Vol.2010-ARC-192 No.5
Vol.2010-HPC-128 No.5
2010/12/16

Fat-Tree 構成 InfiniBand ネットワークにおける競合回避手法の提案

成瀬 彰[†1] 中島 耕太[†1]
住元 真司[†1] 久門 耕一[†1]

本稿では、Fat-tree 構成の InfiniBand(IB) ネットワークにおける全対全通信時の Hot-spot 発生を回避する手法を提案・評価する。

Fat-tree 構成の IB ネットワークでは、適切に使用する計算ノードを選択しないと、Hot-spot 発生により実行通信バンド幅が低下する。本稿では、任意の計算ノード割当てにおいて、全対全通信時の Hot-spot 発生を回避する手法を提案する。提案手法は、各計算ノードに複数の LID(Local Identifier) を割り当てる手法の一種である。全対全通信時の典型的な通信パターンであるシフト通信パターンに着目し、各計算ノード対に生成されるコネクション毎に、使用 LID を適切に選択することで、Hot-spot 発生を回避する。

提案手法を OpenMPI で実装し、6-ary-2-tree トポロジーの Fat-tree 構成 IB ネットワークに接続した 30 ノードの PC クラスタシステムで性能を評価した。任意 16 ノードによる全対全通信性能を測定した結果、提案手法により全対全通信時の Hot-spot 発生を完全に回避できることを確認した。

Proposal of Hot-spot Avoidance Technique on All-to-all Communication in Fat-tree based InfiniBand Networks

Akira NARUSE,[†1] Kohta NAKASHIMA,[†1]
Shinji SUMIMOTO[†1] and Kouichi KUMON[†1]

In this paper, we propose and evaluate a method to avoid a hot-spot occurrence on all-to-all communication in fat-tree based InfiniBand networks.

An effective network bandwidth can degrade because of a hot-spot occurrence in fat-tree based networks, unless proper combination of nodes is assigned to a MPI job. Our proposed method utilizes LMC (LID Mask Control) feature of InfiniBand and assigns multiple LIDs (Local Identifier) to each HCA (Host Channel Adapter) in a compute node. Then, it selects the best LID to use for every source/destination pair to avoids a hot-spot occurrence in view of the characteristics of all-to-all communication pattern.

The proposed method is designed and implemented into OpenMPI, and evaluated on real InfiniBand cluster system consists of 30 compute nodes and 6-ary-2-tree topology networks. All-to-all benchmarks show that our proposed method can avoid a hot-spot occurrence at any time even when arbitrary combination of 16-nodes is assigned to a MPI job.

1. はじめに

近年、高いネットワーク性能を必要とする HPC システムでは、高バンド幅・低遅延な InfiniBand(IB)[2] と FBB(Fully Bisectional Bandwidth) 特性を持つ完全 Fat-tree トポロジーを組み合わせた Fat-tree IB ネットワークが良く採用されている。IB の問題はルーティングである。IB は static ルーティングであり adaptive ルーティングをサポートしていない。そのため、FBB 特性を持つ完全 Fat-tree トポロジーであっても、使用する計算ノードを適切に選択しないと、通信中に特定リンクに通信負荷が集中して (Hot-spot 発生)、実効ネットワークバンド幅が低下する問題がある。

本稿では、Fat-tree 構成の IB ネットワーク上で、全対全通信時の Hot-spot 発生を回避する手法を提案する。提案手法は、各計算ノードに複数の LID(Local Identifier) を割当てる手法の一種であるが、複数 LID を同時使用してマルチパスを形成し、パケットを複数パスに分散させることで Hot-spot 発生による通信性能低下を軽減する手法ではない。全対全通信の典型的な通信パターンであるシフト通信パターンに着目し、各計算ノード対の間に形成されるコネクション毎に適切な LID を選択・使用することで、Hot-spot 発生を回避する手法である。

提案手法を 6-ary-2-tree トポロジーの Fat-tree 構成 IB ネットワークに接続した 30 ノードの PC クラスタシステム上で評価した。全 30 ノードから任意 16 ノードを選択したときの全対全通信性能を測定した結果、提案手法では、安定して高い実効ネットワークバンド幅が得られており、Hot-spot 発生を完全に回避できることを確認した。

以下、2 章で提案手法の理解に必要な技術背景を説明し、3 章で Fat-tree IB ネットワーク上で Hot-spot が発生する仕組みを説明し、4 章で Hot-spot 発生を回避する手法を提案

†1 富士通研究所
Fujitsu Laboratories

Vol.2010-ARC-192 No.5
Vol.2010-HPC-128 No.5
2010/12/16

し、5 章で提案手法を実機上で評価し、6 章で関連研究に言及し、7 章でまとめる。

2. 技術背景

2.1 InfiniBand

InfiniBand(IB)[2] は低遅延かつ高バンド幅なネットワークであり、HPC クラスタ用のネットワークとして広く普及している。現在では 40Gbps のピークバンド幅を持つ 4xQDR が利用可能で、今後も継続的なバンド幅向上が見込まれている。

IB のルーティング方式は static ルーティングである。各 IB スイッチはそれぞれ転送先テーブルを持っており、各パケットのヘッダに記載されている転送先 LID(Local Identifier) と転送先テーブルから、パケット転送先ポートが一意に決まる仕組みとなっている。各 IB スイッチの転送先テーブルの設定は、SM(Subnet Manager) の仕事である。

