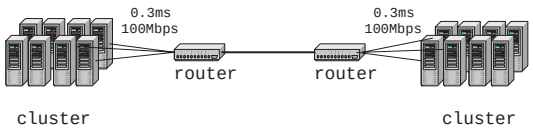

図 2 ネットワークトポロジ

TCP パケットの大きさ	1KByte
ウィンドウの最大値	64 パケット
転送するファイルサイズ	8Mbyte

表 1 各ノードのパラメータ

4.1 ネットワークシミュレータ

クラスタ間データ転送をシミュレートするために、NS(Network Simulator)[4] を用いた。NS は離散イベント型のネットワークシミュレータである。C++、OTcl で実装されており、新しいプロトコル、キューイングシステムの開発、検証に用いられることが多い。NS では様々なレイヤーのプロトコル (ユニキャスト、マルチキャストのルーティングプロトコル、 TCP、HTTP、FTP など) が実装されており、それらを用いて様々なネットワークトポロジをシミュレートできる。今回シミュレーションに用いた NS のバージョンは 2.1b8 である。

4.2 シミュレーション環境

クラスタ間データ転送は図 2 のような構成を想定し、その環境においてデータ転送をシミュレートした。クラスタノードのパラメータを表 1 に示す。それぞれのクラスタノードは他方のクラスタノードと一対一でネットワークストライピングを用いて転送を行う。転送するデータの大きさはそれぞれのノードで等しく、ノード毎のストライプ数も等しい。データを分割する際には、PSockets[7] と同じように、転送を行なう前にデータをストライプ数で等分割する。

ネットワークのレイテンシを 10ms ∼ 320ms、バンド幅を 1.5Mbps ∼ 1000Mbps に変化させて転送のシミュレーションを行なった。クラスタノードと近くのルータ間はレイテンシが 0.3ms、バンド幅は 100Mbps である。また、それぞれのノードは 8Mbyte のデータを転送するとし、シミュレーションは 100 秒で打ち切る。ルータにおけるキューイングシステムは FIFO を用いる。

5. シミュレーション結果と考察

5.1 一対一転送でのストライプ数による転送率の変化

まず、一対一転送における最適なストライプ数を考える。ノード数 1、中央のリンクのバンド幅が 100Mbps の状況で、転送するまでの時間をストライプ数を変化

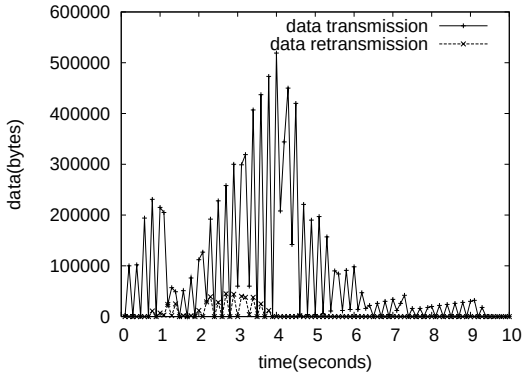

図 3 ストライプ数 52 での実際に転送したデータ量と再送したデータ量の時間変化

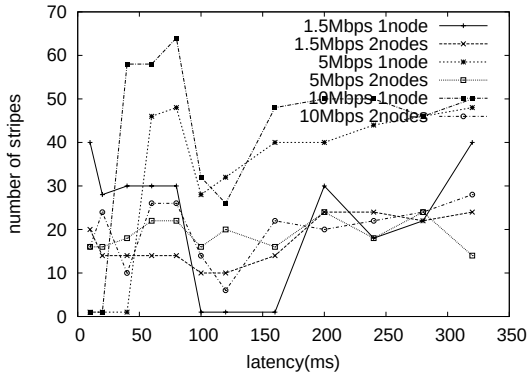

図 4 1.5Mbps、5Mbps、10Mbps のリンクにおけるレイテンシと最適ストライプ数の関係

させて計測した結果が図 9 である。データの転送時間は 1 つのノードの中の全てのストライプがデータ転送を終了するまでの時間とした。レイテンシが小さい環境を考えると、ストライプ数が少ない場合は通信路をデータで満たせないため、通信路の性能を十分に引き出せていない。また、ストライプ数が多い場合に時間がかかってしまうのは、輻輳制御アルゴリズムでのスロースタートフェイズにおいて、それぞれのストリームがウィンドウサイズを最適な値よりも大きくしすぎてしまうことが原因である。

