Vol.2015-HPC-151 No.3
2015/9/30

接続台数を最大化するラテン方陣 Fat-Tree における All-to-all 通信での経路競合回避

清水俊宏[†1]　　中島耕太[†1]

近年，クラスタシステムは大規模化の傾向にあり，より多くのサーバを接続して並列に処理するシステムが増えてきた．サーバを接続する際のトポロジー構造としては Fat-Tree が広く用いられている．サーバを接続する際に必要なスイッチのコストは高いため，出来るだけスイッチ数を削減したい．Fat-Tree でサーバを接続すると，サーバ間の経路が複数あることからネットワーク性能は高いものの，多くのスイッチを必要とするため，コストが高くなる問題が生じる．

ラテン方陣 Fat-Tree という従来の Fat-Tree と比較して少ないスイッチ数で多くのサーバを接続可能なトポロジーが提案されている．このトポロジーは，サーバ接続のためのスイッチ台数が少なくて済む一方で，サーバ間の経路が 1 通りしかなく，All-to-all 通信などの高負荷な通信処理を行う際に経路競合を起こしやすいため，実用上はあまり用いられてこなかった．

本稿ではラテン方陣 Fat-Tree のトポロジーの特徴に注目し，All-to-all 通信の経路競合を回避する手法を考案する．実用的なクラスタシステムの運用においては，部分的にサーバ群を切り出してジョブを投入するため，部分的なサーバ群における All-to-all 通信の経路競合を回避する方法を提案する．

Link congestion avoidance of All-to-all communication in Latin square Fat-Tree which maximizes the number of connected servers.

TOSHIHIRO SHIMIZU[†1]　KOHTA NAKASHIMA[†1]

Recently, large scale cluster systems are required. Large scale cluster systems have many servers connected by switches. Fat-Tree is a one of the major topology for cluster system. The cost of the switch which is used to connect servers is high. To reduce the number of switches is required. Since Fat-Tree has multiple paths between two servers, we can attain high network performance using Fat-Tree. However, many switches are required and the cost will rise.

Latin Square Fat-Tree is proposed as a topology which enables to connect more servers than normal Fat-Tree. Using the topology, we can reduce the number of switches, on the other hand, there is only one route between each two servers and this tends to cause link congestion during high-load network operation such as All-to-all communication. Thus, the topology is not used practically.

In this paper, we focus on the feature of Latin Square Fat-Tree to invent a method of All-to-all communication avoiding link congestion. Since, in practical cluster system, we submit a job to a server group which has some selected servers in the cluster system, we propose a method of All-to-all communication in a server group avoiding link congestion.

1. はじめに

近年，計算科学の進歩が著しく，科学技術計算の高速化の要求が高まっている．高速化のために，クラスタシステムの大規模化が必要とされており，より多くの計算サーバを接続する技術が求められている．サーバ間の中継に用いるスイッチは高価であるため，少ないスイッチで多くのサーバを接続できることが望ましい．

クラスタシステムでは Fat-Tree による接続が広く採用されている．Fat-Tree を用いることで，各サーバから他のすべてのサーバに通信する All-to-all 通信という非常に負荷の高い通信処理を競合なく実行することができるようになった．しかし，Fat-Tree で多くのサーバをつなぐときにはさらに多くのスイッチが必要となるため，少ないスイッチでより多くのサーバをつなぐことのできるトポロジー構造が求められつつある．

多層 full-mesh[1]やラテン方陣 Fat-Tree[2]は従来の Fat-Tree より多くのサーバをつなぐことを目的として提案されたトポロジー構造である．ラテン方陣 Fat-Tree は従来の Fat-Tree, 多層 Fullmesh に比べてスイッチのコスト面での効率がよいが，その分サーバ間の結合が疎になっているために，All-to-all 通信といった高負荷な通信が競合なく行える手法は確立していない．ラテン方陣 Fat-Tree は経路競合を避けることができれば，コストを低く抑えられ有用である．

そこで，本稿ではラテン方陣 Fat-Tree において All-to-all 通信を行う際に経路競合を回避する手法を提案する．

実用的なクラスタシステムの運用を考えるとクラスタシステム全体に一つの計算ジョブを割り当てる事はまれであり，部分的にサーバを切り出してそこにジョブを割り当てる運用が一般的である．

そこで，経路競合を回避する手法を検討するにあたって，ラテン方陣 Fat-Tree における部分的に切り出したサーバ群の All-to-all 通信において，競合をなくす手法とその際のサ

†1 (株)富士通研究所
Fujitsu Laboratories Ltd.

