10-IS01

動的な集団通信アルゴリズム選択技術の実アプリケーションにおける効果の検証

黒川　原佳（理化学研究所）

概要
バッチ型で利用する共同利用大規模並列計算機において，充填率を考慮したスケジューリングを行う場合，割り当てられる計算ノード間の距離が実行の度に変化するため，従来の静的な手法では最適な集団通信アルゴリズム選択を行えない．本研究では，実行中の実測値を用いてアルゴリズム選択を動的に行う手法を，共同利用大規模並列計算機に適用して効果を検証した．実験の結果，動的なアルゴリズム選択手法によって，状況の変化に対応した選択が行えることを確認した．また，実アプリケーションとして NAS Parallel Benchmarks の FT を用いた実験を行った．その結果，動的なアルゴリズム選択手法では，実行時に遅いアルゴリズムまで試すことによるオーバヘッドが，実用上問題になる程度に大きくなる場合があることが判明した．

1. 研究の目的と意義

バッチ型で利用する共同利用の大規模並列計算機において，出来るだけ安定した並列処理性能を得るための技術について効果を検証する．バッチ型利用とは，計算機を対話的に利用するのではなく，処理して欲しい内容を記述したファイルをジョブとして申請し，計算機の空き状況に応じて計算能力が割り当てられ，処理されるのを待つ，という，間接的な計算機の利用形態である．近年，パーソナルコンピュータの普及，及び性能向上によって対話的に利用可能な計算能力は大幅に増大したが，最先端の科学技術計算おける中～大規模計算では，依然として複数の利用者がバッチ型で共同利用する大規模な計算機が必要である．現在利用されている大規模な計算機の構成としては，複数の計算ノードで構成された分散メモリ型の並列計算機が最も一般的である．

この並列計算機をバッチ型で利用する場合，申請されたジョブを計算ノードに割り当てるジョブスケジューラの設定が，計算機の総合的な処理速度，及び個々のジョブにおけるプログラムの実行速度に大きく影響する．総合的な処理速度は，出来るだけ空いている計算ノードが少なくなるようにジョブを割り当てる，ジョブの充填率を重視した割り当てによって向上すると期待される．一方，個々のジョブにおけるプログラムの実行速度は，特に複数の計算ノードを利用する並列計算では，計算ノード間の通信性能に依存する．さらにこの通信性能は，通信する計算ノード同士の，ネットワーク上での距離による影響が大きい．計算ノード同士の距離が近い場合はほとんど性能が変動する要素が無いが，距離が遠い場合，到達するまでの通信遅延時間が長くなるだけでなく，他の通信と経路が競合することによる通信性能の大幅な低下が発生しやすくなる．

特に，並列プログラムにおける全プロセスによる総和の計算や全対全のコピー等の，集団通信と呼ばれる通信では，この転送性能の低下が大きな問題となる．集団通信には複数の実装アルゴリズムが用意されており，状況に応じて最適なアルゴリズムを選択して使用する．従来は，事前に十分な調査を行うことにより，使用するプロセス数と転送するメッセージサイズから最適なアルゴリズムを選択している．しかし通信性能が安定しない環境では，同じプロセス数，メッセージサイズでも事前の調査時とは最適なアルゴリズムが変化する可能性が高い．そのため，実行時の状況に応じて適応的にアルゴリズムを選択する仕組みが必要となる．

そこで本稿では，集団通信の実装アルゴリズム

を実行時の状況に応じて選択する技術を有するソフトウェア STAR-MPI を用いることで，ジョブの充填率を重視したスケジューラによる並列計算機でも最適なアルゴリズムを選択できると予想し，実験によって効果を検証する．

2. 当拠点公募型共同研究として実施した意義

(1) 共同研究を実施した大学名
九州大学
(2) 共同研究分野
大規模情報システム関連研究分野
(3) 当公募型共同研究ならではという事項など
スーパーコンピュータの運用に携わる２つの組織の研究者間で情報や知見の共有を行えた。

3. 研究成果の詳細

3.1 集団通信アルゴリズム

集団通信とは，前述の通り並列プログラムの全プロセスが参加し，全プロセスからのデータの集約や，1 対全，全対全のデータコピー等，科学技術計算で多用される定型の通信パターンである．このうち本稿では，Alltoall と呼ばれる通信について扱う．これは，各プロセスが他のプロセスに対して，自分の所有するデータのうち相手のプロセスに対応する部分を送信するものである(図 1)．**この通信は，FFT(Fast Fourier Transform)や行列の転置等で頻繁に用いられる．**

