社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－HPC－103（31）
2005／8／5

広域メッセージパッシングシステム用の遅延を考慮した接続管理

斎藤秀雄[†]　田浦健次朗[†]　近山隆[††]

本稿では，広域メッセージパッシングシステム用に，遅延を考慮して接続管理を行う方法を提案する．提案手法では，あるプロセスに直接接続しなくても他のプロセスを中継すれば直接接続した場合の遅延の k 倍以内の遅延で通信が行える場合は，そのプロセスには接続しない．3 つのクラスタに分散された 96 プロセッサにおいて行った実験では，$k = 1.2$ とした場合，クラスタ内では可能な接続のほとんど (96.2%) が張られた一方，クラスタ間では可能な接続の 19.2%のみ張られた．また，提案手法を用いて接続を張るのにかかった時間は，2 つのクラスタに分散された 100 プロセッサを用いた場合，1.83 秒であった．さらに，k を変化させることによって張られる接続の数を制御できることを確認した．

Latency-Aware Connection Management for Wide-Area Message Passing Systems

Hideo Saito,[†] Kenjiro Taura[†] and Takashi Chikayama[††]

We propose a method for wide-area message passing systems to perform latency-aware connection management. In our proposal, a processor does not connect directly to another processor if it can communicate with that processor via a different processor with a latency that is within k times the latency of the direct connection. In an experiment using 96 processors distributed over 3 clusters, when k was 1.2, most (96.2%) of the possible connections were established within each cluster, while just 19.2% of the possible connections were established between the clusters. The time to establish connections using our proposal was 1.83 seconds with 100 processors distributed between 2 clusters. Moreover, we were able to control the number of established connections by using different values of k.

1. はじめに

近年，WAN (Wide-Area Network) の整備に伴い，複数の LAN (Local-Area Network) にまたがる計算資源を用いてメッセージパッシングを行う機会が増えてきた．これまでに複数の LAN に分散された多数のプロセッサを用いて効率良く集合通信[5] やロードバランシング[9] を行う方法は色々と提案されてきたが，それらのプロセッサ間の接続を管理する方法に関しては依然改善の余地がある．

従来の LAN 用のメッセージパッシングシステムは，高い性能を出すために全プロセッサ間の接続を直接張る[6),7)]．グリッド用のメッセージパッシングシステムの多くも同様に接続を張れる全プロセッサ間の接続を直接張る (ファイアウォールや NAT (Network Address Translation) のため直接張れないところもある) が[4),11)]，グリッドで多数のプロセッサで全対全の接続を張ると様々な問題が生じる．まず，TCP を用いる場合，`select` システムコールの実行時間がソケットの数に比例するため，多数の接続を維持するとメッセージ送受信のオーバヘッドが大きくなってしまう．また，TCP では各ソケットに送信・受信用のバッファを割り当てなければならないため，多数の接続を維持するためにはメモリを多く要する．さらに，多数の接続を同時に張ろうとするとスイッチでパケットロスが発生するため，全ての接続を張るのには時間がかかる．

一部のグリッド用のメッセージパッシングシステムでは，接続数を減らすために全対全で接続を張るのを LAN 内に限って，LAN 間ではゲートウェイ間やユーザが指定した少数のプロセッサ間のみ接続を張るということが行われてきた[1)~3),8)]．単純なネットワークにおいてはこのような手法で容易に接続数を減らして高い性能を出すことができる．しかし，複雑なネットワークにおいては LAN 内と LAN 間を区別して接続を張るだけでは性能は出ず，性能が出るような接続の張り方を手動で指定するのは非常に困難である．たとえば，VLAN (Virtual LAN) が用いられている場合，

† 東京大学大学院情報理工学系研究科電子情報学専攻
†† 東京大学大学院新領域創成科学研究科基盤情報学専攻

S: set of unprocessed end points

1: $S = \{\text{all end points}\}$
2: **while** $S \neq \phi$:
3: select r randomly from S
4: **if** *accepted(r)* **or** *goodroute(r)*:
5: /* don't need to connect */
6: **else**:
7: connect to r
8: $S = S$ - $\{r\}$

