Vol.2015-HPC-150 No.40
2015/8/6

RDMAモデル向け集団通信インタフェースの実装技術に関する研究開発

杉山　裕宣[†1]　南里　豪志[†1,†2]

概要：近年，多くの大規模並列計算機はRDMA(Remote Direct Memory Access) モデルを基盤としたインターコネクトを採用している．そこで，科学技術計算等で多用される集団通信の実装においても，従来のようなメッセージパッシングモデルに基づいた方法ではなく，RDMA モデルに基づいた方法が重要となる．RDMA モデル上での集団通信の実装では，通信の事前処理のためのオーバーヘッド削減が性能向上の鍵の一つである．そのための手段の一つとして，MPI Forum で現在採用に向けた議論が進められている永続型集団通信インタフェースの利用が挙げられる．RDMA モデルでの永続型集団通信インタフェースは，いくつかの項目で実装手段に複数の選択肢が考えられる．さらに，メッセージサイズやプロセス数によって最適な選択肢は異なる．そこで本稿では，特に通信命令に関して，状況に応じた最適な実装手段を選択するための性能モデルの作成を行い，そのモデルによる予測値と実機の実測値を比較した．実験の結果，64 プロセスにおいて，性能モデルによる実装手段の優劣を見積もる事が出来る事を示した．

1. 背景

近年多くの大規模並列計算機はRDMA（Remote Direct Memory Access）モデルを基盤としたインターコネクトを採用している．RDMA モデルは，メッセージパッシングモデルとは異なり，遠隔プロセスのメモリ領域に対して直接アクセスする片側通信命令により通信を指示出来る．そこで，科学技術計算等で多用される集団通信の実装においても，従来のようなメッセージパッシングモデルに基づいた方法ではなく，RDMA モデルに基づいた方法が重要となる．

RDMA モデルでは，片側通信を可能とするために，通信の事前準備として，通信対象となるメモリ領域を登録し，通信命令発行側のプロセスは遠隔プロセスの登録先メモリアドレス情報を取得しておかなければならない．この準備処理は集団通信の呼び出し毎に行うには無視できないオーバーヘッドが生じる．一方，MPI の規格に関する議論と策定を行う MPI Forum [1] では現在，通信の準備を別の関数として独立させた永続型集団通信インタフェースについて，採用に向けた議論が進められている．永続型集団通信インタフェースは，同一パラメータの集団通信を繰り返し実行する際に一度行った準備の結果を再利用する事でオーバーヘッドの低減を可能にするもので，RDMA モデルによる集団通信に適していると考えられる．

そこで我々は，RDMA モデルでの永続型集団通信インタフェースの実装技術について検討する．RDMA モデル上での永続型集団通信インタフェースでは，様々な項目に関して実装手段に複数の選択肢が考えられる．さらに，メッセージサイズやプロセス数によって，最適な選択肢は異なるため，適切な実装手段を選択する事が難しい．そこで本研究では，状況に応じた適切な実装手段の選択を目的とする．メッセージパッシングモデルでは，最適アルゴリズムの選択手法について [3] [4] 等で議論されている．しかし，RDMA モデルはメッセージパッシングモデルとは実装手段の選択肢が異なるため，それらの手法をそのまま適用する事は出来ない．そこで，RDMA モデルの特徴を考慮した性能モデルを用いる事で，最適な実装手段の選択につなげる．

本稿では，RDMA モデルでの永続型集団通信における実装上の複数の選択肢についてその特徴を述べた後，永続型ブロードキャスト通信を例として，特に通信命令に関して，状況に応じた最適な実装手段を選択するための性能モデルの作成を行い，そのモデルによる予測値と実機による実測値を比較する．

[†1] 九州大学
Kyushu Uniersity
[†2] 独立行政法人科学技術振興機構　戦略的創造研究推進事業
Japan Science and Technology Agency(JST), Core Research for Evolutional Science and Technology(CREST)