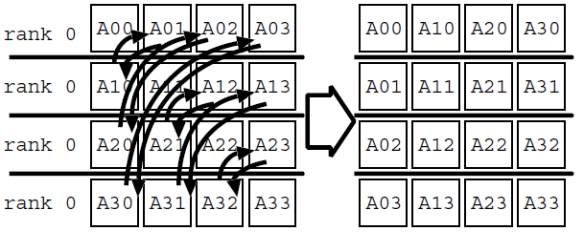

図 1 Alltoall 通信

集団通信の性能は並列プログラムの処理速度に大きく影響するため，それぞれの集団通信について多数の実装アルゴリズムが提案されている．例えば Alltoall で最も簡単な実装である Simple Spread アルゴリズムは，各プロセスが必要な送信命令を全て発行した後，必要な受信命令を発行して受信するものである．このアルゴリズムは，多数の通信を同時に進行させることが出来るが，一つのプロセスに対して複数の送信が同時に発生するため衝突を起こしやすく，特にメッセージが大きい場合に速度が低下する．

一方 Ring アルゴリズムは，通信経路での衝突を回避するため，同じプロセスに送信データが集中しないよう同期を取りながら通信を進める．この場合，衝突の影響は軽微となるが，同期のコストが問題となるため，比較的大きなメッセージに適している．

また，Bruck アルゴリズムは，複数の宛先のデータをまとめて転送することにより，少ない送受信発行回数で Alltoall を実現するものである．例えば 8 プロセスで通信を行う場合，プロセス 0 は最初の送信でプロセス 1 に対して，プロセス 1 宛のデータだけでなく，プロセス 3，5，7 のデータも送信する．プロセス 1 は，2 回目の送信でプロセス 3 に対して，自分のデータ，及びプロセス 0 から受信したデータから，プロセス 3 宛とプロセス 7 宛のデータを抽出して送信する．これにより，各プロセスで必要な送受信回数は 3 ($= \log_2 8$) 回となる．しかし，一回の送受信データ量が 4 倍になるため，Alltoall 全体での総通信量は増加する．そのため，Bruck アルゴリズムは比較的小さいメッセージに適している．

このように様々な集団通信のアルゴリズムが提案されており，しかも状況によって相互の優劣が変わる．上記のメッセージサイズだけでなく，プロセス数や通信性能等によっても，最適なアルゴリズムは変動する．そのため，状況に応じて適切なアルゴリズムを選択することが重要である．

3.2 メタジョブスケジューラ

従来，並列計算機上のジョブスケジューラの多くは，使用する計算ノード等の規模に応じてジョブキューと呼ばれる待ち行列を複数用意し，それ

ぞれ独立に処理を進める方式を採用していた．この方式では，それぞれのジョブキューに対してあらかじめ計算ノードが対応づけられているため，プログラムの規模が同じであれば，割り当てられる計算ノードの配置は変動が少なく，通信性能も比較的安定している．しかしながら，各ジョブキューに対応づけられる計算ノードの比率と，実際に各ジョブキューに投入されるジョブ数の比率に大きな相違がある場合，空いている計算ノードがあるにも関わらず待たされるジョブキューがある等，非効率な運用となり，ジョブの充填率が低下する．また，一般に共同利用計算機に投入されるジョブは多種多様であり，各ジョブキューへの適切な計算ノードの配分は困難である．

一方，理化学研究所の RICC(Riken Integrated Cluster of Clusters)では，メタジョブスケジューラというスケジューラが用いられている[1]．これは，ジョブキューによる計算ノード配分ではなく，各ジョブの属性に従って空いている計算ノードを割り当てていくものである．利用者は，ジョブの属性として，使用するコア数やメモリ量，予想される計算時間，使用するソフトウェアライセンス等を指定する．メタジョブスケジューラは，これらのジョブを時系列で管理する予約表を内部で保有しており，ジョブキューの先頭のジョブから順に，ジョブの属性に従ってこの予約表に割り付けていく．