通常、各 HCA(Host Channel Adapter) に割り当てられる LID は 1 つであるが、IB の LMC(LID Mask Control) 機能を利用することで、各 HCA に最多で 128 個の LID を割り当てることができる。しかし、IB サブネット内で使用可能な LID 数は合計 49,152 個であり、仮に、各 HCA に対して 128 個の LID を割り当てると、同一 IB サブネットに接続できる HCA 数は 384 個に制限される。各 HCA への割当て LID 数と接続可能な HCA 数はトレードオフの関係にあり、大規模システムでは注意が必要である。なお、LMC 機能は、任意サーバ間のマルチパス通信を実現する手段として使われることが多い[10)16)]。

2.2 Fat-tree (K-ary-N-tree)

FBB 特性を持つ完全 Fat-tree(K-ary-N-tree) は、IB 接続の PC クラスタシステムで良く使われるトポロジーである[1)]。理論上、システム全体で、任意サーバ間の通信を同時に実行しても、リンク競合ゼロで通信できる経路が存在する。しかし、動的に経路を変更できない場合、通信パターン次第でリンク競合が発生して通信性能が低下する。動的に経路を変更できない IB ネットワークでは、MPI ジョブに割当てるノードの組み合わせや通信パターン次第で、大幅に通信性能が低下することが報告されている[10)]。

Fat-tree ネットワークのステージ数 (N) は、ネットワークに接続するノード数と (システム規模)、ネットワークの基本構成要素であるスイッチチップのポート数で決まる。IB スイッチチップあたりのポート数は増加傾向にあり、最新チップのポート数は 36 である。この 36 ポート IB スイッチチップでステージ数 2 の完全 Fat-tree を構築すると、最多 648 台の計算ノードを接続することができる (ネットワーク構成は 18-ary-2-tree)。これは、ステージ数 2 の多段ネットワークで相当規模のシステムを実現できることを示している。そこで、本稿ではネットワーク構成を 2 ステージ構成の Fat-tree、K-ary-2-tree に限定して議論する。

2.3 シフト通信パターンとルーティング

シフト通信パターンは、全プロセス間でそれぞれ固有のメッセージを送受する全対全通信時によく出現する通信パターンである[7)]。例えば、プロセス数が Np、各プロセスの ID が P_j $(0 \leq j < Np)$ の場合を考える。全対全通信は、Np 回の通信ステージから構成され、通信ステージ i のとき $(0 \leq i < Np)$、プロセス P_j は i 個先のプロセス P_k $(k = (j + i)\ mod\ Np)$ にメッセージを送信する。全対全通信の特徴は、同じ時刻には、全プロセスが基本的に同じ通信ステージを処理しており、同じシフト距離で通信することである。

シフト通信パターンは全対全通信に固有の通信パターンではなく、典型的な通信パターンの一種である。隣接ノード間通信など、規則的な通信パターンの多くは、特定のシフト距離通信にマップできることが知られている[12)]。実際、Fat-tree IB ネットワークではシフト通信に最適化したルーティング設定で高い性能が得られることが報告されている[6)7)]。また、主要な IB SM である OpenSM[8)] にも、この Fat-tree 向けルーティングが組み込まれており、シフト通信パターンに最適化したルーティングは一般的に使われている。

2.4 MPI ジョブへのノード割当

Fat-tree IB ネットワークでは MPI ジョブに適切なノード割当てを行えば高い通信性能が得られるが、実運用システムで常に適切なノード割当てを実現するのは簡単ではない。数百台規模のシステムは、単一ユーザに占有されるのではなく、複数ユーザに共有されるのが常であり、複数の MPI ジョブが同時に実行されている状態が一般的である。また、各 MPI ジョブの並列度 (使用ノード数) や実行時間はジョブ毎に様々であるため、MPI ジョブに対して規則的に、例えば割当てノード番号が連番となるようにノードを割当てる方針でシステム運用すると、システムの稼働率 (ノード利用率) が低下してしまう。ノード利用率を最大化するには、各 MPI ジョブに対してノード番号が不連続なノード、つまり任意のノードの割当てを許す必要がある。

しかし、前述のシフト通信パターンに最適化したルーティングでは、適切なノードを割当てたときには良い性能が得られるものの、適切なノード割当てができなかったときには Hot-spot が発生して通信性能が低下することが知られている[10)]。つまり、各ジョブの通信性能とシステムのノード利用率はトレードオフの関係にあり、両立は困難である。これは、Fat-tree IB ネットワークの解決すべき課題である。

Vol.2010-ARC-192 No.5
Vol.2010-HPC-128 No.5
2010/12/16

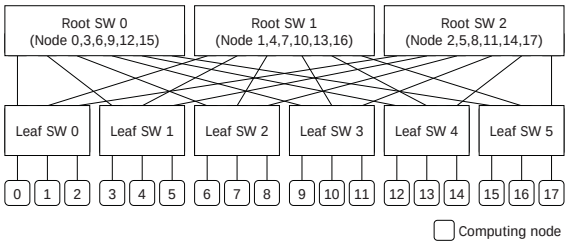


図 1 3-ary-2-tree 構成の Fat-tree ネットワーク例

3. Fat-tree IB ネットワーク上での Hot-spot 発生の仕組み

本章では、Fat-tree IB ネットワーク上で Hot-spot が発生する仕組みを説明する。以下では、ノード数 18、3-ary-2-tree の Fat-tree ネットワーク構成を事例に説明する (図 1)。ルーティング設定は文献 6) と同様の手法を採用し、ノード N_i $(0 \leq i < 18)$ へのパケットは、Root スイッチ $RS_j (j = i \ mod \ 3)$ を経由するルーティング設定になっているものとする。