図 1 各プロセッサが従う接続アルゴリズム
Fig. 1 Connection algorithm followed by each processor

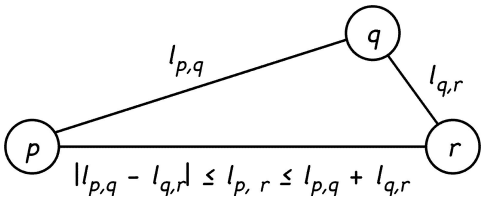

図 2 プロセス p とプロセス q の間の遅延が $l_{p,q}$ でプロセス q とプロセス r の間の遅延が $l_{q,r}$ の場合，プロセス p とプロセス r の間の遅延 $l_{p,r}$ は $|\ l_{p,q} - l_{q,r}\ | \leq l_{p,r} \leq l_{p,q} + l_{q,r}$ を満たすと仮定する．
Fig. 2 When the latency between processor p and processor q is $l_{p,q}$ and the latency between processor q and processor r is $l_{q,r}$, we assume that $l_{p,r}$, the latency between processor p and processor r, satisfies$|\ l_{p,q} - l_{q,r}\ | \leq l_{p,r} \leq l_{p,q} + l_{q,r}$.

IP アドレス上同じ LAN にあるプロセッサが物理的には遠いという可能性がある．また，一つの LAN 内のプロセッサの多いが大きい場合は，LAN 内の全プロセッサを全対全で接続するのではなく，近いスイッチに接続されているプロセッサのみ全対全で接続し，遠いスイッチに接続されているプロセッサとは少数の接続を張るということが必要である．逆に，一つの LAN 内のプロセッサの数が少ない場合は，他の LAN のプロセッサと比較的多くの接続を張った方が性能が上がる可能性がある．

そこで本稿では，広域メッセージパッシングシステムにおいて手動の設定なしで接続管理を行う手法を提案する．提案手法では，各プロセスが自律的に他のプロセスとの RTT を測定し，それを考慮してどのプロセスと接続を張るかを決定する．プロセスの参加・脱退があるようなシステムにおいて接続管理を行うことも視野には入れているが，現時点では計算に参加するプロセスはアプリケーション起動時に分かっているものとする．

以下，2 で我々の提案する接続管理の実現方法を説明し，3 で評価実験の結果を示す．4 章で関連研究について述べる．最後に，5 章でまとめを行う．

2. 提 案 手 法

我々の提案する接続管理では，各プロセスはアプリケーション起動時に全プロセスの接続先 (IP アドレスとポート番号) が与えられているとし，その中から実際に接続するプロセスを選択することによって自律的に接続数を抑制する．ただし，全プロセスが互いに接続できる必要はなく，中継によって全プロセスを連結にすることが可能であれば良い．つまり，各プロセスはファイアウォールの後ろにあったり，NAT でプライベートアドレスしか持っていなくてもかまわない．

あるプロセスに接続するかどうかは，そのプロセスに接続した場合としなかった場合のそのプロセスとの通信遅延の比によって判断する．接続しなくても他のプロセスを中継すれば k 倍以内の遅延で通信が行える場合は，接続しない．ただし，k はユーザによって与えられるパラメータである．また，中継オーバヘッドは無視する．

図 1 に各プロセスが他のプロセスと接続を張るにあたって従うアルゴリズムを示す．ただし，*accepted(r)* は，r から既に接続を受け付けている場合に TRUE を返し，それ以外の場合は FALSE を返す．また，*goodroute(r)* は，r に接続しなくても既に接続している他のプロセスを経由すれば r 接続した場合の k 倍以内の遅延で r と通信が行える場合に TRUE を返し，それ以外の場合は FALSE を返す．

goodroute(r) はその中で r と直接通信を行った場合の遅延を用いるが，r にはまだ接続していないのでこの値は分からない．そこで，次のように既に接続しているプロセスとの遅延を基にその値を見積もる．