2. Remote Direct Memory Access モデル

RDMA モデルの片側通信命令として，一般には，遠隔プロセスのメモリ領域に対して直接データを書き込む put，遠隔プロセスのメモリ領域から直接データを読み込む get が用意されている事が多い．put 命令は図 1 に示す通り，命令発行側のプロセスのメモリ領域から遠隔プロセスのメモリ領域に対し直接データを転送する．命令発行側のプロセスは，転送先からの完了通知をもって転送の完了を検知する事が出来る．一方で，get 命令は図 2 に示す通り，遠隔プロセスのメモリ領域から命令発行側のプロセスのメモリ領域に対し直接データを転送する．命令発行側のプロセスは，自分のメモリ領域に対するデータの書き込みが完了した時点で転送の完了を検知する事が出来る．

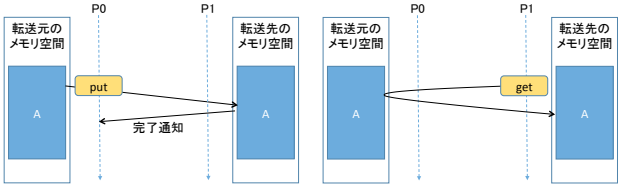

図 1　put 命令によるデータ転送

図 2　get 命令によるデータ転送

これらの片側通信を可能とするために，通信命令発行側のプロセスは，遠隔プロセスの通信対象領域に関する情報を知っておく必要がある．そのため，RDMA モデルでは通信の準備として，通信命令発行側のプロセスと遠隔プロセスの双方が通信対象となるメモリ領域を登録し，通信命令発行側のプロセスは遠隔プロセスの登録先メモリアドレス情報を取得しておかなければならない．これらの処理は，集団通信の呼び出し毎に行うには無視できないオーバーヘッドが生じるため，その影響を低減させる事が，RDMA モデル上での集団通信高速化の鍵の一つである．

3. 永続型通信インタフェース

並列プログラムでは，ループ中で同一の引数を持った通信を繰り返し実行する事が頻繁にある．特に RDMA モデルによる通信では，それぞれの通信で準備処理を行う必要があり，オーバーヘッドが問題となる．しかし，一度行った準備処理の結果を保持しておき，再利用する事が出来れば，同一引数の通信を繰り返し実行するプログラムでオーバーヘッドの影響を低減出来る可能性がある．

そこで MPI では，永続型通信が定義されている．永続型通信とは，通信を行うために必要な準備を別の関数として通信関数から独立させ，そこで取得した情報を保持しておき，保持した情報を基に通信を行うものである．MPI では，永続型通信のための関数として，通信準備関数，通信開始関数，通信完了待ち関数が用意されている．通信準備関数では，与えられたパラメータにより通信の準備処理を行い，その結果を，永続的通信要求として指定されたアドレスに保存する．通信開始関数，通信完了待ち関数では，指示された永続的通信要求による通信の開始と完了待ちをそれぞれ行う．

一方，このインタフェースを集団通信に適用した永続型集団通信の提案が 2015 年 6 月の MPI Forum[1] でされ，採用に向けた議論が進められている．永続型集団通信インタフェースでは，永続型一対一通信インタフェースと同様に，通信準備関数，通信開始関数，通信完了待ち関数が用意され，同一引数の集団通信を繰り返し実行するプログラムで準備処理によるオーバーヘッドの低減が期待される．

そこで本稿では，RDMA モデル上での永続型集団通信インタフェースの実装技術に関する検討と評価を行う．

4. RDMA モデル上での永続型集団通信インタフェースにおける実装上の課題

本研究では，RDMA モデル上での永続型集団通信インタフェースの実装技術について，様々なインターコネクトにおける共通の課題に対して検討をするため，IBverbs のような，特定のインターコネクト上の RDMA モデルを前提としていない．そこで本研究では，インターコネクトを抽象化した通信ライブラリである，ACP[2] を用いる事とする．ACP[2] は，InfiniBand と Tofu，Ethernet のそれぞれで実装された，特定のインターコネクトに依存しない RDMA モデルを提供する．