予約表の割り当て時刻になったら，該当するジョブを計算ノードに割り当てて実行を開始する．

このスケジューラは，基本的には，計算ノードの位置によらず，空いている計算ノードを必要数集めてジョブに割り当てることが出来るため，ジョブの充填率が向上し，効率の良い運用が可能となる．しかしながら，同じ規模のプログラムでも計算ノードの配置が大きく変動するため，毎回通信性能が変わる．特にネットワーク上で離れた距離にある計算ノードが割り当てられた場合，他の通信との通信経路の競合が発生しやすくなり，実行中でも通信性能が変動する．

3.3 動的集団通信アルゴリズム選択技術 STAR-MPI

STAR-MPIは，MPI(Message Passing Interface)を用いた並列プログラムにおいて，集団通信のアルゴリズムを実行時に自動選択する技術である[2]．MPIとは，事実上の世界標準として広く用いられている並列プログラミングインタフェースであり，

STAR-MPIは，このMPIの上に構築されている．利用者は，プログラム中の MPIの集団通信関数名をSTAR-MPIにおける集団通信関数名に置き換えた上で，STAR-MPIのソースプログラムと自分のプログラムを通常の MPIプログラムと同様にコンパイルし，リンカで結合することにより，STAR-MPIの機能を利用できる．

STAR-MPIの処理は，以下の 2つのフェーズに分けて行われる．

Learningフェーズ:

集団通信が呼ばれると，利用可能なアルゴリズムのうちの一つを用いて実行し，結果を返す．その際所要時間を計測し，記録する．全アルゴリズムについて規定回数の計測が終了するまで，各集団通信呼び出しに対してこのフェーズを実行する．

全アルゴリズムの計測が完了すると，それらの所要時間の記録から最も高速なアルゴリズムを選出し，次の Monitoringフェーズで使用するアルゴリズムとする．なお，アルゴリズムの選出に用いる各アルゴリズムの所要時間としては，全プロセスの所要時間の平均値を用いる．

Monitoringフェーズ:

選出されたアルゴリズムを用いて集団通信を行う．このフェーズは Learningフェーズが終了してアルゴリズムが選択された後，各集団呼び出しに対して実行する．

実際の計算環境では実行中に状況が大きく変化することも考えられるため，定期的に集団通信の所要時間と Learningフェーズで得られた所要時間を比較する．もし，計測した時間が

Learning フェーズ時の所要時間から大きく変動した場合，他のアルゴリズムの方が高速となった可能性があるので，再度 Learning フェーズに入り，アルゴリズムを選択する．

STAR-MPI には，選択対象の各アルゴリズムが MPI により実装されている．例えば Alltoall 通信のアルゴリズムとしては，初期設定時に Simple Spread(以下 Simple と表記)，Pairwise(Pair)，Pairwise with Light-Barrier(PairLB)，Pairwise with MPI-Barrier(PairMB)，Pairwise with One-Barrier(PairOB)，Ring(Ring)，Ring with Light-Barrier(RingLB)，Ring with MPI-Barrier(RingMB)，Ring with One-Barrier(RingOB)から選択されるようになっている．さらに STAR-MPI は，実行環境における MPI ライブラリの MPI_Alltoall 関数も，選択肢として用いる．

3.3 充填率を考慮したジョブスケジューラによる環境における STAR-MPI の効果

3.3.1 実験環境

実験に使用したのは，理化学研究所の RICC のうち超並列 PC クラスタと呼ばれるシステムである．このシステムの計算ノードは Intel Xeon 5570 (2.93GHz，4CPU コア)を 2 基と，12GB の主記憶を搭載している．また，計算ノード間は InfiniBand で接続されている．計算ノード間の接続に用いられているトポロジは 2段構成の Fat-Tree と呼ばれる形である．Fat-Tree は複数段の木構造の形をしており，それぞれの段にスイッチが複数配置される．このうち最下段のスイッチはリーフスイッチと呼ばれ，計算ノードと直接接続する．一方下段から上段へは，下段の各スイッチと上段の各スイッチを全対全で接続する．このため，上段のスイッチを複数用意することにより，上段と下段の間の通信経路における競合の頻度を低減できる．RICC の場合，リーフスイッチ 53 台で 1040 台の計算ノードと接続する．一方，上段スイッチは 2 台あり，各リーフスイッチが各上段スイッチと 2 本ずつの線で接続する構成となっている．

