ATM ワークステーションクラスタにおける通信スループットの改善

大井拓哉　大澤範高　弓場敏嗣
電気通信大学大学院 情報システム学研究科

ATM ワークステーションクラスタ上の並列アプリケーションがノード間通信を行う場合、ノード間の処理能力の変動に起因するノード内通信バッファのオーバフローによるエラーと、プロトコル処理のオーバヘッドが通信スループットを著しく低下させる原因となっている。本稿では、 ATM ワークステーションクラスタ上に MPI ライブラリを実装する際、通信バッファのオーバフローが発生しない、受信ノード毎に最適な送信レートを求める通信方式を提案する。本方式にパケットヘッダとメモリコピーを用いない軽量プロトコルを組み合わせることにより、数メガバイトのメッセージ転送においても平均95Mbps以上、最大120Mbps程度のスループットを得ることができた。

Improvement of MPI Data Transfer Throughput on an ATM Workstation Cluster

Takuya Ooi　Noritaka Osawa　Toshitsugu Yuba
Graduate School of Information Systems, The University of Electro-Communications

In data transfer between nodes on an ATM workstation cluster, there is significant performance degradation caused by buffer overflow and protocol processing overhead. First, we propose a data transfer model for an ATM workstation cluster, and then by using the model, we show an optimization method of tuning data transfer rate of a sender node to that of a receiver node. Second, we realize "light-weight" protocol which eliminates packet headers and memory copies. Finally, we show that such performance improvement as average throughput of over 95Mbps is achieved by implementing the optimization technique and the light-weight protocol on a real ATM workstation cluster environment.

1.はじめに

複数のワークステーションを ATM ネットワークで接続したクラスタシステムによる、並列処理技術が注目されている。ワークステーションクラスタはネットワークによる通信遅延が単一コンピュータにおける並列処理に比べ大きいため、より粒度の大きな並列処理に適している。ATM は通信帯域が大きくネットワーク遅延も小いためワークステーションクラスタのネットワーク技術に適していると考えられる。

しかし、ATM を用いてワークステーションクラスタを構築する場合、受信ノードの処理能力を越える速度でデータを転送すると、受信ノードにおいて通信バッファのオーバフローが発生する。結果としてデータの廃棄が起こり、送受信ノード間の通信スループットが低下する原因となっている。一般に、クラスタを構成するノードが高性能ワークステーションであっても、受信側の負荷状況やコンテクストの切り替えなどの影響により、時間的に受信処理能力が変動する。その結果ATM から 100Mbps 以上の通信速度でバースト的に転送される大容量データを、受信側で取りこぼしなく受信し続けることは難しい。また、従来の低速ネットワークでは顕在化しなかったプロトコル処理のオーバヘッドもATM本来の性能の発揮を阻む大き

な原因となっている。

本稿では、ATM ワークステーションクラスタ用 MPI(Message Passing Interface)ライブラリ[1]の実装の過程において採用したこれら問題点の解決方法を示す。まず2節で、MPI ライブラリの概要とその通信の特徴を紹介する。3節で、ノードの受信処理能力と転送データサイズにあわせたバッファオーバフローを起こさない最適な送信レートを求める通信方式を提案する。4節で、ATM の特徴を生かしたパケットヘッダとメモリコピーを用いない軽量プロトコルを提案する。5節で、ATM ワークステーションクラスタ用 MPI ライブラリの実装環境を説明する。6節で、これらの機能を組み込んだ MPI プロトタイプ実装上での性能評価の結果を示す。7節で、6節の実験結果をもとに議論する。

2.MPI

MPI[1]は、分散メモリ型並列計算機用に MPI Forum によって標準化されたメッセージ通信用ライブラリインタフェースである。MPI の通信はノード間のバッファ内データの送受信を基本としている。すなわち有限長のデータ領域を指定して送受信を行うという特徴がある。また、非常にバースト性の強い通信でもある。MPI には一対一通信、グループ内マルチキャストなどの一対多通信、データの収集などの多対一通信、ノード間の同期などが仕様として定められている。

3.通信モデルによる送信レートの最適化

本節では通信バッファのオーバフローによるエラーのメカニズムと通信モデル、同モデルを使用した実際の制御方法を説明する。

3.1.通信バッファのオーバフロー

クラスタを構成するノードはハードウェア、ソフトウェア環境等によって決まる基本的な性能に加えて、その時々における受信ノードの負荷状況によってデータの受信能力が変動する。ATM コネクションから、その時点でのノードの受信処理能力を越えてデータを受け取った場合、データはドライバ等が管理するノード内の通信バッファに蓄えられる。しかし、バッファが一杯であった場合、バッファ量を越えた受信データは廃棄され、その結果パケットロスが発生する（図1参照）。ATM は信頼性が高くコネクション確立時に予約した帯域の範囲内で通信を行うのであればほとんどデータが失われないことが報告されている[5]。それにもかかわらず、パケットロスが発生し、スループットを大きく低下させることになる。