一バ群の切り出し方法を提案し，評価する．

2. 課題

2.1 従来の Fat-Tree における All-to-all 通信

2.1.1 Fat-Tree とは

多くのクラスタシステムでは InfiniBand[3]による図 1 のような Fat-Tree トポロジーが採用されている．Fat-Tree は通信負荷の高い All-to-all 通信において経路競合を回避できる．これによって，All-to-all 通信を内部で実行する FFT などの処理を高速かつ並列に処理することができる．

Fat-Tree の具体的な構造は図 1 のようになっている．上段のスイッチは spine スイッチ，中段のスイッチは leaf スイッチと呼ばれ，spine スイッチからすべての leaf スイッチへ均等にリンクでつなぐことで実現できる．なお，spine スイッチは leaf スイッチに比べて半分のポートしか用いていないため，spine スイッチの反対側に同様の構造を構築することが可能であり，2 倍のサーバをつなぐことが可能となる．今後は片側のみの接続で考えるものとする．

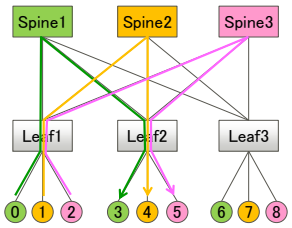

図 1 Fat-Tree の構造

2.1.2 Fat-Tree における All-to-all 通信の競合回避

Fat-Tree において All-to-all 通信を行う場合，シフト通信パターンを採用することで競合を回避することが知られている．シフト通信パターンとは，i番目のフェーズでサーバ番号Sのサーバは$(S+i)\%N$に送信する，という通信パターンである．ただし，Nは切り出したサーバ数である．[4]

切り出しサーバ台数がスイッチのポート数の倍数である場合は，同一の leaf スイッチ配下のサーバ群を選んで，シフト通信パターンでの All-to-all 通信を実行することで競合が回避できる．

例えば，図 1 のような Fat-Tree のすべてのサーバを用いたシフト通信パターンでの All-to-all 通信は図 2 のようになる．

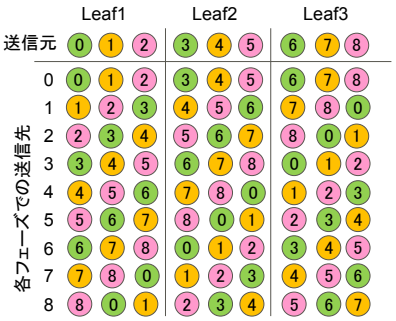

図 2 シフト通信パターンの様子

2.1.3 実用的な運用を考えた All-to-all 通信

実用的には常にクラスタシステムのすべてのサーバを利用した大規模な All-to-all 通信を行うわけではないため，一部のサーバを切り出してそれらの間で All-to-all 通信のジョブを投げるということも頻繁に行われる．切り出しの規模は計算の用途に応じて様々であるため，多様な台数で柔軟に All-to-all 通信が行えることが望ましい．

2.2 ラテン方陣 Fat-Tree

クラスタシステムの大規模化に伴い，少ないスイッチ数でより多くのサーバを並列に処理する必要性が生じてきている．Fat-Tree よりも多くのサーバをつなぐという目的で，ラテン方陣 Fat-Tree[2]というトポロジー構造が提案されている．このトポロジー構造は同一のスイッチ数で Fat-Tree よりも多くのサーバを接続できる．

2.2.1 有限射影平面の定義

ラテン方陣 Fat-Tree の構造を説明するために，有限射影平面[5]の構造を述べる．

nを素数とする．位数nの有限射影平面とは，n^2+n+1個の点とそれらの点のうち$n+1$個を結ぶn^2+n+1本の直線からなる平面である．具体的には以下で定義される．

点は以下のn^2+n+1個とする

- P
- $P(c)\ (0 \le c < n)$
- $P(c,r)\ (0 \le c,r < n)$

直線および直線が通る点は下記のn^2+n+1本とする．

- $L = \{P\} \cup \{P(c) \mid 0 \le c < n\}$
- $L(c) = \{P\} \cup \{P(c,i) \mid 0 \le i < n\}\ (0 \le c < n)$
- $L(c,r) = \{P(c)\} \cup \{P(i,(r+ci)\%n) \mid 0 \le i < n\}\ (0 \le r,c < N)$