図 2 に示すように，プロセス p とプロセス q の間の遅延が $l_{p,q}$，プロセス q とプロセス r の間の遅延が $l_{q,r}$ の場合，プロセス p とプロセス r の遅延は $l_{p,r}$ は $|\ l_{p,q} - l_{q,r}\ | \leq l_{p,r} \leq l_{p,q} + l_{q,r}$ であるとする．中継オーバヘッドは無視するので，p が q を経由して r と通信を行った場合の遅延と p が r と直接通信を行った場合の比は最大で $\frac{l_{p,q}+l_{q,r}}{|l_{p,q}-l_{q,r}|}$ 倍である．したがって，式 1 を満たすような q が存在する場合，*goodroute(r)* は TRUE を返す．

$$\frac{l_{p,q} + l_{q,r}}{|l_{p,q} - l_{q,r}|} < k \tag{1}$$

接続したプロセスとは 1 バイトのメッセージを用いて ping-pong を行い，遅延を測定する．我々の提案手法ではプロセス間で全対全で接続を張るわけではない

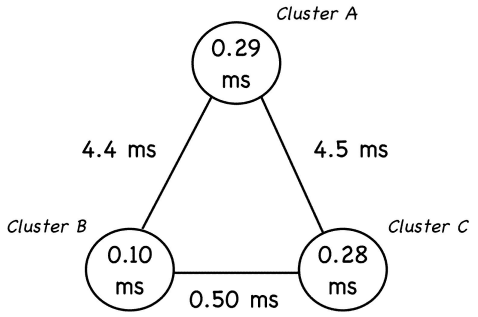

図 3 各クラスタ内の RTT(丸) と各クラスタ間の RTT(辺)
Fig. 3 RTTs within each cluster (circle) and between each cluster (edge)

表 1 $k = 1.2$ とした場合に各クラスタ内と各クラスタ間で張られた接続数
Table 1 Number of connections established within and between each cluster with $k = 1.2$

Cluster A	Cluster B	Cluster C	Total
481/496	468/496	482/496	1431/1488
97.0%	94.4%	97.2%	96.2%
A-B	B-C	C-A	Total
168/1024	227/1024	191/1024	586/3072
16.4%	22.2%	18.7%	19.1%

のでルーティングを行う必要があるが，そのメトリックにはこのようにして求めた遅延を用いる．

3. 評価実験

本章では，提案手法を評価するために行った評価実験について述べる．3.1 節で接続数に関する実験について述べ，3.2 節で接続を張るのに要する時間に関する実験について述べる．また，**図 3** に実験に用いたプロセッサの LAN 内と LAN 間の RTT を ping コマンドを測定した結果を図 3 示す．

3.1 接続数

最初の実験では，k を 1.2 とし，3 つのクラスタに分散された 96 プロセッサ (図 3 の 3 つのクラスタにそれぞれ 32 プロセッサにおいてどのような接続が張られるかを調査した．

表 1 に結果を示す．クラスタ内では中継を行うと通信遅延が 2 倍になってしまうため，可能な接続のほとんど (96.2%) が張られた．一方，クラスタ間では同一クラスタのプロセスを中継しても通信遅延は大きくは変化しないため，可能な接続の 19.1%しか張られなかった．全体として，可能な接続の 44.2% (2017/4560) が張られた．

次の実験では，2 つのクラスタ (図 3 の Cluster B と Cluster C) に分散された 128 プロセッサ (各クラスタに 64 プロセッサ) 用い，k を変化させた時にどのような接続が張られるかを調査した．

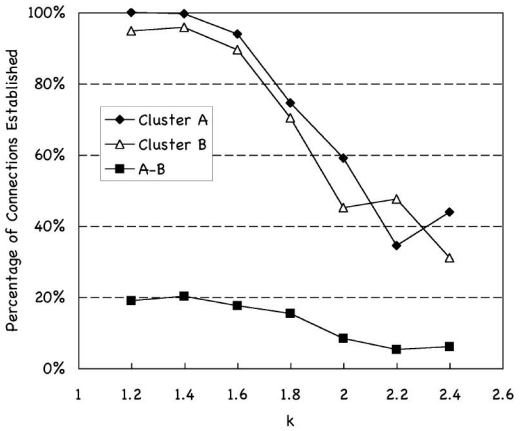

図 4 k を変化させた時に各クラスタ内とクラスタ間で張られた接続の割合
Fig. 4 Percentage of connections established within and between each cluster for different values of k