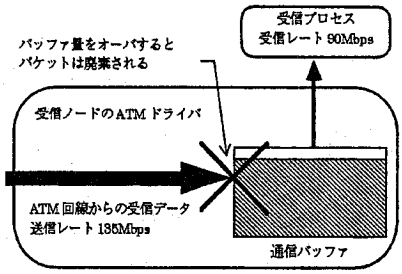

図 1: 通信バッファオーバフローの例

3.2.通信モデル

本稿で提案する通信モデルの目的は、ノードの受信処理能力と転送するデータサイズに合わせて、送信ノードの送信レートを最適化する。これにより、上記で説明した通信バッファのオーバフローによるパケットロスをなくし、スループットを改善することである。よって受信ノード内の通信バッファを中心にして、モデル化を行う。また通信バッファサイズよりも大きなデータ転送を最適化の対象とする。

ここで通信バッファのオーバフローを数式モデルによって表すことを試みる。受信処理能力 R で内部通信バッファサイズ B を持つノードに対して、サイズ M のデータを送信レート S で転送する場合。バッファのオーバフローによるデータ廃棄量 Berror は次の式で表せる。

$$Berror = \int_0^T S\,dt - \left(B + \int_0^T R\,dt\right) \quad (1)$$

ただし R：受信レート,
S：送信レート,
B：バッファサイズ,
M：データサイズ（M ＞ B),
T：送信ノードがサイズ M のデータを送信レート S で送信する場合の送信時間,

(1)式において、データの廃棄量 Berror を 0 としてその場合の送信レート S を求める。

$$0 = \int_0^T S\,dt - \left(B + \int_0^T R\,dt\right)$$

ここで $T = \frac{M}{S}$, $\int_0^T S\,dt = M$ から次式を得る。

$$M = B + RT = B + R\frac{M}{S}$$

$$\therefore \quad S = \frac{RM}{M-B} = \frac{R}{1-\frac{B}{M}} \qquad (2)$$

よって(2)式を用いることによって、データサイズとノードの受信処理能力に応じて最適な送信レートを求め、バッファのオーバフローをなくすことができる。

3.3.通信モデルによる制御方法

次に上記(2)式によって求めた通信モデル式を実際の通信へ適用し、送信レートを制御する方法を示す。

(2)式を用いるには、送信するデータサイズMの他に、受信ノードの受信処理能力R、受信ノード内通信バッファBを求める必要がある。以下にこれらのパラメータの導出方法を示す。MPIライブラリの内部では、下記のようにして事前に求めておいたパラメータR、Bとメッセージ毎に決まるデータサイズMを(2)式に代入することによって最適な送信レートSを求める。この求めた送信レートSで目的ノードにデータを送ることになる。

1. 受信ノード内通信バッファB
   ノードに搭載しているATM NIC(Network Interface Card)のRAM量、ドライバの実装方法によって固定的に決められ、ドライバの実装資料などから得ることができる。
2. 受信ノードの受信処理能力R
   ノードのハードウェアやソフトウェア環境、負荷状況、コンテクストの切り替えなどによる影響で、ノード毎に、また時間によって大きく変化し、正確に把握することは非常に難しい。そこで、事前にクラスタを構成するノードの種類毎に受信処理能力の統計を実測し、その期待値を受信処理能力Rとする。

4.軽量プロトコルの実装

前節で提案した通信モデルは結果としてノードの能力に合わせて送信レート抑制する。したがって基本的な通信においてATMの性能を最大限に発揮できるということを前提にしている。

しかしTCP/IP等の従来のプロトコルを用いる場合、信頼性が低い物理層を前提に設計され、また、個々のIPパケット毎に経路制御等を行うため数十バイトのパケットヘッダを必要とする[6]。これらのプロトコル処理とデータのオーバヘッドはATM本来のスループットを引き出す際の障害になっている。

そこで本稿のMPIライブラリにおいては以下に示すATMの特徴を生かした、より軽量なプロトコルを実装した。本プロトコルはCPCS(Common Part Convergence Sublayer)層の上に位置する(図2参照)。