$P(c,r)\ (0 \le c,r < n)$の部分を格子部分と定義する．

$n = 2,3$の場合の構造はそれぞれ図 3，図 4 のようになる．

点$P, P(c)$は無限遠点と呼ばれ，格子部分では交わらない(平行な)二直線の交点となっている．射影平面内ではどの二直線も交わるという性質をもつ．

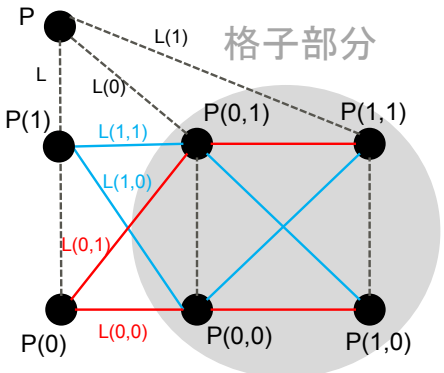

図 3 $n = 2$の場合の有限射影平面

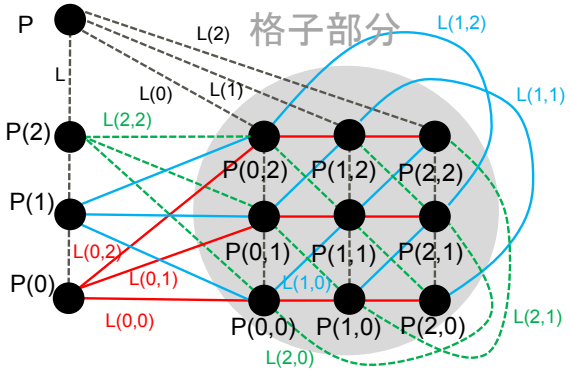

図 4 $n=3$の場合の有限射影平面

位数nの有限射影平面はサイズnの直交ラテン方陣$n-1$枚と等価であることが知られている[2]

2.2.2 有限射影平面とトポロジーの対応

ラテン方陣 Fat-Tree のトポロジー構造は位数nの有限射影平面に対して以下の変換を施すことで得られる．[2]

1. 直線LをspineスイッチL'，点PをleafスイッチP'に対応させる．
2. 直線L上に点Pがある場合，またその場合に限りL'とP'をリンクで結ぶ．
3. 各leafスイッチの配下に$n+1$本のサーバにつなげる．

このようにして得られるトポロジー構造をラテン方陣 Fat-Tree と定義する．使用するスイッチのポート数はspineスイッチは$n+1$ポート，leaf スイッチは$2(n+1)$ポートとなる．このようにして得られたラテン方陣 Fat-Tree の接続サーバ台数は$(n+1)(n^2+n+1)$台であり，従来の Fat-Tree のn^2台と比べて格段に多いものである．

今後は対応後の leaf スイッチ, spine スイッチの名称を対応前の点の名称で呼ぶことにする．

$n=2, n=3$の場合のラテン方陣 Fat-Tree の構造は図 5，図 6 のようになる．

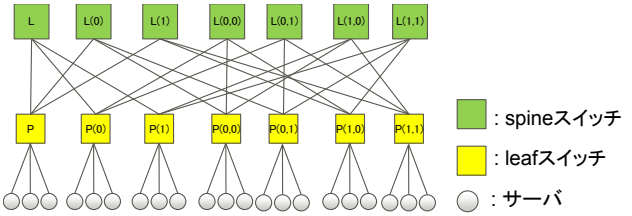

図 5 $n=2$の場合のラテン方陣 Fat-Tree

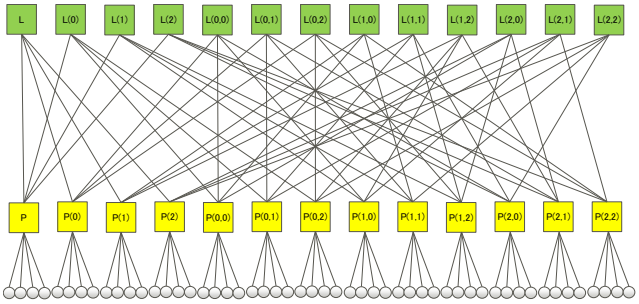

図 6 $n=3$の場合のラテン方陣 Fat-Tree

2.3 課題

ラテン方陣 Fat-Tree は少ないスイッチで多くのサーバを接続できるというメリットがあるが，トポロジー構造が疎になってしまうため，競合なく All-to-all 通信をすることが可能かどうか，知られていなかった．このため，実用的なクラスタシステムの接続方式としては採用されていない．本稿では，この課題を解決するために，ラテン方陣 Fat-Tree の一部のサーバを切り出して All-to-all 通信を行う手法を提案する．

3. 提案手法

3.1 サーバの切り出し方法

まず，投入するジョブのサーバ群を選択する方式について説明する．ラテン方陣 Fat-Tree に対応する有限射影平面の格子部分に注目し，格子部分($n \times n$の形状)の$n \times k$(kは任意の自然数)の長方形を切り出す．切り出した点に対応する leaf スイッチそれぞれに対して，配下のサーバのうちm台(mは$m \le k$をみたす自然数)を取り出す．したがって，合計nmk台が切り出される(図 7 参照)．本手法では，$m \le k \le n$なる自然数n, kを用いて$D=mnk$と表せるようなD台を用いたジョブが実行でき，All-to-all 通信が可能となる．