図 3 は、ノード数 1、ストライプ数 52 で転送を行なった際の、実際に転送したデータ量と再送を行なったデータ量である。転送を開始してからしばらく経つと再送しているデータ量が増えている。これはウィンドウサイズが最適な値よりも大きくなりすぎ、パケットロスが増えていることを示している。

5.2 レイテンシと最適ストライプ数の関係

図 4 はバンド幅が 1.5Mbps、5Mbps、10Mbps のリンクにおけるレイテンシと最適ストライプ数の関係である。最適ストライプ数はデータ転送にかかった時間が最も短かかったストライプとし、複数ノードの場

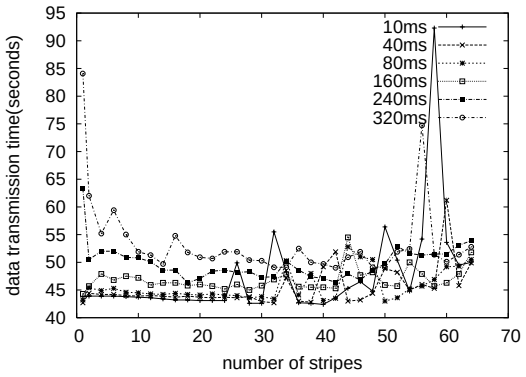

図 5 1.5Mbps、ノード数 1 でのストライプ数毎のデータ転送時間

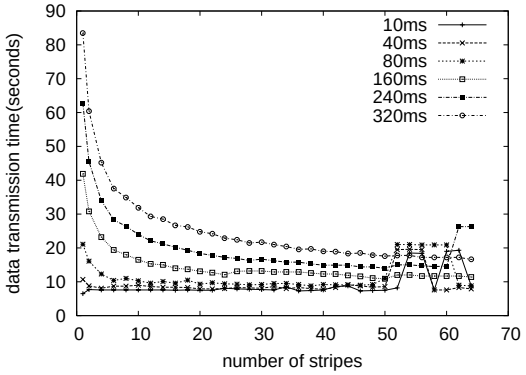

図 6 10Mbps、ノード数 1 でのストライプ数毎のデータ転送時間

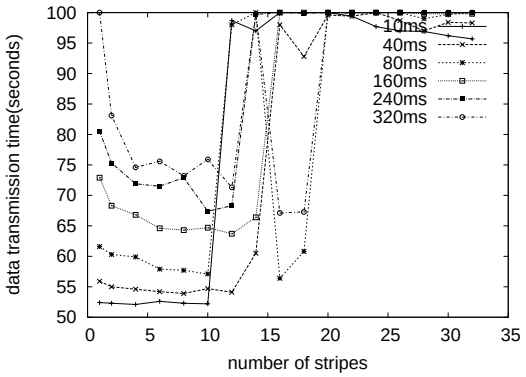

図 7 10Mbps、ノード数 8 でのストライプ数毎のデータ転送時間

合には、全てのノードのデータ転送が終わった時間が最短のストライプ数である。図 4 では、レイテンシが小さい場合に最適ストライプ数が大きく変化している。バンド幅が 1.5Mbps、6 つの異なるレイテンシのリンク上での、ストライプ毎のデータ転送時間を示した図 5 を見ると、レイテンシが小さいもの程ストライプ毎のデータ転送時間のばらつきが大きいことがわかる。また、レイテンシの大きいものでは、それぞれのストライプ毎のばらつきは小さい。このようにバンド