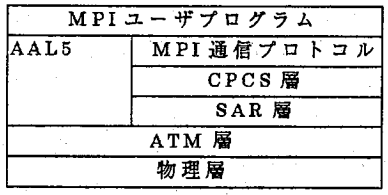

図2: MPIライブラリプロトコル階層図

プロトコルはユーザプログラムから受け取ったデータを、64KBを最大長とするCPCS-SDU(Service Data Unit)単位(以後単にパケットと呼ぶ)に分割しCPCS層に渡す。送受信の間にロスパケットがあれば、このパケットを単位として再送処理を行う。

以下に本軽量プロトコルを実装する上で利用したCPCS層の特徴をまとめる。

1. 送信した個々のパケットはそのままのサイズと形で受信される。
2. 送信したパケットの順番は受信時に入れ替わることはない。
3. 受信パケットにエラーが検出された場合、CPCS層内部において破棄される。よって上位プロトコルレイヤでのパケットのエラーチェックは必要ない。
4. コネクション確立時に経路が決定されるので個々のパケットには相手アドレスなどの経路情報は必要ない。

以上の特徴を利用し、本プロトコルのパケットにはIPパケットのヘッダのようなデータを一切持たない方針とした。

しかし、パケットの順序番号、パケットサイズ、転送データにおけるパケットの位置を示すオフセット情報などは通信において最低限必要な情報である。そこで前記の1.から4.の特徴を利用し、転送データをパケットに分割した際のパケットサイズそのものに順序番号、サイズ、オフセット情報などの情報を持たせることにした。主な通信処理を以下に示す。

1. 通信開始時に送信ノードから受信ノードにこれか

ら送信するトータルのデータサイズを開始メッセージとして通知する。

2. 受信ノードは送信ノードに 1.の開始メッセージを受信したことを報告する。
3. 送信ノードは送受信間であらかじめ決めておいた開始パケットサイズから送信を始め、パケット毎に一定バイトパケットサイズをデクリメントさせながら全てのデータを送信する。
4. パケットヘッダを持たないので送受信をユーザバッファ間でダイレクトに行う。
5. 受信ノードは受信したパケットのサイズから一意にパケットの順序番号とオフセット情報、ロスパケットの有無を識別する。
6. 受信ノードはパケット受信の最中は受信に専念する。途中ロスパケットが見つかった場合、受信処理が一通り済んでから再送要求を行う。
7. 受信ノードは全てのパケットの受信を完了したら、送信ノードに受信処理の完了を通知する。

図 3 に、開始パケットサイズを 8000Byte に設定し、56000Byte のデータを送信する場合のパケットサイズによるパケットの識別方法を示す。

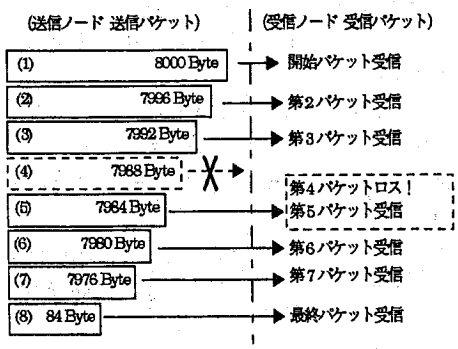

図 3: パケットサイズによる受信パケットの識別

ただし、データサイズが大きくなるとパケットサイズが小さくなりすぎる、その場合、データを何回かに分けて上記プロトコル処理を繰り返してやる必要がある。

以上により、通信の信頼性を維持しながらもパケットヘッダサイズ分のデータのオーバヘッドと、メモリコピーを伴うヘッダ処理オーバヘッドの大半を削減することができる。

5.実装環境

本ライブラリの実装において使用しているハードウェアとソフトウェアの環境を示す。帯域予約を含む ATM のコネクション設定に関する処理は SDK に含まれるコネクション設定用ライブラリを使用する。コネクションを確立した後は UNIX の標準システムコールを用いて基本的な送受信を行うことができる。これにより CPCS 層に直接アクセスすることができる（図 2 参照）。本環境で得られる ATM コネクションのユーザデータ最大転送レートは 135.63Mbps である。

1. ノード
   SUN-SS5 × 5 台
   MicroSparcII 70MHz, 32MB Memory, Solaris2.5
   SUN-SS20 × 3 台
   SuperSparc 75MHz, 48MB Memory, Solaris2.5
2. ATM スイッチ
   住友電気工業（株） SUMINET-3700SH
   SONET/SDH, 155Mbps, UTP5, 8 ポート
3. ATM-NIC（ネットワークカード）
   Efficient Networks, Inc. ENI-155s-U5-c
   155.52Mbps, Sbus adapter, 512KB Memory
   Software Development Kit (SDK) 3.2.3