なお，以降の説明のため，各 leaf スイッチ配下のサーバm台に$0,1,2,\ldots,m-1$という leaf 配下内 ID を割り当てる．

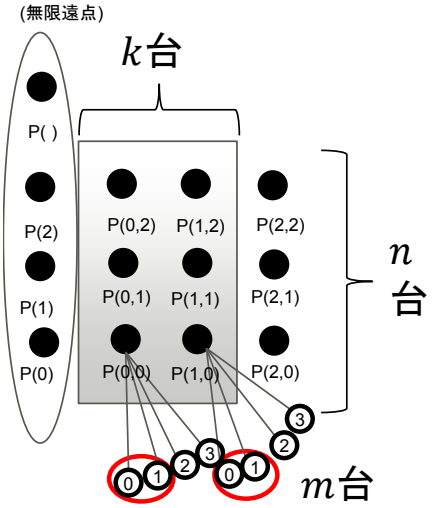

図 7 leaf スイッチの取り出し方法

3.2 All-to-all 通信における通信順序の決定法

3.2.1 フェーズ群内の通信

切り出したサーバ間で all-to-all 通信を行う場合のフェーズ数もnkmフェーズとなる．このnkmフェーズをmフェーズごとにnk個のフェーズ群に分ける．同一のフェーズ群で各サーバは同じ宛先 leaf スイッチに送信する．ただし，この宛先 leaf スイッチ配下の宛先サーバは次のようにして，フェーズ群内のフェーズごとにそれぞれ異なるようにする．

宛先 leaf スイッチは各フェーズでm台からの通信を受信する．このm本の通信の送信元 leaf 配下内 ID は異なってお

り，その leaf 配下内 ID に応じて$0,1,\dots,m-1$と通信番号をつけておく．通信番号がiである通信のj番目のフェーズにおける宛先サーバは，leaf スイッチ配下内 ID $(i+j)\%m$のサーバに送るものとする．$m=2,4$の場合の具体例はそれぞれ表 1，表 2 のようになる．

表 1 $m=2$の場合における 1 フェーズ群内の転送手順

通信番号＼フェーズ	0	1
0	0	1
1	1	0

表 2 $m=4$の場合における 1 フェーズ群内の転送手順

通信番号＼フェーズ	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

3.2.2 各フェーズ群に対応する送信先 leaf スイッチの決定

3.2.1 項では，フェーズ群内のフェーズにおける送信先を決定した．ここでは，それぞれのフェーズ群でどの leaf スイッチを選択するかを議論する．

まず，格子状に並べた各 leaf スイッチについて，leaf スイッチ間の移動を「ベクトル」として表現する．このベクトルが右にxマス，上にyマス移動しているとき，(x,y)と成分表示する．次に，ベクトルの傾きを定義する．成分表示が(0,0)であるベクトルをゼロベクトルと呼ぶ．成分表示が(x,y)である(ゼロベクトルでない)ベクトルの傾きを，$x\neq 0$であるときは$yx^{-1} \bmod n$と定義し，$x=0$であるときは∞と定義する．ただし，x^{-1}は$\bmod n$におけるxの乗法の逆元であり，$xx^{-1}\equiv x^{-1}x\equiv 1(\bmod n)$をみたす．例えば，$n=3$の場合において$\bmod 3$で考えると，ベクトル(1,2)の傾きは$2\times 1^{-1}=2$である．ここで，$2\times 2\equiv 2\times 2\equiv 1$であることから，mod 3 における 2 の逆元は 2 であるので，ベクトル(-1,2)の傾きは$2\times(-1)^{-1}\equiv 2\times 2^{-1}=1$であり，ベクトル(2,1)の傾きは$1\times 2^{-1}=2$である．また，ベクトル(0,2)の傾きは∞である．

次に，ベクトルのホップ数を定義する．成分表示が(x,y)であるベクトルのホップ数は，$x>0$である場合にはxで定義し，$x<0$である場合には$k+x$で定義する．$x=0$である場合のホップ数は$y>0$であればy，$y<0$であれば$n+y$で定義し，ゼロベクトルのホップ数は 0 と定義する．

傾きs，ホップ数hのベクトルを$[s,h]$で表す．また，ゼロベクトルを[*]と表す．例えば，図 7 のように$n=3,k=2$である場合，ベクトル(1,2)の傾きは 2，ホップ数は 1 であることから，[2,1]と表される．また，ベクトル(-1,2)の傾きは 1，ホップ数は-1+2=1 であることから，[1,1]と表され，ベクトル(0,2)の傾きは∞，ホップ数は 2 であることから，[∞,2]と表される．

格子上の点(a_x,a_y)とベクトル$[s,h]$によって，移る点(b_x,b_y)を次のように定義する．$s\neq 0$であるときは，$b_x=(a_x+h)\%k, b_y=\big(s\,(b_x-a_x)+a_y\big)\%n$とする．これは，$(a_x,a_y)$を通る傾き$s$の直線上の点のうち，$x$座標が$(a_x+h)\%k$である点を表している．$s=0$であるときは，$(b_x,b_y)=(a_x,(b_y+h)\%n)$とする．これは，$(a_x,a_y)$から上に$h$個移動した点である．例えば，点(1,2)をベクトル[1,1]によって移すと，$b_x=(1+1)\%2=0$，$b_y=1*(0-1)+2=1$より，(0,1)に移る．なお，どの点もゼロベクトル[*]によって同一の点に移るものとする．どのベクトルvについても，選択した長方形部分の leaf スイッチにおいて，vによって移る点は異なっている．