以下、任意ノード割当て時の Hot-spot 発生の仕組みと、連番ノード割当て時の Hot-spot 発生の仕組みを説明する。

3.1 任意ノード割当て時の Hot-spot 発生

MPI ジョブに任意ノードが割当てられる場合の Hot-spot 発生に関して説明する。図 2 に具体的な Hot-spot 事例を示す。図 2 では、MPI ジョブは 4 ノードジョブであり、ノード N_i $(i \in \{3, 5, 6, 9\})$ の 4 ノードが割り当てられ、ノードあたりのプロセス数は 1、プロセス番号 (rank 番号) はノード番号順に割り当てられている。この条件で、シフト距離が 2 のシフト通信を行うと、ノード N_3 はノード N_6 へ、ノード N_5 はノード N_9 へメッセージを送信する。この 2 通信の送信元であるノード N_3 と N_5 はどちらも Leaf スイッチ LS_1 に接続されたノードである。そして、送信先であるノード N_6 と N_9 へのパケットはどちらも Root スイッチ RS_0 を経由する。従って、この事例ではシフト距離が 2 のシフト通信時に、Leaf スイッチ LS_1 から Root スイッチ RS_0 へのリンクで競合が発生する。

これが任意ノード割当て時の HotSpot 発生の基本的な仕組みである。図 2 では、2 つの通信が 1 つのリンクに集中しているが、状況次第でより多数の通信が 1 つのリンクに集中することがある。その場合には、ネットワークバンド幅が大幅に低下することが報告されている[10)]

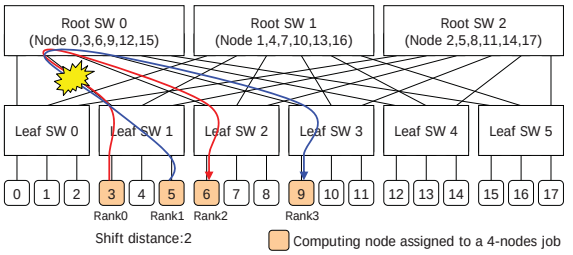


図 2 任意ノード割当て時の Hot-spot 発生例

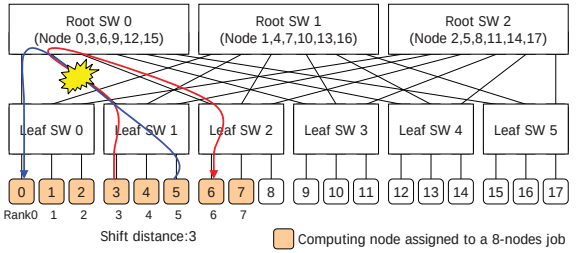


図 3 連番ノード割当て時の Hot-spot 発生例

3.2 連番ノード割当て時の Hot-spot 発生

MPI ジョブに連番ノードを割当てる場合の Hot-spot 発生に関して説明する。図 3 に具体的な Hot-spot 発生事例を示す。図 3 では、ジョブは 8 ノードジョブであり、ノード N_i $(i \in \{0, 1, ..., 7\})$ のノード番号が連続した 8 ノードが割り当てられ、ノードあたりプロセス数は 1 で、プロセス番号 (rank 番号) はノード番号順に割り当てられている。この条件で、シフト距離が 3 のシフト通信を行うと、ノード N_3 はノード N_6 へ、ノード N_5 はノード N_0 へメッセージを送信する。この 2 つの通信の送信元であるノード N_3 と N_5 はどちらも Leaf スイッチ LS_1 に接続されたノードである。一方、送信先であるノード N_6 と N_0 へのパケットはどちらも Root スイッチ RS_0 を経由する。従って、この事例ではシフト距離が 3 のシフト通信時に、Leaf スイッチ LS_1 から Root スイッチ RS_0 へのリンクで競合が発生する。

一般的には、連番ノード割当て時には Hot-spot は発生しないと思われているが、実際には図 3 に示す通り、Hot-spot が発生する。この Hot-spot は、ネットワーク構成に対して使用ノードの配置が非対称なときに発生する。具体的には、Leaf スイッチ毎の使用ノード

Vol.2010-ARC-192 No.5
Vol.2010-HPC-128 No.5
2010/12/16

配置が、Leaf スイッチ間で一致していないときに発生する可能性がある。図 3 の事例では、Leaf スイッチ LS_0 と LS_1 に接続のノードはそれぞれ 3 台使用されているが、Leaf スイッチ LS_2 に接続のノードは 2 台しか使用されておらず、Leaf スイッチ間で使用ノード数が一致していない。

図 3 の事例で Hot-spot 発生を回避するには、例えばノード N_i ($i \in \{0, 1, 3, 4, 6, 7, 9, 10\}$) のように、ネットワーク構成に対して使用ノードの配置が対称となるよう注意深くノードを選択する必要がある[10]。そうしないと、Hot-spot が発生し通信性能が低下する。