RDMA 上での永続型集団通信インタフェースの実装に関して，通信準備関数では，RDMA モデル上での通信に必要な，通信対象領域の登録と遠隔プロセスの登録先メモリアドレス情報の取得，さらに集団通信アルゴリズムを構成する，依存関係を持った複数の通信のリストを作成し，その結果を永続的通信要求として指定したアドレスに保存する．通信開始関数と通信完了待ち関数では，指定した永続的通信要求による通信の開始と完了待ちを行う．これらの中で，特に性能面で重要となる通信開始関数と通信完了待ち関数について，実装する上で様々な項目に関して複数の選択肢が考えられる．例えば，アルゴリズム，通信命令，親子プロセス間の同期方法，パイプライン転送，非ブロッキング通信の推進方法が挙げられ，これらの項目においてそれぞれ適切な選択肢を選ぶ必要がある．以下に，各項目に関してそれぞれの特徴を述べる．

4.1 アルゴリズム

現在，集団通信の実装には様々なアルゴリズムが提案されており，その種類は二項木，二分木，n 項木，n 分木，Ring，Bruck など，多岐にわたる．例えば，二項木アルゴリズムによるブロードキャスト通信は，ステップ 0 で木の根にあたるルートプロセスが別の一つのプロセスに対しデータを転送する．次のステップでは，ステップ 0 でデータを

受け取ったプロセスとルートプロセスが，それぞれ他のプロセスに対しデータを転送する．このように，既にデータを受け取ったプロセスが別のプロセスに対しデータを送っていく事で，根から葉に向かってデータを伝搬する．二項木アルゴリズムでは，プロセス数 P に対し $\log_2 P$ ステップで通信を完了する．

一方で Ring アルゴリズムは，ステップ 0 で各プロセスは隣接するプロセスに対し，自分の持つデータ転送する．次のステップでは，同じプロセスに対し，ステップ 0 で受け取ったデータを転送する．このように，データをリング状に隣へ隣へと渡していく事で，P-1 ステップで通信を完了する．

このように，各アルゴリズムはそれぞれ異なる特徴をもっており，さらに，メッセージサイズやプロセス数によってアルゴリズム間の優劣は異なる．

4.2 通信命令

RDMA モデルでは一般に，put 命令，もしくは get 命令を使用する事によりデータ転送を指示する．put 命令は，送信側から受信側のメモリ領域へデータを直接書き込む方式である．一方で get 命令は，受信側が送信側のメモリ領域からデータを直接読み込む方式である．ただし，get 命令では，送信側の NIC に get 命令を伝えるために通信が発生し，片道分の通信時間が加わる事となる．このように，RDMA モデルでは一方向のデータ転送を行うための通信命令に二つの選択肢がある．これらは，次節で説明する送受信プロセス間の同期方法と密接に関係するため，双方を合わせて検討する必要がある．

4.3 親子プロセス間の同期方法

put 命令も get 命令も，あるプロセスのメモリ領域のデータを別のプロセスのメモリ領域に転送する．以降，この転送における転送元を親プロセス，転送先を子プロセスと呼ぶ．RDMA モデルでは，親プロセスもしくは子プロセスのどちらか一方だけが通信命令を発行するので，もう一方のプロセスの状態を確認するために，親子間で同期のための通信が別途必要となる場合が多い．例えば put 命令でデータ転送を行う場合，仮に子プロセスの受信領域が計算等の他の処理で使用されている間にデータが書き換えられると，データの整合性が取れなくなる．これを避けるために，親プロセスは子プロセスの受信準備完了を待ってデータを書き込む必要がある．一方 get 命令では，親プロセスの送信領域にあるデータが正しくない場合，その後の処理に影響をきたす恐れがある．そのため，子プロセスは親プロセスの送信準備完了を待ってデータを読み込む必要がある．

また，RDMA モデルでは，通信命令発行側のプロセスはデータ転送の完了を検知出来るものの，相手側のプロセスはデータ転送の完了を検知する事が出来ず，いつ次の処理に進んで良いかが分からない．そのため，通信命令発行側のプロセスは，データ転送の完了を相手プロセスに通知する必要がある．

このように，RDMA モデルでは親子プロセス間で状態確認ための同期が必要で，その方法として以下の二つが考えられる．

一つ目は，バリア同期を用いる方法である．これは，ソースコード中のある個所に全プロセスが到達するまでプログラムの進行を停止するものである．しかし，この方法では到達確認に全プロセスが通信を行うため，繰り返しデータ転送を行う場合に無視できないオーバーヘッドが生じる．