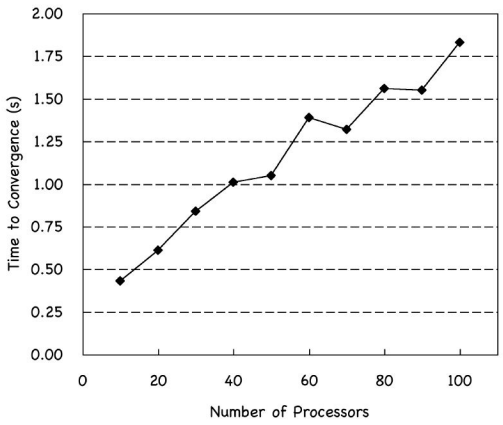

図 5 接続を張るのに要した時間
Fig. 5 Time to establish connections

図 4 に結果を示す．k が小さい間は多くの接続が張られるが，k が大きくなると張られる接続は減り，クラスタ内でも $k \geq 2.2$ では張られる接続より張られない接続の方が多かった．

このように，k を変化させることによって張られる接続の数を制御することができる．これを用いれば，大きなクラスタにおいて全対全では接続を張らずに近いスイッチに接続されているプロセスを中心に張るなどといったこともできる．

3.2 接続を張るのに要する時間

最後の実験では，2 つのクラスタ (図 3 の Cluster

A と Cluster B) に分散されたプロセッサを用い，接続を張るのにかかる時間を測定した．k は 1.2 とし，2 つのクラスタではそれぞれ同じ数のプロセッサを用いた．

図 5 に結果を示す．100 プロセッサ (それぞれのクラスタに 50 プロセッサ) を用いても接続は 1.83 秒で張り終わった．今後，我々の手法を用いて接続を張るのにかかる時間と他の手法を用いて接続を張るのにかかる時間を比較しなければならない．

4. 関連研究

4.1 グリッド用メッセージパッシングシステム

近年，グリッド環境で高性能な並列計算を行うためのメッセージパッシングシステムが数多く提案・実装されている[4),11)]．しかし，我々の提案手法のように手動の設定なしで接続管理を行うものはない．

MPICH-G2[4)]，GridMPI[11)] は全プロセッサで全対全の接続を張る．そのため，プロセッサ数が大きくなると `select` システムコールのオーバヘッドが大きくなり，メッセージ送受信のオーバヘッドが大きくなってしまう．また，多数のソケットを維持するために多くのメモリを要する．

PACX-MPI[2)]，Stampi[3)]，MPICH/MADIII[1)] では，ユーザがプロキシを設定し，クラスタ間の通信はこのプロキシを中継して行う．プロキシを用いることによって接続数を減らせるが，ユーザが設定を手動で行わなければならなく，煩雑である．また，用いられるネットワークやプロセッサに応じてクラスタ内でも接続数を減らしたり，逆にクラスタ間で複数接続を張ったりなどといったことはできない．

Phoenix[8)] では，全プロセッサで全対全の接続を張るモードと，ユーザがどのプロセッサ間で接続を張るかを詳細に指定するモードがある．プロセッサ数が多い場合，前者は接続数が多くなってしまい，後者は適切な設定が困難である．

4.2 接続を動的に制御する機構

金田らは，ノード間の RTT とスループットを測定することによってスイッチ間の結合関係を推定して自分と同じスイッチに属するノードを中心に接続を張るという手法を提案している[10)]．この研究は測定を行ってどのプロセス間で接続を張るかを決定するという点では我々の提案手法と似ているが，スイッチ間の結合関係を詳細に推定するため，我々の提案手法より著しく時間がかかってしまう．