4. Hot-spot 発生を回避する手法の提案

本章では、Fat-tree 構成 IB ネットワーク上で、全対全通信時に発生する Hot-spot を回避する手法を提案する。

IB は static ルーティングであり、動的な経路変更は困難である。そこで、提案手法では、IB の LMC 機能を利用して、事前に、各計算ノードに複数の LID を割当て、各 LID に対してそれぞれ異なる経路を設定する。MPI ジョブ実行時には、全ての送信元ノード-送信先ノード対に関して、それぞれ Hot-spot 発生を回避できる通信経路を算出し、その経路が設定された LID を使用して該当計算ノード対間でコネクションを生成する。これにより、動的な経路変更に相当する効果を得ることができる。

各計算ノード対間の適切な経路は、全対全通信時の特徴、すなわち、同じ時刻には全プロセスが同じシフト距離で通信する特徴を利用して決定する。具体的には、各計算ノード対間に生成されるコネクション毎に、同じ時刻に使用されるコネクションと競合しない通信経路を選択し、その経路に対応する LID を使用する。従って、各計算ノードが使用する LID は、通信相手により変わる可能性がある。

以下、各ノードに複数 LID を割り当てる方法と、コネクション毎に適切な LID を選択する方法を説明する。

4.1 複数 LID 割当てと各 LID のルーティング設定

本節では、複数 LID を割り当てる方法と、各 LID のルーティング設定方法を説明する。

各計算ノードに複数 LID を割り当てる方法は、文献 14) と同じである。LMC 機能を用いて、各計算ノードに K 個以上の LID を割り当てる (ネットワークトポロジーは K-ary-2-tree を想定、ルートスイッチ数は K 個)。ルーティングは、K 個の LID を宛先とするパケットが、それぞれ別個のルートスイッチを経由するように設定する。

図 4 に、ネットワーク構成が 3-ary-2-tree の場合の具体的な複数 LID 割当て例と各

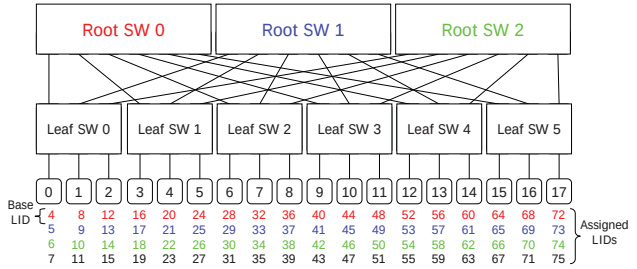


図 4 複数 LID 割当てと各 LID に対するルーティング設定の例

LID のルーティング設定例を示す。計算ノード N_0 には LID が 3 個以上割り当てられ (LID_j ($j \in \{4, 5, 6, 7\}$))、LID_4、LID_5、LID_6 宛のパケットは、それぞれルートスイッチ RS_0、RS_1、RS_2 を経由するように設定される。

4.2 使用 LID の選択

MPI プログラムが起動して、各計算ノード対の間にコネクションを生成するときに、各コネクション端 (計算ノード) の使用 LID を選択する方法を説明する。使用 LID の選択手順は、図 5 に示す通り、次の 6 ステップである。

(1) ローカルノード ID の割当て
(2) 初期ルートスイッチテーブルの作成
(3) 初期ルートスイッチテーブルの変換
(4) 各計算ノード対間のパケットが経由するルートスイッチの決定
(5) 最終ルートスイッチテーブルの作成
(6) LID テーブルの作成

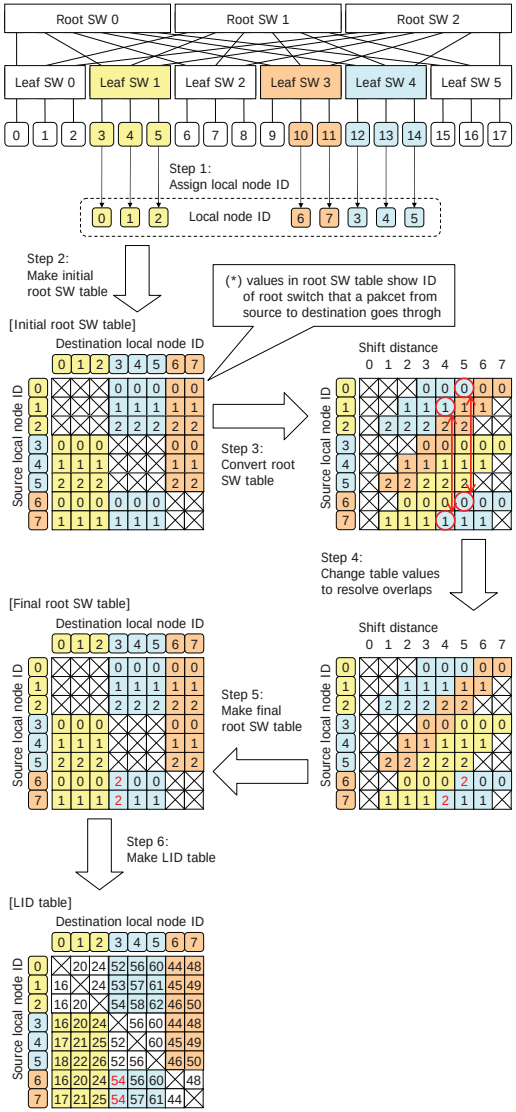
以下、それぞれのステップを説明する。

手順 1: ローカルノード ID の割当て

MPI ジョブに割り当てられた計算ノードに対して、ローカルノード ID を割り当てる。通常は、hostfile 等に記載の順序で、各ノードに対してローカルノード ID を割り当てるが、本手法では、ネットワーク構成に準じてローカルノード ID を割り当てる。具体的には、Leaf スイッチ毎の割当て計算ノード数を集計し、割当て計算ノード数の多い Leaf スイッチ配下の計算ノードから、順次、ローカルノード ID を昇順で割り当てる。