二点目は，別途 put 命令などを用いて相手プロセスに特定の値を書き込み，データ転送の開始や完了を通知する方法である．put 型の場合，図 3 に示す通り，まず子プロセスから親プロセスに対し受信準備完了を通知するための値を書き込む．親プロセスは子プロセスから書き込まれた値から受信準備の完了を確認し，データ転送を行う．その後，親プロセスから子プロセスに対し転送完了を通知するための値を書き込む．get 型の場合はこの逆で，図 4 に示す通り，まず親プロセスから子プロセスに対し送信準備完了を通知するための値を書き込む．子プロセスは親プロセスから書き込まれた値から送信準備の完了を確認し，データ転送を行う．その後，子プロセスから親プロセスに対し転送完了を通知するための値を書き込む．

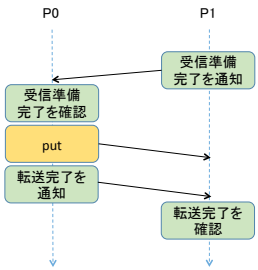

図 3 put 型での同期手順

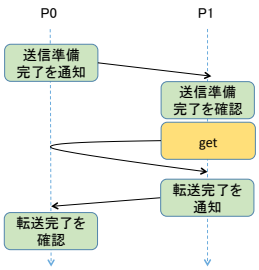

図 4 get 型での同期手順

4.4 パイプライン転送

親プロセスから子プロセス，さらにその子プロセスへとデータを伝搬する場合，送るべきデータを任意のサイズのセグメントに分割し，各セグメントをパイプライン転送する事で，リンクを有効に使う事が出来る．例えば，プロセス 0 からプロセス 1 にデータを送り，さらにそのデータをプロセス 2 へ送る場合，セグメントに分割しない場合は，プロセス 0 からプロセス 1 へのデータ転送が全て完了するまで，プロセス 1 からプロセス 2 へのデータ転送は開始出来ない．しかし，データをいくつかのセグメントに分割して送る事で，プロセス 1 に到着したセグメントを順次プロセス 2 へ転送していく事が可能となり，複数の通信を並行して進行させる事が出来る．

このパイプライン転送を RDMA モデルで実装する場合，各セグメントの転送毎に，前節で述べた同期処理が発生する．そのため，パイプライン転送によるデータ転送の短縮効果と，同期処理のためのオーバーヘッドの影響の見積もりに基づいて，パイプライン転送の可否，および最適なセグメントのサイズを決定する必要がある．

4.5 非ブロッキング通信の推進手段

MPI の永続型集団通信では，非ブロッキング集団通信を行うために，集団通信を開始する Start と，集団通信の完了を待つ Wait を用意している．非ブロッキング通信とは，データ転送を行う際に，転送完了を待たずに他の処理に移る事で，通信とその通信の結果に依存しない計算のオーバーラップを可能とする通信方法である．非ブロッキング集団通信の実装には以下の四つの方法が考えられる．

一つ目は，Start で無条件に発行可能な通信命令を発行し，Wait で残りの通信処理を全て行う方法である．この方法は，親子プロセス間で依存関係のあるアルゴリズムの場合，同期が必要なため，プロセス数が多い場合はほとんどの通信処理を Wait で行う事になるため，オーバーラップの効果は薄いと考えられる．

二つ目は，通信処理を進行させるためのプログレス関数を使用する方法である．アプリケーション中で Start により非ブロッキング集団通信を開始後，適切な頻度でプログレス関数を呼び出す事で，通信と計算をオーバーラップさせる事が出来る．ただし，プログレス関数の呼び出しはユーザーの責任で行わなければならず，また，プロセス数やメッセージサイズ，アルゴリズムの違いを考慮して適切な呼び出し頻度を調べる必要があり，ユーザーの負担が大きくなる．