5. おわりに

本稿では，広域メッセージパッシングシステム用に，遅延を考慮して接続管理を行う方法について述べた．

評価実験では，3 つのクラスタに分散された 96 プロセッサにおいて $k = 1.2$ として実験を行ったところ，クラスタ内では可能な接続のほとんど (96.2%) が張られた一方，クラスタ間では可能な接続の 19.2%のみ張られた．また，k を変化させることによって張られる接続の数を制御できることを確認した．さらに，2 つのクラスタに分散された 100 プロセッサにおいて，我々の提案手法を用いて接続を張るのには 1.83 秒かかった．

今後の課題としては，以下が挙げられる．

通信性能の評価
本稿では，提案手法を用いることによって遅延を考慮して選択的に接続を張ることができることを示した．しかし，提案手法の有用性を示すためには，接続の数が実際に問題になるような大規模な環境で実験を行い，性能を評価する必要がある．特に，MPICH などの MPI ライブラリを提案手法を用いて接続を張るように変更して，実アプリケーションの性能を評価することも目指す．

接続を張るのに要する時間の比較
我々の提案手法の有用性を示すためには，提案手法を用いて接続を張るのに時間がかからないことを示す必要がある．そのために，中継を行わずに全対全で接続を張る手法やプロキシを設定して中継する手法などと，接続を張るのにかかる時間を比較する必要がある．

中継オーバヘッドの考慮
本稿のアルゴリズムでは中継オーバヘッドは無視したが，実際には中継オーバヘッドは無視できない．中継するプロセスが維持している接続の数を基に `select` システムコールの実行時間を求め，中継オーバヘッドを求めるということが考えられる．

複雑なネットワークにおける実験
2 つや 3 つではなく多数のクラスタを用いた実験を行って，提案手法のスケーラビリティを示す必要がある．また，大きなクラスタにおいて k を大きくすることによってクラスタ内でも接続制御を行うことを目指す．複雑なネットワークでは，k を大きくした方が有理なプロセスと k を小さくした方が有理なプロセスが混在する可能性があるので，そういった状況に対応できるように提案手法を改良する必要もあるかもしれない．

参 考 文 献

1) Aumage, O. and Mercier, G.: MPICH/MADIII: A Cluster of Clusters Enabled MPI Implementation, *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 26–33 (2003).
2) Gabriel, E., Resch, M., Beisel, T. and Keller, R.: Distributed Computing in a Heteregeneous Computing Environment, *Proceedings of the 5th European PVM/MPI User's Group Meeting*, pp. 180–187 (1998).
3) Imamura, T., Tsujita, Y., Koide, K. and Takemiya, H.: An Architecture of Stampi: MPI Library on a Cluster of Parallel Computers, *Proceedings of the 7th European PVM/MPI Users' Group Meeting*, pp. 200–207 (2000).
4) Karonis, N. T., Toonen, B. and Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface, *Journal of Parallel and Distributed Computing*, Vol. 63, No. 5, pp. 551–563 (2003).
5) Kielmann, T., Hofman, R. F. H., Bal, H. E., Plaat, A. and Bhoedjang, R. A. F.: MAGPIE: MPI's Collective Communication Operations for Clustered Wide Area Systems, *Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 131–140 (1999).
6) LAM/MPI Parallel Computing: http://www.lam-mpi.org.
7) MPICH-A Portable Implementation of MPI: http://www-unix.mcs.anl.gov/mpi/mpich.
8) Taura, K., Endo, T., Kaneda, K. and Yonezawa, A.: Phoenix: A Parallel Programming Model for Accommodating Dynamically Joining/Leaving Resources, *Proceedings of the Ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 216–229 (2003).
9) van Nieuwpoort, R. V., Kielmann, T. and Bal, H. E.: Efficient Load Balancing for Wide-Area Divide-and-Conquer Applications, *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 34–43 (2001).
10) 金田憲二, 田浦健次朗, 米澤明憲: 接続を動的に制御するメッセージパッシングシステム, *SWoPP'04*, pp. 241–246 (2004).
11) 松田元彦, 石川裕, 鐘尾宜隆, 枝元真彦, 岡崎史裕, 工藤知宏, 児玉祐悦, 手塚宏史: GridMPI の性能評価, *SWoPP'04*, pp. 127–132 (2004).