上記の傾きの定義によって，ゼロベクトルを除く各ベクトルの傾きは 0 以上$n-1$以下の整数または∞で表される．各$s=0,1,\dots,n-1$について，傾きがsであるベクトルは$k-1$本存在し，傾きが∞であるベクトルは$n-1$本，ゼロベクトルが 1 本存在し，これらを合計するとnk本となる．

各フェーズ群で決める各サーバの送信先 leaf スイッチはベクトルを指定することで決めることができ，これは傾きとホップ数によって決まる．各フェーズ群では各 leaf スイッチ配下にある leaf 配下内 ID が同一の複数サーバは，同一の成分表示を持つベクトルを指定する．これによって，同一 ID のサーバ同士が競合することはない．また，同一の leaf スイッチ配下の全サーバはそれぞれ傾きの異なるベクトルを指定することで経路競合を回避することができる．

例えば，図 8 のように leaf スイッチ$P(0,0)$配下の leaf 配下内 ID 0 番のサーバと leaf スイッチ$P(1,0)$配下の leaf 配下内 ID 0 番のサーバは同一のベクトル$[\infty,2]$(赤いベクトル)を指定することで，経路競合は発生しない．

また，leaf スイッチ$P(0,0)$配下の leaf 配下内 ID 0 番と 1 番のサーバにおいて，別々の傾きであるベクトル$[\infty,2]$(赤)，[1,1](青)に送ると，同一の spine スイッチを共有することがないため，経路競合は発生しない．

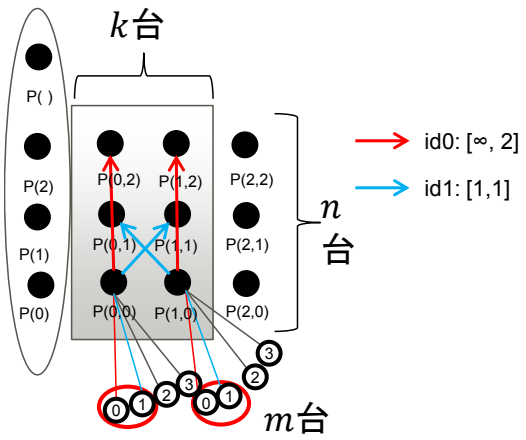

図 8 各サーバの送信方向

以上をまとめるとnk個のフェーズ群について，leaf 配下内 ID ごとにm本のベクトルを指定することになるが，その制約は以下のようになる．

1. どのフェーズ群においても，指定するベクトルの傾きはすべて異なる．
2. 各 leaf 配下内 ID について，全nkフェーズ群で現れるベクトルは上記のnk本が網羅されている．

この制約をみたすような通信順序を決定する．

3.2.3 ベクトル表の作成方法

ここで，通信パターンをベクトル表で表現する．

ベクトル表とは，ある送信元 leaf スイッチ配下の leaf 配下内 ID 0～m-1 に対応するサーバから各フェーズ群における，宛先 leaf スイッチを決定する表である．

行方向に leaf 配下内 ID を並べ，列方向にフェーズ群を並べたときの leaf 配下内 ID におけるフェーズ群での宛先 leaf スイッチをベクトルで指定する．このベクトルはすべての送信元 leaf スイッチにおいて，その送信元 leaf を始点としたベクトルであり，これによって対応する宛先が定まる．

leaf 配下内 ID0 の列には順に，傾きが∞のベクトル($n-1$本)，各$s=0,1,\dots,n-1$について，傾きがsであるベクトル($k-1$本ずつ)を並べ，最後にゼロベクトルを並べる．

leaf 配下内 ID1 の列に関しては，leaf 配下内 ID0 の列を垂直方向下向きに$n-1$個シフトした列を作成する．ただし，下の$n-1$マスに関しては上に移動させる．以下，各列は左隣の列を垂直方向下向きに$n-1$個シフトした列として作成する．$(n,k,m)=(3,2,2)$の場合は表 3 のようになる．

表 3 $(n,k,m)=(3,2,2)$の場合のベクトル表

フェーズ群/leaf 配下内 ID	0	1
0	[∞,1]	[2,1]
1	[∞,2]	[*]
2	[0,1]	[∞,1]
3	[1,1]	[∞,2]
4	[2,1]	[0,1]
5	[*]	[1,1]

このようにして作成したベクトル表が実際に前節の制約をみたしていることを証明する．制約 2. については自明であるため，制約 1. を示す．

$0\le i<j<m$なる整数i,jについて，第j列は第i列を$(j-i)(n-1)$個分垂直方向下向きにシフトしたものである．ここで，各列において傾きが同一の区間は連続しており，この区間の長さは最長で$n-1$である(傾き∞の場合)ことに注目する．垂直方向下向きに$v(0\le v<nk)$個シフトしたとき，この区間が元の区間と重ならないための条件は，$n-1\le v<nk-(n-1)$であり，$v=(j-i)(n-1)$の場合にこの不等式をみたすことを示せばよい．$j-i\ge 1$であることから左の不等号は自明である．右の不等号は$(j-i+1)(n-1)<nk$と同値であり，$(j-i+1)(n-1)\le m(n-1)<kn$より従う．以上により，経路競合が起こらないことが証明された．