この環境において，まず STAR-MPI の効果を検証するため，Alltoall 通信を 200 回呼び出すプログラムを用いて所要時間の計測を行った．このプログラムは STAR-MPI のベンチマークプログラムとして提供されているものである．このプログラムを，プロセス数とメッセージサイズを変えながら RICC のメタジョブスケジューラに投入した．それぞれのプロセス数とメッセージサイズの組み合わせについて 10 回ずつジョブを投入し，回毎の各アルゴリズムの所要時間，選択されたアルゴリズム，全体的な平均所要時間，及びオーバーヘッドについて計測した．RICC では計算ノードの単位でジョブを割り当てる．1 台の計算ノードには 8CPU コアが搭載されているので，例えば 64 プロセス時で 8 台，128 プロセス時で 16 台の計算ノードが割り当てられる．

なお，RICC のメタスケジューラには，16 台以下の計算ノードを使用するジョブについて，全部の計算ノードが 1台のリーフスイッチに納まるような割り当てを指示する機能がある．この機能を使った場合，計算ノードが割り当てられるまでの待ち時間が長くなるが，計算ノードの相対的な位置が毎回ほとんど同じであるため，通信性能の変動が少ないと予想される．そこで，この機能を使った場合と使わなかった場合で計測結果を比較する．

さらに，実アプリケーションにおける効果を検証するため，NAS Parallel Benchmarks の FT について，内部の Alltoall 通信を STAR-MPI に置き換えることによる所要時間の変化を計測した．実験に用いた Class は C である．

3.3.2 Alltoall 通信実験結果

Alltoall 通信を 200 回呼び出すプログラムについて，128 プロセス，メッセージサイズ 1MB で，使用する全計算ノードを同じリーフスイッチに配置するよう指定した場合の結果を図 2 に，その指示を行わなかった場合の結果を図 3 に，それぞれ示す．また，同様の実験をメッセージサイズ 1KB

で行った場合の結果を図 4，図 5 に，プロセス数 64，メッセージサイズ 1MB で行った場合の結果を図 6，図 7 に，プロセス数 64，メッセージサイズ 1KB で行った場合の結果を図 8，図 9 に，それぞれ示す．

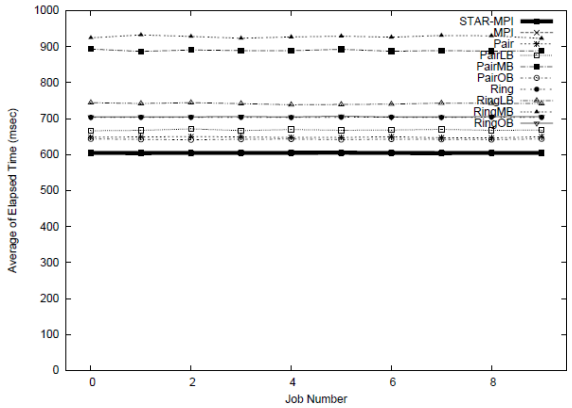

図 2　同一スイッチでの所要時間
(128 プロセス，1MB)

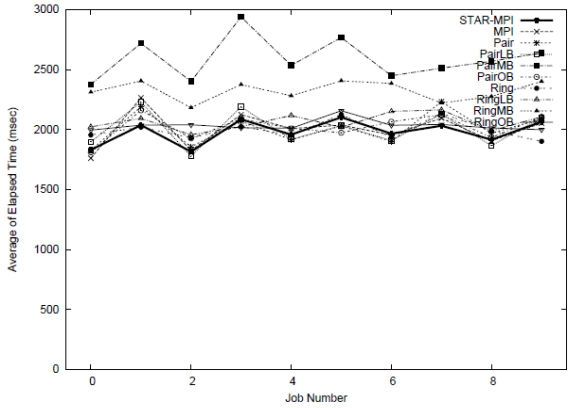

図 3　複数スイッチでの所要時間
(128 プロセス，1MB)