図 5 の事例では、Leaf スイッチ LS_1 と LS_4 に接続された割当て計算ノード数が 3 である

Vol.2010-ARC-192 No.5
Vol.2010-HPC-128 No.5
2010/12/16




```
/*
 *  Psudo code of step 3 -- 5
 */
for (sd=1; sd< N; sd++) {
    /*** clear path counters ***/
    memset(up_path_cnt, 0, ...);
    memset(dn_path_cnt, 0, ...);
    memset(up_path_likely, 0, ...);
    memset(dn_path_likely, 0, ...);
    /*** initialize path counters ***/
    for (s=0; s<N; s++) {
        d=(s+sd) % N;
        if (ls(s) == ls(d)) continue;
        if (ls(s) == ls(0)) {
            up_path_cnt(ls(s), rs(s,d)) += 1;
            dn_path_cnt(ls(d), rs(s,d)) += 1;
        } else {
            up_path_likely(ls(s), rs(s,d)) += 1;
            dn_path_likely(ls(d), rs(s,d)) += 1;
        }
    }
    /*** check and update root switch table ***/
    for (s=N-1; ls(s) != ls(0); s--) {
        d=(s+sd) % N;
        if (ls(s) == ls(d)) continue;
        rs_new = rs_likely = rs(s,d);
        if (up_path_cnt(ls(s), rs(s,d))>0 ||
            dn_path_cnt(ls(d), rs(s,d))>0) {
            /* initial path is busy. need to find idle path */
            for (i=1; i<K; i++) {
                j = (rs(s,d) - i + K) % K;
                if (up_path_cnt(ls(s), j)>0) continue;
                if (dn_path_cnt(ls(d), j)>0) continue;
                if (rs_tmp == rs(s,d)) {
                    /* likely-idle path is found */
                    rs_likely = j;
                }
                if (up_path_likely(ls(s), j)>0) continue;
                if (dn_path_likely(ls(d), j)>0) continue;
                rs_new = j;    /* idle path is found */
                break;
            }
            if (rs_new == rs(s,d)) {
                rs_new = rs_likely;
            }
        }
        /* update path counters and root switch table */
        up_path_likely(ls(s), rs(s,d)) -= 1;
        dn_path_likely(ls(d), rs(s,d)) -= 1;
        rs(s,d) = rs_new;
        up_path_cnt(ls(s), rs(s,d)) += 1;
        dn_path_cnt(ls(d), rs(s,d)) += 1;
    }
}
//
//  N:   Number of nodes
//  K:   Number of root switchs in K-ary-2-tree network
//  sd:  Shift distance
//  s:   Source local node id
//  d:   Destination local node id
//  ls(i):  ID of leaf switch which local node i is connected to
//  rs(s,sd):  Root switch table
//  up_path_cnt(i, j):  Number of connections that use the path
//                      from leaf sw i to root sw j
//  dn_path_cnt(i, j):  Number of connections that use the path
//                      from root sw j to leaf sw i
//  up_path_likey(i, j):  Number of connections that are likely to
//                        use the path from leaf sw i to root sw j
//  dn_path_likey(i, j):  Number of connections that are likely to
//                        use the path from root sw j ro leaf sw l
//
```

図 5　使用 LID 選択の処理フロー

のに対して、Leaf スイッチ LS_3 に接続された割当て計算ノード数が 2 と少ないため、Leaf スイッチ LS_1 と LS_4 に接続の計算ノードからローカルノード ID を割当て、その後、Leaf スイッチ LS_3 に接続の計算ノードへのローカルノード ID を割り当てる。

手順 2: 初期ルート スイッチテーブルの作成

各計算ノード対間のパケットが経由するロートスイッチの初期値を設定する。送信元ノードと送信先ノードのローカルノード ID を、それぞれ s、d とし、送信元ノード s から送信先ノード d へのパケットが経由するルートスイッチ ID を $rs(s,d)$ とすると、下記計算式を用いて初期値を設定する (K はルートスイッチ数)。

- ルートスイッチテーブルの初期値:　$rs(s,d) = s \ mod \ K$

この方法で、各計算ノード対間のパケットが経由するルートスイッチを設定すると、同一 Leaf スイッチに接続された計算ノードから送出されるパケットは、いずれも別のルートスイッチを経由することになる。従って、Leaf スイッチからルートスイッチへの Up パスでは、競合が発生しない。しかし、ルートスイッチから Leaf スイッチへの Down パスでは、競合が発生する可能性がある。

なお、送信元ノードと送信先ノードが、同一 Leaf スイッチに接続している場合、該当パケットはルートスイッチを経由しないので、本手法では考慮しない。

手順 3: 初期ルート スイッチテーブルの変換

手順 2 で作成した初期ルートスイッチテーブルから、送信元ノードのローカルノード ID s と、シフト距離 sd を引数とするテーブルを作成する。送信元ノード s がシフト距離 sd 先の相手にパケットを送出するときの経由ルートスイッチ ID を $conv\_rs(s,sd)$ とすると、$rs(s,d)$ との関係は以下の通りとなる (N はノード数)。