三つめは，通信処理進行用のスレッドをライブラリ中で生成し，そのスレッドに通信処理の進行を担当させる方法である．この方法では，進行の処理をユーザーが明示する必要はなく，オーバーラップの効果も高い．しかし，プロセス内のスレッド数が一つ増える事になるため，既にコア数分の計算スレッドを立てている場合に，性能低下の恐れがある．

四つ目は，通信管理を NIC にオフロードする方法である．この方法では，通信命令列を NIC へ渡し，通信の管理を NIC に任せる事で，通信を行っている間も CPU は計算に専念する事が出来る．しかし，オフロードが可能なインターコネクトや，オフロード可能な通信命令は限られている．

4.6 通信バッファ

RDMA モデルによる永続型集団通信では，通信準備関数で指示された送受信領域間でのデータの直接転送と，明示的に確保した集団通信専用のバッファを介したデータの転送が考えられる．

通信バッファを介した転送では，通信準備関数で必要なバッファ領域の確保と登録，登録先のメモリアドレス情報の通知を行う．子プロセスの受信バッファがある限り，親プロセスはいつでもデータを転送出来るため，子プロセスから親プロセスに対する受信準備完了通知のための同期が必要ない．しかし，送受信領域と通信バッファ間でメモリコピーが必要なため，メモリコピーのためのオーバーヘッドと同期の削減による性能向上のトレードオフとなる．一般に，メッセージサイズが大きくなるにつれてメモリコピーの処理時間も増大するため，主に小メッセージサイズでの高速化が期待できる．

5. 永続型ブロードキャスト通信の性能評価

5.1 永続型ブロードキャスト通信の実装

今回，永続型ブロードキャスト通信インタフェースを例として，実装を行った．ブロードキャスト通信は，あるプロセスが持つデータ構造を，グループ内の他の全プロセスに送る集団通信である．インタフェースとして現在 MPI Forum で議論されている案では、永続型集団通信の Init 関数において、送受信データを、開始アドレスとデータ型およびデータ要素数で指定する。プログラマは、整数や実数等の基本的なデータ型を組み合わせた複合的なデータ型や、離れたメモリ領域にまたがって配置された離散的なデータ型等を、この送受信データの型として指定することができる。一方、現在の我々の実装では、まだ様々なデータ型の処理を実現しておらず、送受信を連続領域に限定しているため、送受信データは開始アドレスとデータのバイト数で指定する。

実装アルゴリズムとしては，4.1 節で紹介した二項木を採用した．二項木アルゴリズムでは，最初のステップで木の根に当たるプロセスは子プロセスに対しデータを転送する．以降は，既に親プロセスからデータを受け取ったプロセスが子プロセスにデータを転送していく事で，全プロセスへデータを伝搬する．今回は内部で用いる通信命令の選択肢として put 型と get 型の二つを試し，性能を比較する．

put 型と get 型ともに，親子間の同期には，値として整数値のカウンタを用意する．カウンタの値は，通信開始関数の呼び出し毎に増加する．put 型の実装について，2 プロセスの場合を例に説明する．まず子プロセスは受信準備が完了すると自らのカウンタの値を親プロセスに対し put 命令により転送する．親プロセスはカウンタの値が変化するまで待ち，値の変化から受信準備の完了を確認する．その後，子プロセスに対しデータを転送する．転送完了後，転送完了を通知するために自らのカウンタをの値を子プロセスに対し転送する．子プロセスはカウンタの値が変化するまで待ち，値の変化から通信の完了を確認する．get 型についても同様に，put 命令を用いたカウンタによる同期，

表 1　実験環境

System	FUJITSU PRIMERGY CX400
Interconnect	Infiniband
Nodes	1476
OS	Red Hat Linux Enterprise
Library	ACP Library

get 命令によるデータ転送を行う．

5.2　実験環境

実験環境を表 1 に示す．インターコネクトは，InfiniBand FDR であり，通信ライブラリとしては，ACP を利用した．

5.3　性能モデル

メッセージサイズやプロセス数に応じて，選択する通信命令による通信性能の優劣が変化すると考えられるため，適切な通信命令を選択するために，put 型と get 型それぞれの実装における性能モデルを構築する．