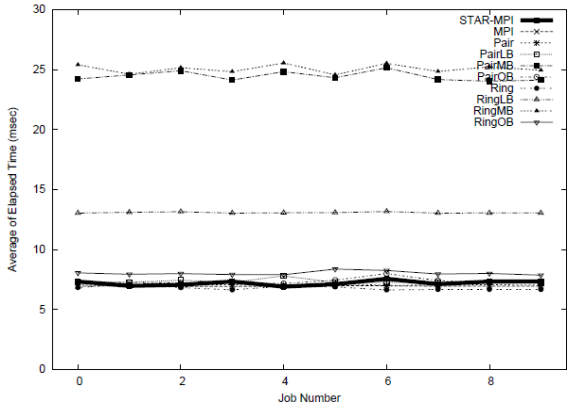

図 4　同一スイッチでの所要時間
(128 プロセス，1KB)

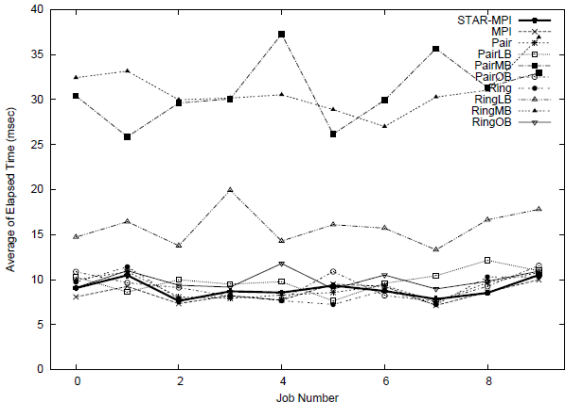

図 5　複数スイッチでの所要時間
(128 プロセス，1KB)

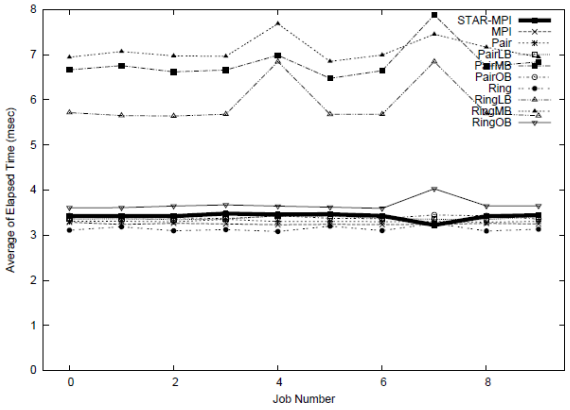

図 6　同一スイッチでの所要時間
(64 プロセス，1MB)

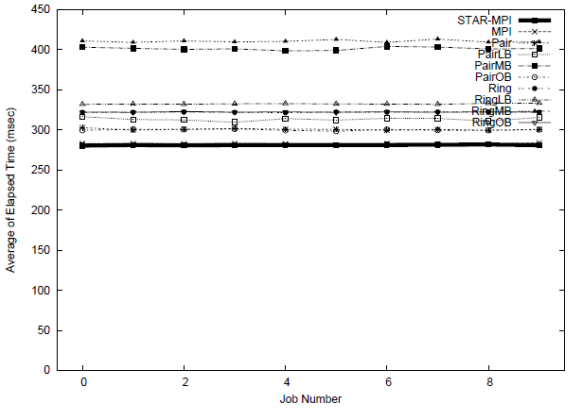

図 7　複数スイッチでの所要時間
(64 プロセス，1MB)

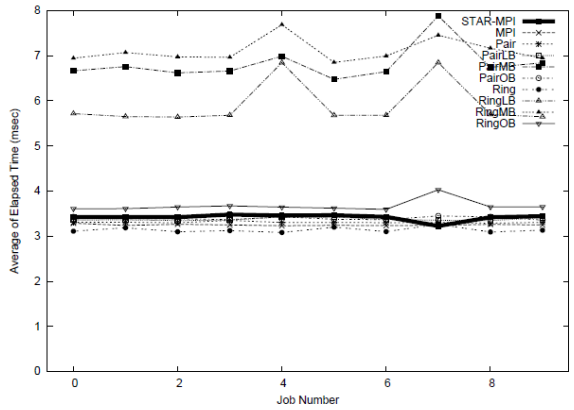

図 8　同一スイッチでの所要時間
(64 プロセス，1KB)