3.2.4 送信表の作成

フェーズ群内の通信 3.2.1 項のフェーズ群内の通信パターンと 3.2.3 項のベクトル表から，通信パターンを作成すると表 4 のようになる．なお，サーバに図 9 のような通し番号をつけた．

表 4 送信表

フェーズ群	フェーズ	フェーズ（通し番号）	サーバID											
			0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	4	11	6	9	8	3	10	1	0	7	2	5
	1	1	5	10	7	8	9	2	11	0	1	6	3	4
1	0	2	8	1	10	3	0	5	2	7	4	9	6	11
	1	3	9	0	11	2	1	4	3	6	5	8	7	10
2	0	4	2	5	0	7	6	9	4	11	10	1	8	3
	1	5	3	4	1	6	7	8	5	10	11	0	9	2
3	0	6	6	9	4	11	10	1	8	3	2	5	0	7
	1	7	7	8	5	10	11	0	9	2	3	4	1	6
4	0	8	10	3	8	1	2	7	0	5	6	11	4	9
	1	9	11	2	9	0	3	6	1	4	7	10	5	8
5	0	10	0	7	2	5	4	11	6	9	8	3	10	1
	1	11	1	6	3	4	5	10	7	8	9	2	11	0

第 0 フェーズ群の 0 番目のフェーズでは図 9 のような通信が行われる．0 番のサーバの通信経路について説明する．表 3 のベクトル表によると，第 0 フェーズ群では leaf 配下内 ID 0 のサーバにはベクトル[∞,1]が指定されている．したがって，直上の leaf スイッチ$P(0,0)$を経由して leaf スイッチ$P(0,1)$に到達する．ここで，表 1 の$n=2$の場合によって，送信元 leaf 配下内 ID が 0 の通信(通信番号 0)は宛先 leaf 配下内 ID も 0 に転送されるので，最終的にサーバ 4 に転送される．

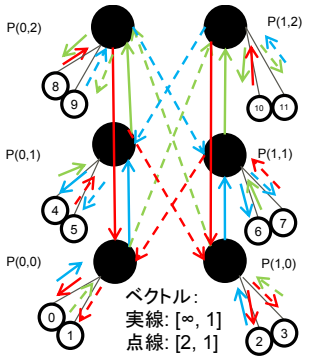

図 9 第 0 フェーズ群 0 番目のフェーズの通信

3.3 切り出しできる台数とその分布

$n=17$の場合，ラテン方陣 Fat-Tree での接続サーバ台数は 5526 台である．

まず，ちょうど必要な台数分のみ切り出すことを許す場合を考える．部分切り出し可能な台数は 114 通りあるため，全体の 2.06%の台数でこのような切り出しが可能となる．

次に，必要な台数よりも多くの台数を切り出して用いることを許す場合を考える．余分に必要となった台数を本来の必要台数で割った比率をオーバーヘッドとする．例えば，1000 台での All-to-all 通信を行いたい場合に 1000 台以上で実現可能な台数である 1020(=17 × 12 × 5)台での All-to-all 通信を行う場合，2%(20 台)のサーバは必要でないにも関わらず All-to-all 通信に参加させなくてはならず，オーバーヘッドとなる．

台数の区間別のオーバーヘッドの平均値は図 10 のようになる．どの区間でもオーバーヘッドの平均値は高々 2.96%であり，実用上の問題は低いと考えられる．

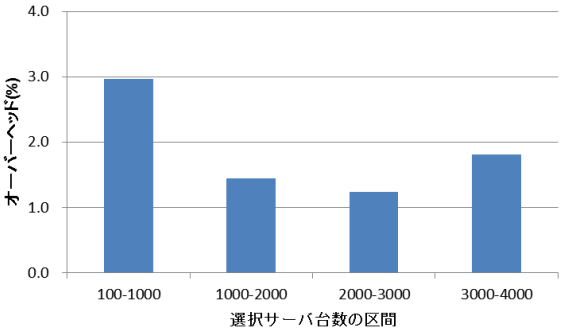

図 10 区間別のオーバーヘッド

4. シミュレーションによる評価

4.1 評価方法

トポロジー構造として従来の Fat-Tree とラテン方陣 Fat-Tree の 2 種類について，次数と通信パターン(3 種類)から決まるそれぞれの事例について一部を切り出した All-to-all 通信によるスループットをシミュレーションにより計測する．