- 変換後のテーブル: $conv\_rs(s,sd) = rs(s,d) \qquad (d = (s+sd) \ mod \ N)$

なお、本手順は、提案手法の理解を容易にするためのものであり、本質的には不要である。

手順 4: 各計算ノード 対間のパケットが経由するルート スイッチの決定

全対全通信では、基本的に、同一時刻には全ての計算ノードが同じシフト距離で通信を行う特性があるため、全計算ノードが同じシフト距離先のノードにパケットを送出しているときに、通信経路で競合が発生しないようにすればよい。図 5 の変換後ルートスイッチテーブルから分かるように、同じシフト距離で通信を行うときの経由ルートスイッチは、表の縦一列に並んでいる。つまり、変換後ルートスイッチテーブルの縦一列毎に、経路競合の発生しない経由ルートスイッチの組み合わせを見つければよいことになる。

Vol.2010-ARC-192 No.5
Vol.2010-HPC-128 No.5
2010/12/16

表 1 評価環境

Server	30x Self-made IA server
CPU	2x Intel Xeon X5570 (2.93GHz)
Mem	24GB (6x 4GB DDR3-1333 DIMM)
IB HCA	Mellanox MHGH29-XTC (4xDDR)
OS	RHEL5.4 (64bit)
MPI	OpenMPI 1.4.1
Compiler	gcc 4.1.2 (option: -O3)
Network	
Topology	6-ary-2-tree
Switch	7x Flextronics F-X430044 (4xDDR, 24-port)
SM	OpenSM 3.2.6

経由ルートスイッチの組み合わせを全探索すると、その計算量は $O(N \times K^N)$ となり (N はノード数、K はルートスイッチ数)、N が大きくなると現実的な時間内で解くのは困難である。そこで、提案手法では、図 5 の右に疑似コードを示した計算量が $O(N \times N \times K)$ の手法で経由ルートスイッチを決定する。この手法は、後戻りなしで経由ルートスイッチを決定しており、現実的な時間で解を求めることができる。

残念ながら、この手法で最適解 (競合が発生しない解) が求まることを証明できていないが、ネットワーク構成が 3-ary-2-tree 構成である 18 ノードシステムにおいては、任意ノード選択で最適解が求まることを確認している。

手順 5: 最終ルートスイッチテーブルの作成

手順 4 で決定したルートスイッチテーブルから、最終ルートスイッチテーブルを作成する。操作は、手順 3 の逆操作となる。なお、本手順も手順 3 同様、本質的には不要である。

手順 6: LID テーブルの作成

手順 5 で作成した最終ルートスイッチテーブルから、LID テーブルを作成する。各計算ノード対の間でコネクションを生成するときには、この LID テーブルに基づいて、コネクション端で使用する LID を決定する。

5. 評　　価

従来手法 (単一 LID 割当て方式と文献 14) 方式) と提案手法を、All-to-all ベンチマークで評価した結果を説明する。

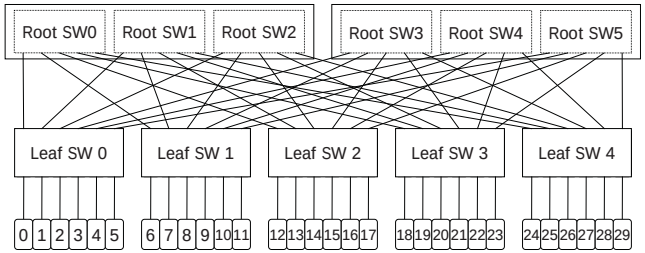


図 6 テスト環境のネットワーク構成 (6-ary-2-tree, 30nodes)

$$Alltoall\_Bandwidth = \frac{Msg \times (Nn-1) \times Nlp \times Nlp}{T}$$

```
Msg: Size of message exchanged
     among each process-pair.
Nn:  Number of nodes.
Nlp: Number of processes in a node.
T:   Elapsed time of all-to-all.
```

図 7 Alltoall バンド幅の定義

5.1 評 価 環 境

測定環境は表 1 に示す通りである。計算ノードは Intel Xeon X5570 を搭載した 2 socket サーバで、それぞれ 1 枚の IB HCA(4xDDR 対応) を搭載している。ノード数は 30 である。ネットワークトポロジーは 6-ary-2-tree であり、7 台の 4xDDR 対応の IB スイッチで構成した (図 6)

Subnet Manager(SM) は OpenSM を使用した。従来手法、提案手法、いずれの場合も適切なルーティングファイルを作成し、それを入力ファイルとして OpenSM を FILE エンジンで起動した。MPI プログラム起動時に使用 LID を選択する仕組みは、OpenMPI[9] に組み込んだ。具体的には、MPI_Init() 直後に各ノードの使用 LID を選択し、OpenMPI の blt_openib コンポーネント内で選択 LID を用いて QP を生成した。

Alltoall ベンチマークの性能指標には、図 7 に示す Alltoall バンド幅を用いた。これは、全対全通信中のノードあたり実効ネットワーク性能を示す指標である。ネットワーク性能 (ノード間通信性能) を知ることが目的であり、ノード内通信量は計算式から除外している。