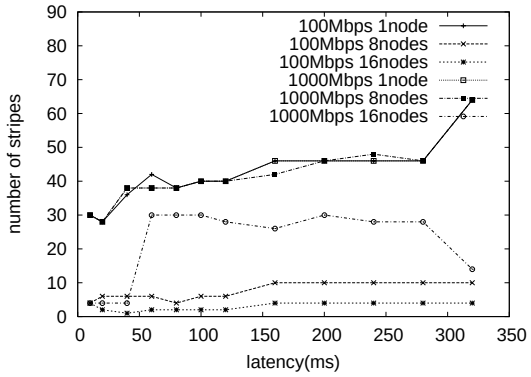

図 8 100Mbps、1000Mbps のリンクにおけるレイテンシと最適ストライプ数の関係

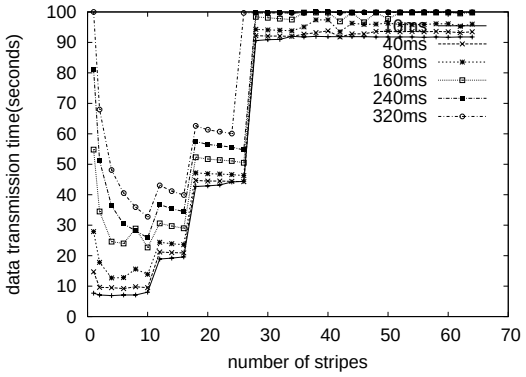

図 11 100Mbps、ノード数 8 でのストライプ数毎のデータ転送時間

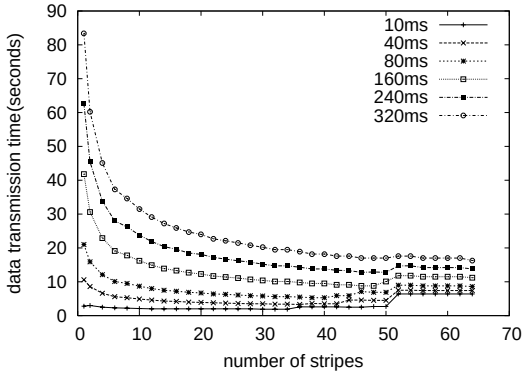

図 9 100Mbps、ノード数 1 でのストライプ数毎のデータ転送時間

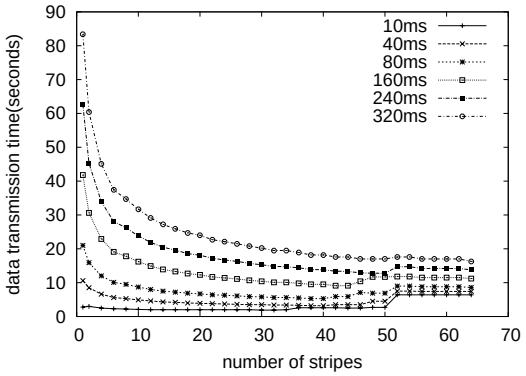

図 12 1000Mbps、ノード数 8 でのストライプ数毎のデータ転送時間

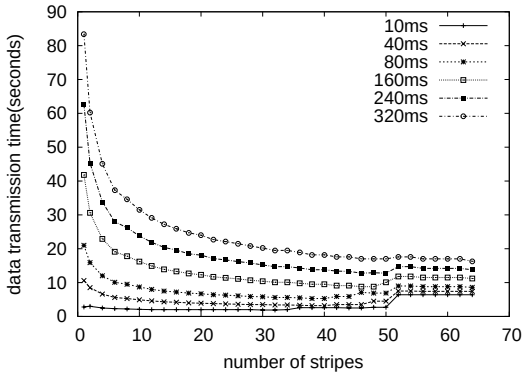

図 10 1000Mbps、ノード数 1 でのストライプ数毎のデータ転送時間

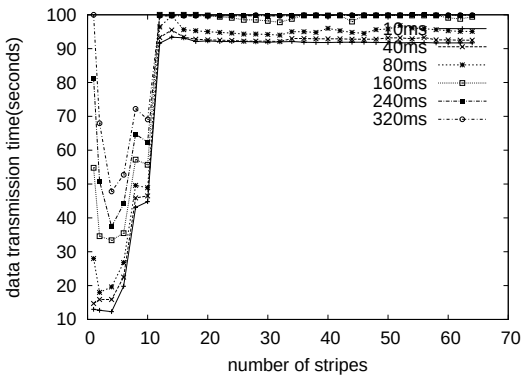

図 13 100Mbps、ノード数 16 でのストライプ数毎のデータ転送時間

幅が小さく、レイテンシも小さいようなリンクでは、ネットワークストライピングを行なわなくても通信路をデータで満たすことができるため、ネットワークストライピングを行なうとかえってデータ転送の効率が下がることがわかった。

100Mbps、1000Mbps のリンクの場合には、リンクのレイテンシが上がる毎に最適なストライプ数が上がっている。これは、リンクの $bandwidth \times latency$ の値が大きくなるにつれ、最適なストライプ数が大きくなっていることを示している。また、図 9、10、12 はほとんど同じ形のグラフになっているが、これらは通信路のボトルネックがどれもクラスタノード、ルータ間にあるために、TCP の振舞いが同じようになっ