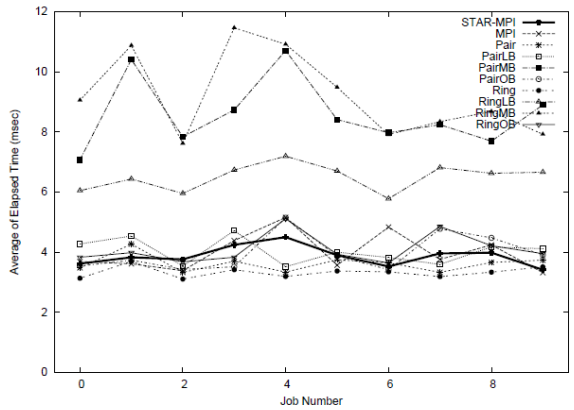

図 9　複数スイッチでの所要時間
(64 プロセス，1KB)

図の横軸は投入したジョブの番号，縦軸は平均所要時間である．計測結果としては，最初の Learning フェーズにおける各アルゴリズムの平均所要時間(Pair～RingOB)，MPI の MPI_Alltoall 関数の平均所要時間(MPI)，及び STAR-MPI の Monitoring フェーズにおける平均所要時間(STAR-MPI)である．STAR-MPI は，最初の Learning フェーズで最も速かったアルゴリズムを選択するので，基本的に毎回，図中の各アルゴリズムのうち一番下のもので Monitoring フェーズを実行する．ただし，Monitoring によるオーバヘッドや実行中の性能の変動，さらに，実行途中での再度の Learning フェーズによるアルゴリズム切り替え等により，選択したアルゴリズムの所要時間と STAR-MPI の所要時間は一致しない．

全計算ノードが同一スイッチに配置された場合，各アルゴリズムの性能は毎回ほとんど変動が無い．そのため STAR-MPI も，毎回ほとんど同じ選択をしている．例えば図 2 では，MPI_Alltoall 関数を最適アルゴリズムとして選出している．このような状況であれば，STAR-MPI を利用しなくても，事前に十分な調査をすればプロセス数とメッセージサイズだけで最適なアルゴリズムを選択することができると考えられる．

一方，同一スイッチへの配置を指示しなかった場合，計算ノードが複数のスイッチに跨って配置されるため，通信時間が大きく変動する．その結果，図 3，図 5，図 7，図 9 に示す通り，各アルゴリズムの性能も大きく変動している．例えば 図 3 の Job Number 0 で最速だった MPI_Alltoall 関数は，Job Number 1 では最速アルゴリズム(Ring)よりも 10%程度遅くなっている．このように毎回通信性能が変動する場合，従来のプロセス数とメッセージサイズだけでは最適なアルゴリズムを選択できず，STAR-MPI のような動的な手法が必要となる．

3.3.3　NAS Parallel Benchmarks FT における効果に関する実験結果

NAS Parallel Benchmarks FT (Class=C)について，内部の Alltoall 通信として MPI_Alltoall を用いた場合と，STAR-MPI の Alltoall を用いた場合の所要時間を計測した結果を表 1 に示す．

表 1　FT(Class=C)の所要時間

#procs	MPI_Alltoall(sec)	STAR-MPI(sec)
16	502.7	481.5
32	264.4	269.9
64	189.8	189.5
128	131.0	141.3
256	74.1	97.3

表中で #procs はプロセス数を示す．FT の反復回数は 300 回で固定し，所要時間としては 5 回計測した平均値を用いた．

プロセス数が 16 の場合は STAR-MPI で動的にアルゴリズム選択が行うことにより，所要時間を短縮できている．しかしそれ以外のプロセス数では，

かえって STAR-MPI を用いることによって所要時間が増加している．これは，今回の実験においてプロセス数16以外ではほとんどの場合，STAR-MPI が内部で試したアルゴリズムの中で MPI_Alltoall が最速であったためである．STAR-MPI では，Learning フェーズで利用可能な全てのアルゴリズムを試す．このフェーズで遅いアルゴリズムを試すことによる所要時間の増加分が最適化のオーバヘッドとなる．このオーバヘッドにより，MPI_Alltoall が選択される状況では，STAR-MPI を用いない方が STAR-MPI を用いるよりも高速となる．