Vol.2010-ARC-192 No.5
Vol.2010-HPC-128 No.5
2010/12/16

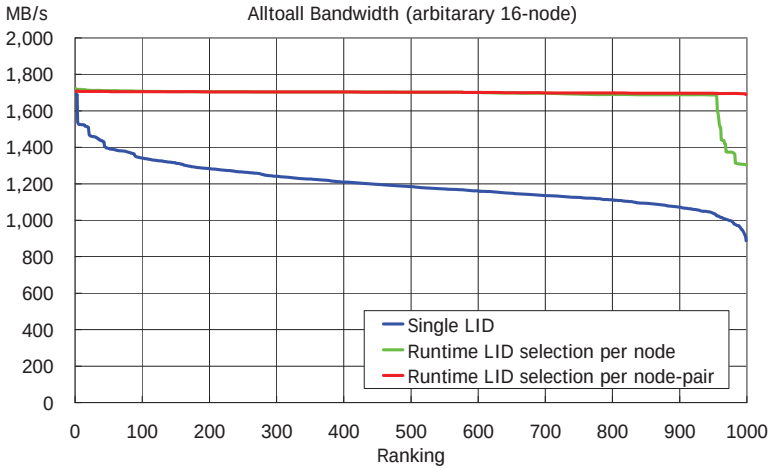


図 8 任意 16 ノードでの全対全通信性能測定結果

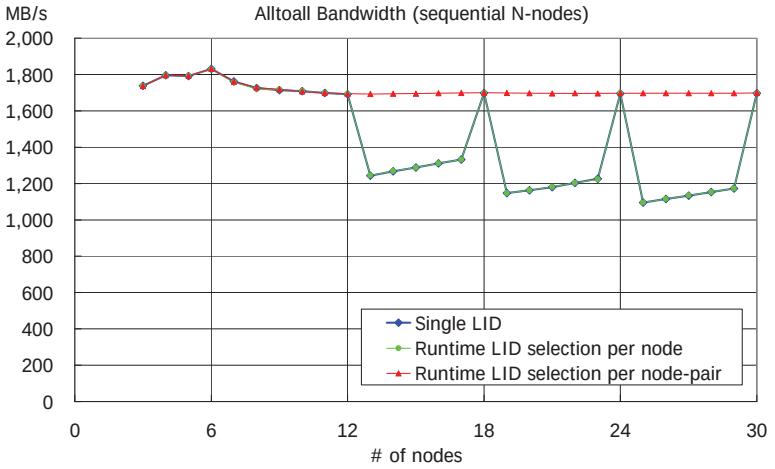


図 9 連番ノードでの全対全通信性能測定結果

実行時間 T にはノード内通信時間も含まれており、厳密には Alltoall バンド幅は実効ネットワーク性能を示していない。しかし、一般的に、ノード内通信はノード間通信に比べ高速であり、実行時間 T に占めるノード内通信の比率は小さいと考えられるので、本稿では Alltoall バンド幅を実効ネットワークバンド幅とみなす。なお、All-to-all 通信アルゴリズムには、マルチコアシステムに適切な 2-level ring アルゴリズムを用いた[13]。また、ノードあたりのプロセス数は 8 で、各プロセス間で交換するメッセージサイズは 1MiB とした。

5.2 任意ノード割当て時の全対全通信性能

図 8 に、全 30 ノードから任意 16 ノードを割当て Alltoall ベンチマークを実行した結果を示す。測定は、それぞれ割当てノードセットの異なる 1,000 種類のホストファイルを用い、1,000 回実施した。図 8 は、縦軸が Alltoall バンド幅であり、横軸は 1,000 回の測定結果を Alltoall バンド幅順に並べたものである。

単一 LID 割当て方式 (青線) は、最高性能は 1,690MB/s と高いが、平均性能は 1,198MB/s、最低性能は 888MB/s であり、割当てノード次第で性能が大きく変動した。文献 14) 方式 (緑線) は、LID 割当て方式と比べて非常に安定しているが、5%程度の割当てノード組み合わせで、Hot-spot 発生によるバンド幅低下が発生した。それに対して、提案手法 (赤線) の性能は、割当てノードと関係無く、常に 1,700MB/s 程度と安定して高バンド幅を記録した。提案手法でネットワーク上の Hot-spot 発生をほぼ完全に回避できた証左といえる。

5.3 連番ノード割当て時の全対全通信性能

図 9 に、連番ノード割当て時の Alltoall バンド幅測定結果を示す。性能測定はノード数 3〜30 で実施、ノード数 Nn のときにはノード N_i $(0 \leq i < Nn)$ を使って性能測定した。図 9 は、縦軸が Alltoall バンド幅であり、横軸がノード数である。

従来手法 (青線と緑線) では、割当てノード数が、12 $(2 \times K)$ 以上、かつ、6 (K) の倍数でない場合には、性能が低下した。これは、従来手法では連番ノード割当て時に発生する Hot-spot を回避できないためである。それに対して、提案手法 (赤線) では、いずれのノード数でも、安定して高いバンド幅 (1,700MB/s 程度) を記録した。これは、計算ノード対毎に適切に使用 LID を選択する本提案手法の効果である。

6. 関連研究

Fat-tree ネットワークでの Hot-spot 発生は以前から知られている問題であり、そのため、Fat-tree 向けの adaptive ルーティング方式がいろいろ検討評価されている[15)16)]。しかし、IB は static ルーティングであり、これら手法は IB ネットワークには適用できない。