4.1.1 評価対象トポロジー

次数とラテン方陣 Fat-Tree におけるパラメータ (n, m, k) と従来の Fat-Tree の台数を下記の 3 種類とした．ラテン方陣 Fat-Tree におけるパラメータから All-to-all 通信に参加する台数が決定し，この台数と従来の Fat-Tree での台数がほぼ同じ値となるように調整した．構成の詳細を表 5 に示す．表 5 に示す次数とは，各スイッチが他のスイッチと接続しているポート数であり，スイッチ全体のポート数の 1/2 である．

表 5 実験に用いるトポロジーのパラメータ

	次数 4	次数 6	次数 8
次数	4	6	8
ラテン方陣 パラメータ (n, m, k)	(3,2,2)	(5,3,2)	(7,3,3)
ラテン方陣 選択サーバ数	12	30	63
Fat-Tree 選択サーバ数	12	30	64
Fat-Tree 比率調整済みサーバ数	4	5	9
ラテン方陣 接続サーバ数	52	186	456
Fat-Tree 接続サーバ数	16	36	64

4.1.2 通信パターン

以下の 3 つの通信パターンを用いる

- シフト通信パターン
  2.1.2 項で説明したシフト通信パターンの評価である．
- 提案手法
  3 章で説明した通信パターンである．これについては，ラテン方陣 Fat-Tree のみ評価した．
- ランダム通信パターン
  各送信フェーズにおける送信先は，異なるフェーズで同一のサーバが同一の宛先に送信しないようにランダムに送信する．

4.1.3 スループットの計算方法

各通信フェーズにおいて，各リンクを同時に通る通信の本数をそのリンクに対する経路競合数と定義する．経路競合数が 1 であれば，そのリンク上で経路競合が発生していないことを表している．各サーバ間通信において，途中で通るそれぞれのリンクの経路競合数の最大値を算出し，これを最大経路競合数と呼ぶ．この最大経路競合数の逆数を平均した数値を通信フェーズにおけるスループットと定義し，このスループットを全フェーズで平均した数値を All-to-all 通信時のスループットと定義する．[1]

4.2 評価

4.2.1 実験 1: 提案手法の検証

従来の Fat-Tree とラテン方陣 Fat-Tree でのシフト通信方式と本手法のスループットを比較する．

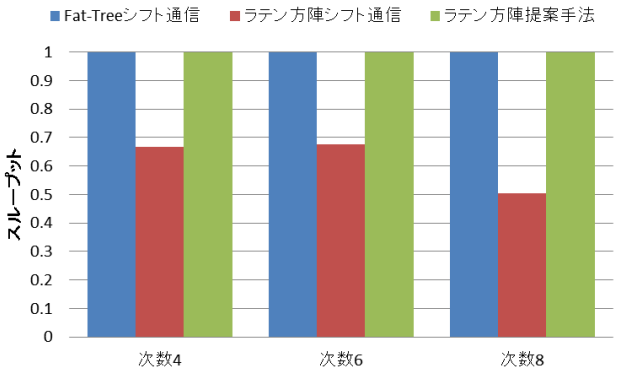

図 11 従来方式と提案方式の比較

従来の Fat-Tree でシフト通信パターンを採用すると，スループットが 1.0 であることから，競合なく通信できている．これに対して，ラテン方陣 Fat-Tree でシフト通信パターンを採用してしまうと，スループットが低下し，競合が起こっていることがわかる．提案手法では，スループットが 1.0 であり，提案手法を採用することで，ラテン方陣 Fat-Tree でも競合なく All-to-all 通信ができていることが確認された．

4.2.2 実験 2: ランダム通信パターンの性能検証

ランダム通信パターンを用い，従来の Fat-Tree とラテン方陣 Fat-Tree でのスループットを比較する．また，従来の Fat-Tree で選択する台数の比率をラテン方陣 Fat-Tree と同程度になるように調整した場合のスループットも比較する．

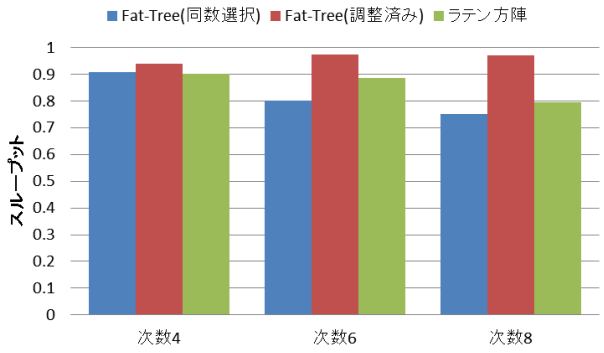

図 12 ランダム通信パターンでの性能

いずれのトポロジー構造でもランダム通信パターンを用いることで一定のスループット(0.8 程度)を期待することができるが，ランダム通信パターンでは競合を完全になくすことはできない．ネットワークの規模が大きくなると性能は低下する傾向にある．