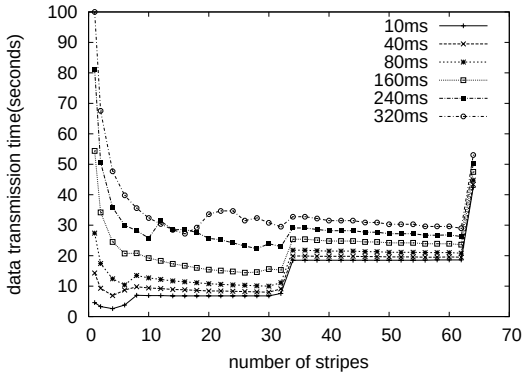

図 14 1000Mbps、ノード数 16 でのストライプ数毎のデータ転送時間

ていると考えられる。

また、図 11、13 を見ると、ノード数が 16 の場合のあるストライプ数のデータ転送時間は、ノード数が 8 の場合の 2 倍のストライプとほぼ等しい。これにより、通信路のボトルネックが変わらない状況では、最適なノード数とストライプ数の積は一定であるとも考えられる。

5.3 理論値との整合性

1 対 1 転送でネットワークストライピングを考えたとき、ラウンドトリップタイム RTT 、通信路のバンド幅 $bandwidth$ 、最大のウィンドウサイズ max_wnd_size、ストライプ数 $stripe$ とすると、パケットロスが無ければデータ転送量 は $stripe \times max\_wnd\_size$、通信路の容量 は $RTT \times bandwidth$ と考えられる。通信路を使いきるとしたとき、理論的に最適なストライプ数は

$$\frac{RTT \times bandwidth}{max\_wnd\_size}$$

と考えられる。例えば、100Mbps、160ms のリンクにおいて最適なストライプ数を考えると、ノード数 × ストライプ数が 6 であるはずである。しかし、実際の最適ストライプ数は 10 であり、この式には当てはまらない。また、この式が成り立つとすれば最適なストライプ数がレイテンシに比例するはずである。しかし、図 4、8 のグラフをみればわかるように、シミュレーションの結果を見る限り比例しているとは言えない。特にバンド幅が小さい場合には、近いストライプ数であってもデータ転送時間が大きく異なることもある。

このように、静的にわかる値を用いて最適なストライプ数を予測することは難しい。したがって、効率的なクラスタ間データ通信を行うためには、以下の 3 つの方法が考えられる。

- パケットロス率のような動的にわかるパラメータを用いて、動的にノード数、ストライプ数を変化させる。動的に変化させることにより、グリッド環境上での輻輳にも対応することが可能になる。
- 動的にシミュレーションを行ない、その結果によってノード数、ストライプ数を決定する。動的にシミュレーションを行なうため、大きなデータを保持する必要がない。
- 予めシミュレーションを行なって、レイテンシ、バンド幅にあった最適なストライプ数をデータベースにまとめておき、転送開始時にそのデータベースにより最適なパラメータを決定する。動的にシミュレーションを行なわないので、重い計算が必要ない。

6. まとめと今後の課題

本研究ではクラスタ間データ転送を想定し、同時に通信を行うノード数、ストライプ数等のパラメータについて、自動的に通信路に合ったパラメータを決定するシステムを提案した。本稿では、通信路のパラメータを変化させ、その上でクラスタ間データ転送のシミュレーションを行なった。その結果、データ転送にかかる時間はストライプ毎にばらつきが大きく、静的な値で最適なストライプ数、ノード数を予測するのは難しいことがわかった。その上で、効率的なデータ転送を行なうために、3 つの方法を提案した。今後の課題として、より多くの環境を想定したシミュレーションを行なうこと、効率的なデータ転送を行なうために考えられる 3 つのシステムを検討し、実際のシステムとして実装することが挙げられる。

参 考 文 献

1) 建部修見, 森田洋平, 松岡聡, 関口智嗣, 曽田哲之. ペタスケール広域分散データ解析のための grid datafarm アーキテクチャ. *HPCS2002*, 2002.
2) IETF homepage. http://www.ietf.org/.
3) W. Feng and P. Tinnakornsrisuphap. The failure of tcp in high-performance computational grids. *SC2000*, 2000.
4) ns. Network simulator. http://www-mash.cs.berkeley.edu/ns.
5) Jeonghoon Mo, La Richard J., Venkat Anantharam, and Jean Walrand. Analysis and comparison of TCP Reno and Vegas. *Proceedings of INFOCOM'99*, March 1999.
6) 倉田謙二, 長谷川剛, 村田正幸. Tcp reno と tcp vegas の混在環境における公平性の評価. 電子情報通信学会技術研究報告 (SSE99), March 1999.
7) H. Sivakumar, S. Bailey, and R. L. Grossman. PSockets:the case for application-level network striping for data intensive applications using high speed wide area networks. *SC2000*, 2000.