各ノード間に複数パスを形成、ネットワークトラフィックを複数パスに分散させて HotSpot

Vol.2010-ARC-192 No.5
Vol.2010-HPC-128 No.5
2010/12/16

発生の影響を低下させる方式も提案されている[3)4)]。しかし、この方式ではスイッチチップ内で HoL ブロッキングが発生して性能が低下するという問題がある[10)]。また、コネクション数が増えるため、通信用バッファ(メモリ使用量)が増加する問題もある。

シフト通信パターンに特化した Fat-tree 向け static ルーティング方式が提案されている[6)7)]。この手法は OpenSM に既に組み込まれ一般的な手法であるが、本稿で示した通り、ネットワーク構成に対してノード割当てが非対称な場合には Hot-spot が発生する。

文献 14) の手法は、基本的な概念は本提案手法に近いが、ノード単位で使用 LID を選択しており、Hot-spot 発生を完全には回避できない。また、連番ノード割当て時の Hot-spot 発生も回避できない。

7. ま と め

本稿では、2 ステージ Fat-tree IB ネットワークにおける全対全通信時の Hot-spot 発生を完全に回避する手法を提案した。6-ary-2-tree 構成のネットワークに接続した 30 ノードの PC クラスタシステム上で提案手法を評価した結果、任意割当てノードにおける全対全通信中の Hot-spot 発生を、従来手法では完全には回避できなかったのに対して、提案手法では完全に回避できることを確認した。特に、従来手法が不得意とした、連番ノード割当て時の Hot-spot 発生も、提案手法で完全に回避できることを示した。

今後の課題は、大規模システムへの応用である。提案手法は、少なくともルートスイッチ数と同数の LID を、各計算ノードに割当てることを前提としているが、システム規模が大きくなると、計算ノード数が増加すると同時に、ルートスイッチ数も増加する。そのため、1,000 ノード超システムに本提案手法を適用するのは現実的ではない。今後は、より大規模なシステムへの適用を目指し、提案手法を改良する予定である。

参 考 文 献

1) F. Petrini, and M. Vanneschi: k-ary n-trees: High Performance Networks for Massively Parallel Architectures, In *Proceedings of the 11th International Parallel Processing Symposium (IPPS97)*, 1997.
2) InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.2, October 2004.
3) X. Lin, Y. Chung, and T. Huang: A Multiple LID Routing Scheme for Fat-tree-Based InfiniBand Networks, In *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium*, 2004.
4) A. Vishnu, M. Koop, A. Moody, A.R. Mamidala, S. Narravula, and D.K. Panda: Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective, In *Proceedings of the 7th IEEE Inernational Symposium on Cluster Computing and the Grid*, 2007.
5) M. Karol, M. Hluchyj, and S. Morgan: Input Versus Output Queueing on a Space-Division Packet Switch, *IEEE Transactions on Communications*, Volume 35, Issue 12, pp. 1347-1356, 1987.
6) C. Gomez, F. Gilabert, M.E. Gomez, P. Lopez, and J. Duato: Deterministic versus Adaptive Routing in Fat-trees, In *Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium (IPDPS07)*, 2007
7) E. Zahavi, G. Johnson, D.J. Kerbyson, and M. Lang: Optimized infiniband fat-tree routing for shift all-to-all communication patterns, In *Proceedings of the International Supercomputing Conference 2007 (ISC07)*, 2007.
8) The OpenFabrics Alliance: http://www.openfabrics.org/
9) OpenMPI: http://www.open-mpi.org/
10) T. Hoefler, T. Schneider, and A. Lumsdaine: Multistage switches are not crossbars: Effects of static routing in high-performance networks, In *Proceedings of the 2008 IEEE International Conference on Cluster Computing (Cluster08)*, 2008.
11) A. Faraj, and X. Yuan: Automatic generation and tuning of MPI collective communication routines, In *Proceedings of the 19th Annual international Conference on Supercomputing (SC05)*, 2005.
12) K.J. Barker, A. Benner, R. Hoare, A. Hoisie, A.K. Jones, D.K. Kerbyson, D. Li, R. Melhem, R. Rajamony, E. Schenfeld, S. Shao, C. Stunkel, and P. Walker: On the Feasibility of Optical Circuit Switching for High Performance Computing Systems, In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing (SC05)*, 2005.
13) 成瀬 彰, 中島 耕太, 住元 真司, 久門 耕一: マルチコア PC クラスタ向け All-to-all 通信アルゴリズムの提案と評価, In *Proceedings of SACSIS 2010*, 2010.
14) 成瀬 彰, 中島 耕太, 住元 真司, 久門 耕一: 多段スイッチ InfiniBand ネットワークにおける全対全通信性能の評価, 情報処理学会研究報告, Vol.2010-HPC-126, No.16, 2010.
15) Z. Ding, R.R. Hoare, A.K. Jones, and R. Melhem: Level-wise scheduling algorithm for fat tree interconnection networks, In *Procddings of the 2006 ACM/IEEE conference on Supercomputing (SC06)*, 2006.
16) P. Geoffray, and T. Hoefler: Adaptive Routing Strategies for Modern High Performance Networks, In *Proceedings of the 16th IEEE Symposium on High Performance Interconnects*, 2008.
17) G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato: Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control, In *Proceedings of HP-IPC 2005*.