6.性能評価

6.1.評価方法

まず通信モデルを使用する上で必要なノードの受信処理能力を求める。求め方は 135.6Mbps の ATM コネクションから 2MByte のデータを 1000 回バースト的に送信し、受信ノードでの受信レートの頻度を確立として求めた。図 4 に測定結果を頻度分布として表した。この結果より SS5 と SS20 の受信処理力の期待値 R はそれぞれ 52[Mbps], 101[Mbps]となった。

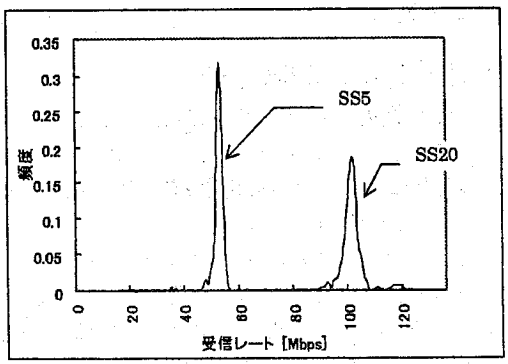

図 4: SS5 受信処理能力 頻度分布

バッファサイズBは ATM-NIC から得られる構成情報からSS5で64KB、 SS20で128KBとした。

次に上記で求めたR、B を使用し3節(2)式からデータサイズとノード毎に最適な送信レートを求める（表1参照）。表では計算の結果から得れた送信レートでMbps 以下の桁は四捨五入している。ATM の送信レートを越えるレートがモデル式から導かれた場合はATM の最大レートで送信を行う。

データ [Byte]	SS5 [Mbps]	SS20 [Mbps]
500,000	60	137
1,000,000	56	116
1,500,000	54	111
2,000,000	54	108
2,500,000	53	107
3,000,000	53	106
3,500,000	53	105
4,000,000	53	104
4,500,000	53	104
5,000,000	53	104

表 1: ノード毎の送信レートの最適化結果

MPI 関数プロタイプ実装上で、表1で求めた最適化した送信レートで送信した場合と、最適化を行わずATM の最大転送レート 135.63Mbps で送信した場合の通信スループットとパケットロス率を比較する。最適化の有無に関わらず4節で提案した軽量プロトコルを共に使用している。

スループットの計測方法は、受信ノードがデータの要求をだしてから、全てのデータの受信を完了してMPI の通信関数を終了するまでの時間でデータサイズを割ったものである。これには送達確認やロスパケットの再送処理も含まれている。パケットロス率はデータを構成する総パケット数に占めるロスパケットの比率を百分率で表している。なお今回示す図は各データサイズ毎に上記の計測をそれぞれ10 回行いその平均をグラフ化したものである。

6.2.測定結果

図5はSS20 同士、図6はSS20 からSS5 にデータを送信した場合、図7はSS5 からSS20 へ送信した場合である。

図5のSS20 同士の通信では、最適化なしの場合データサイズが 1MByte を越えたあたりからパケットロスが増え始め、そのための再送処理によりスループットが落ちている。最適化なしのパケットロス率は5Mbyte のデータ送信で約24%であったが、最適化後は、ほぼ0%にすることができた。スループットは最適化により平均20%程度向上できた。最適化にかかわらず500KByte あたりのデータで120Mbps 近くのスループットを得ることができた。データサイズ1.5MByte 未満では最適化を用いない方がスループットが高かった。

図6のSS20 からSS5 への送信では、最適化なしの場合90%以上のパケットロスが発生した。最適化後は20%程度に削減できた。スループットは約5倍向上し、SS5 の受信能力いっぱいの50Mbps 近くまで得ることができた。

図7のSS5 からSS20 への送信の場合、最適の有無による変化を確認することができなかった。双方ともデータサイズ2Mbyte 以降に70Mbps 近くのスループットで飽和した。