3.3.4 考察

3.3.4.1 計算ノードの配置と選択されたアルゴリズムの関係

今回の実験で計算ノードの配置がアルゴリズムの選択に与えた影響を調べるため，図 3 の実験における各 Job Number について，使用したリーフスイッチの数と，リーフスイッチから上段のスイッチに接続するポートの利用頻度の偏りを調査し，それらに対して選択されたアルゴリズムの関係を確認した．RICC の Fat-Tree では，リーフスイッチから上段スイッチへの接続ポートが 4 ポートあり，それぞれの通信について送信先の計算ノードの番号を 4 で割った余りを，その通信で使用するポート番号としている．そのため，使用する計算ノードの番号を 4 で割った余りの分布を，使用ポート番号の偏りを示す情報として用いる．

表 2 に，各 Job における計算ノードの配置に関する情報と，選択されたアルゴリズムを示す．ここで Port 0～3 は，上段スイッチへの接続ポート 0～3 を経由してデータが転送される計算ノードの数である．まだサンプルの数が少ないが，傾向として，上段のスイッチの利用頻度のばらつきによって，選択されるアルゴリズムが変化する様子が見える．ただし，実験環境では計測に用いたプログラムの他にも多数のプログラムが動作しており，それらによる影響も否定できないので，さらに調査が必要である．

表 2 計算ノードの配置と選択されたアルゴリズム

Job	Leaf Sw.	Port 0	Port 1	Port 2	Port 3	Algo.
0	9	6	2	4	4	MPI
1	10	6	2	3	5	Ring
2	9	6	2	4	4	PairLB
3	10	6	2	3	5	RingOB
4	9	7	2	3	4	Pair
5	10	4	3	4	5	PairOB
6	9	7	2	3	4	Pair
7	10	4	3	4	5	RingOB
8	9	7	2	3	4	PairLB
9	10	4	3	4	5	Ring

3.3.4.2 STAR-MPI のオーバヘッド

STAR-MPI の最適化に要するオーバヘッドとして，3.3.2 節で示した実験で同一スイッチへの配置を指定しない場合のうち，Job Number 0 の場合を例として示す．この場合，Learning フェーズでの Alltoall 通信の平均所要時間は約 2,031ms，Monitoring フェーズでの平均所要時間は 1,829ms であった．一方，選択された MPI_Alltoall の Learning フェーズにおける平均所要時間は 1,761ms であるので，1 回あたりのオーバヘッドとして Learning フェーズで約 15%，Monitoring フェーズで約 4%を要したと考えられる．このうち特に Learning フェーズは，遅いアルゴリズムを試す時間が含まれるため，オーバヘッドが大きくなっている．

なお，3.3.3 節の FT による実験で，表 1 のうちプロセス数が 256 の場合，STAR-MPI を用いることによる性能低下が約 31%と，非常に大きくなっている．これは，選択されたアルゴリズムが MPI_Alltoall であり，さらに他の全てのアルゴリズムに対して MPI_Alltoall の所要時間が 2～3 倍高速であったため，上記のオーバヘッドが大きくなったと考えられる．

参考文献

[1] 宇治橋善史，久門耕一，計算センタシステム高稼働率

とジョブ待機時間短縮を実現するジョブスケジューラ，情報処理学会創立50周年記念全国大会，3A-1，2010

[2] A. Faraj, X. Yuan, and D. Lowenthal, "STAR-MPI:Self Tuned Adaptive Routines for MPI Collective Operations," In Proceedings of the 20th ACM International Conference on Supercomputing (ICS06), 2006.

4. これまでの進捗状況と今後の展望

充填率を重視してランク配置を行うジョブスケジューラを用いた環境で，STAR-MPIを用いることにより，通信性能の変動によらず最適なアルゴリズムが選択されることを確認した．また，実用上の問題として，Learningフェーズで遅いアルゴリズムを試すことによるオーバヘッドが，無視できないほど大きくなる場合があることも判明した．今後，遅いアルゴリズムを選択肢から外す等の効率化により最適化のオーバヘッドを低減し，より実用性の高いアルゴリズム選択技術の構築を目指す．

5. 研究成果リスト

(1) 学術論文（投稿中のものは「投稿中」と明記）
(2) 国際会議プロシーディングス
(3) 国際会議発表
(4) 国内会議発表
南里豪志，黒川原佳，充填率を考慮したジョブスケジューラによるランク配置に対する集団通信アルゴリズム自動選択技術の効果，第14回環瀬戸内応用数理研究部会，2011.1（発表予定）．
(5) その他（特許，プレス発表，著書等）