使用サーバ数が同程度の場合，従来の Fat-Tree はラテン方陣 Fat-Tree よりもスループットが悪いという結果となった．これは従来の Fat-Tree がラテン方陣 Fat-Tree に比べて全体のサーバ数が少ないために通信が密になることが原因と考えられる．従来の Fat-Tree において，サーバ選択率をラテン方陣 Fat-Tree の選択率と同程度になるように調整した場合，スループットは従来の Fat-Tree の方が良いという結果となった．

4.3 結論

ラテン方陣 Fat-Tree は少ないスイッチ数でより多くのサーバを接続できるものの，トポロジーの性質としては従来の Fat-Tree よりもベースの性能が低い．しかし，4.2.1 項で検証したようにサーバ選択と通信順序を工夫することで All-to-all 通信において従来の Fat-Tree と同等の性能を維持できる．

5. 関連研究

文献[6], [7]では，単一の Fullmesh における All-to-all 通信の経路競合回避方法が提案されている．単一の Fullmesh は接続可能台数が少なく，実用性が低い．実際，次数 18 のスイッチであれば，つなぐことのできるサーバ台数は 342 台である．

また，スケーラブルなトポロジー構造として，Dragonfly というトポロジー構造が提案されている[8], [9], [10]．Dragonfly はランダム通信を行う場合に高性能で競合が少ないことが知られており，MPI レベルでの経路競合回避方法は考えられていないため，各フェーズでの経路競合を完全に避ける手法にまで言及したものはない．

さらに，上記のいずれの文献においても部分的に切り出して All-to-all 通信を行う手法についての言及はない．

6. おわりに

本稿によって，ポート利用効率が最適なラテン方陣 Fat-Tree を用いた大規模分散並列処理環境上で一部のサーバを柔軟に切り出して部分対部分通信を行うことが可能であることが明らかにした．

一般にポート利用効率の向上を追求した場合，サーバ間の密度が小さくなり，競合が発生しやすくなるが，本結果はポート利用効率の向上と競合の回避を両立するものである．

本手法を用いた場合，シフト通信パターンにおける 50%程度のスループット低下やランダム通信パターンにおける 20%程度のスループット低下をなくし，競合の全くない All-to-all 通信を実現できることを示した．

また，コスト面ではある程度のスイッチ数(次数が 18 であれば，100 台以上)を参加させることで，オーバーヘッドは平均 3%程度におさえることができることがわかった．

今後の課題としては，All-to-all 通信に参加させることのできるサーバ台数のバリエーションの制限の緩和がある．現状では，必要な台数と実際に参加させる台数が大きく異なった場合にスイッチ台数のコスト面において無駄が発生してしまう．この問題を解決するため，より細かく調整できるような台数指定を可能とする手法改良が必要となる．また，ラテン方陣 Fat-Tree にも従来の Fat-Tree と同様に 3 段以上の構成としてより多くのサーバをつなげる手法が存在するが [2]，このトポロジー構造に対する経路競合のない All-to-all 通信を実現する手法については知られていない．したがって，こちらの解決も今後の課題である．

参考文献

1) 中島 耕太，三輪 真弘，“スイッチ台数の削減と高い All-to-all 通信性能を両立する多層 Fullmesh トポロジーの提案，” 情報処理学会研究会報告, 2014.
2) M. Valerio, L. E. Moser and P. M. Melliar-Smith: “Recursively Scalable Fat-Trees as Interconnection Networks,” IEEE 13th Annual International Phoenix Conference on Computers and Communications, 1994.
3) InfiniBand Architecture Specification Release 1.3, Infiniband Trade Association, http://www.infinibandta.org.
4) 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.
5) A. A. Adrian, S. Reuben: “An Introduction to Finite Projective Planes,” New York, Holt, Rinehart and Winston, 1968.
6) E. Totoni and L. V. Kale: “ACM SRC poster: optimizing all-to-all algorithm for PERCS network using simulation,” Proceedings of the

2011 companion on High Performance Computing Networking, Storage and Analysis Companion, 2011.
7) E. Totoni, A. Bhatele, E.J. Bohm, N. Jain, C. L. Mendes, R. M. Mokos, G. Zheng, and L. V. Kale: “Simulation-based Performance Analysis and Tuning for a Two-level Directory Connected System,” Proceedings of IEEE 17th International Conference on Parallel and Distributed Systems, 2011.
8) J. Kim, W.J. Dally, S. SCott, D. Abts: “Technology-Driven, Highly-Scalable Dragonfly Topology, ” ISCA '08. 35th International Symposium on Computer Architecture, 2008.
9) N. Jain, A. Bhatele, N. Xiang, N. J. Wright, L. V. Kale: “Maximizing Throughput on a Dragonfly Network,” Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2014.
10) B. Prisacari, G. Rodriguez, P. Heidelberger, D. Chen, C. Minkenberg, T. Hoefler: “Efficient Task Placement and Routing of Nearest Neighbor Exchanges in Dragonfly Networks,” Proceedings of the 23rd international symposium on High-performance parallel and distributed computing, 2014.