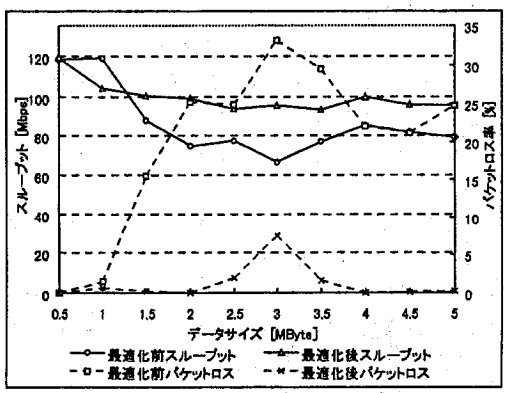

図 5: SS20 から SS20 への通信

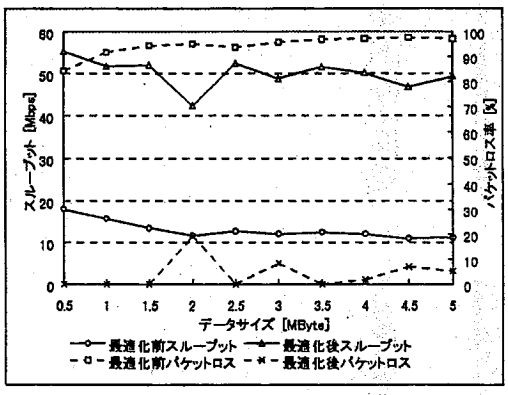

図 6: SS20 から SS5 への通信

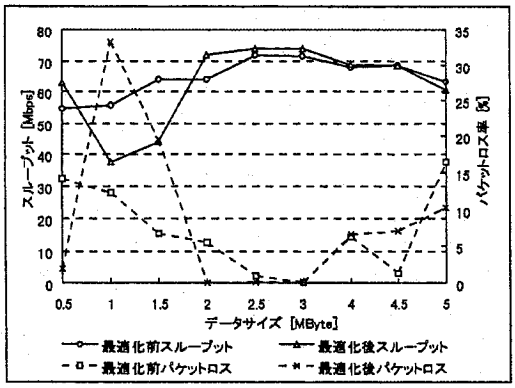

図 7: SS5 から SS20 への通信

7. 議論

実験結果から、ATM ワークステーションクラスタにおいて SS20 など高速のマシンからデータを送り出す場合、パケットロスの大半を削減することができ、スループットを大きく改善できることが確認できた。SS20 同士の通信で平均約 20%程度、SS20 から SS5 への送信では約 5 倍のスループットを改善することができた。

図 5 の送るデータサイズが 1.5MByte 未満では最適化を行わない方がスループットが高かった通信モデルがまだ完全にバッファリングと送・受信レートの関係を表現できてないことが解る。ノードの受信能力は負荷状況によって時間的に変化するため、その時点での受信能力を正確に評価するのは難しい。受信レートの期待値をもって受信処理能力とすることは妥当であると考えられる。今後はモデルに受信処理能力の時間的変動量、エラー率等のパラメータを加えさらに改善していく必要がある。

図 7 のように遅いマシンからデータを送り出す場合は、最適化の効果がないことが解った。また最適化の有無によらずスループットが 70Mbps あたりで飽和している。これは送信側の SS5 に ATM の転送レート以下の70Mbps程度のI/O 能力しかないためと思われる。従ってこのような処理能力をもつノードの組み合わせでは最適化は必要ないと言える。

本モデルはデータサイズと受信処理能力に応じて送信レートを最適化する。しかし、ATM にはコネクションの確立後に送信レートを動的に制御できる機能は提供されていない。現在のところ、最適化したコネクションを新たに張り直すしかない。今後は送信レートをプログラムから動的に制御する仕組みを考えていく必要がある。

図 5 では最適化にかかわらず SS20 で最大 120Mbps 程度、平均 95Mbps 以上のスループットを達成できた。これはパケットヘッダとメモリコピーを用いない軽量プロトコルによるところが大きいと思われる。

今後の課題は次の通りである。

1. 受信処理能力の時間的変動を考慮するなどの通信モデルの詳細化によるスループットの改善。
2. 本稿で提案した軽量プロトコルの有効性の評価。
3. MPI ライブラリへの機能の組み込み。

参考文献

[1] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. MPI The Complete Reference. The MIT Press, 1996.

[2] Cheng Chang Huang, and Philip K. McKinley. Communication issues in parallel computing across ATM networks. Technical Report MSU-CPS-94-34. Department of Computer Science, Michigan State University, 1994.

[3] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. Vol.22, No.6, Parallel Computing, pp 789-828, 1996.

[4] Martin De Prycker. Asynchronous Transfer Mode Solution for Broadband ISDN, Prentice Hall PTR, 1995.

[5] Yasunori Matsui, Shuusuke Utsui, Daichi Funato, Yoshiyuki Nakamura, Tomio Narita, Tatsumi Hosokawa, and Hideyuki Tokuda. A comparison of transmission characteristics between ATM-LAN and ATM-WAN environment.

[6] Willibald A. Doeringer, Doug Dykeman, Matthias Kaiserswerth, Bernd Werner Meister, Harry Rudin, and Robin Williamson. A survey of Light-Weight transport protocols for High-Speed networks. Vol.38, No.11, IEEE Transactions on communications, 1990.