今回構築したモデルでは，木の根にあたるルートプロセスの通信開始から通信完了までの時間をブロードキャスト通信の所要時間として見積もる．今回の実装において，一回の同期処理に要する時間を t_{sync}，メッセージサイズ M における put 命令と get 命令によるデータ転送に要する時間をそれぞれ $t_{put,M}$ と $t_{get,M}$，とそれぞれ表す．なお，同期処理に要する時間とは，二つのプロセス間で，一方のプロセスが put 命令により整数値を送り，もう一方のプロセスが送られてきた整数値を確認するまでの時間である．

put 型では，通信開始関数が呼ばれると，ルート以外の全プロセスは親プロセスに対し受信準備完了通知のための同期を行う．ルートプロセスは同期の完了を t_{sync} だけ待つ．その後，子プロセスに対してデータを書き込むための put 命令を発行し，$t_{put,M}$ だけ待つ．データ転送完了後，子プロセスに対し転送完了を通知するための同期を行い，t_{sync} だけ待つ．これらの操作をステップ数分行う必要がある．一方，今回使用しているインターコネクトは双方向通信に対応しているため，ルートプロセスに対するデータ転送と，ルートプロセスからのデータ転送は同時に進行可能である．そのため，2 回目以降の子プロセスからの受信準備完了通知のための同期に要する時間は無視できる．以上をふまえ作成した put 型の性能モデルを (1) に示す．

$$T_{put,M} = log_2 P \times t_{put,M} + (log_2 P + 1) t_{sync} \quad (1)$$

一方，get 型は，通信開始関数が呼ばれると，ルートプロセスは全ての子プロセスに対し，送信準備完了を通知するための同期を行う．この際，全ての同期処理のための put 命令は連続して発行するため，t_{sync} は 1 回分として考える事が出来る．ルートプロセスからの送信準備完了の通知

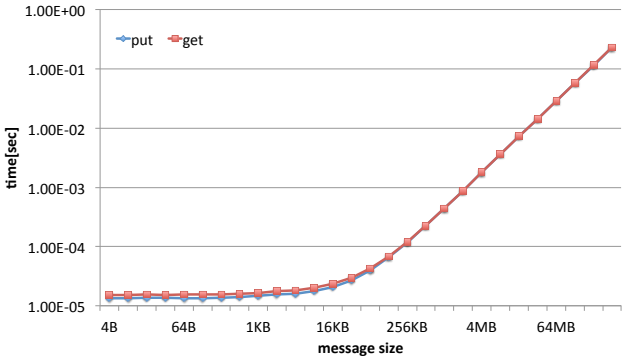

図 5　4 プロセスでの put 型と get 型の実行時間

を確認した子プロセスは，ルートプロセスからデータを読み込むための get 命令を発行し，転送完了後ルートプロセスに対し転送完了通知のための put 命令を発行する．ルートプロセスから子プロセスに対するデータ転送と，データ転送完了通知のための同期は同時進行可能である．以上をふまえ作成した get 型の性能モデルを (2) に示す．

$$T_{get,M} = log_2 P \times t_{get,M} + 2 t_{sync} \quad (2)$$

なお，このモデルに用いているで t_{sync} は，2 プロセスにおける 4 バイトでの put 型の実測結果を 3 で割った値として考える．また，$t_{put,M}$ と $t_{get,M}$ はそれぞれ，今回実装した put 型と get 型におけるブロードキャスト通信の 2 プロセスでの実測結果から，二回分の t_{sync} を引いた値とする．

5.4　実験結果

実験では，put 型と get 型の二つの実装について，複数のメッセージサイズとプロセス数で実行時間を計測した．実験結果の中から，特に 4 プロセス，16 プロセス，64 プロセスでの結果を図 5 から図 7 に示す．各グラフの横軸はメッセージサイズ，縦軸は時間を表している．図 5 より，4 プロセスでは 4KB から 16KB 程度のメッセージサイズにおいて put 型が get 型の性能を上回っている事が分かる．一方で，図 6 と図 7 より，16 プロセスと 64 プロセスでは 4KB から 16KB 程度のメッセージサイズにおいて put 型が get 型に比べ性能が劣っている事が分かる．また，4 プロセス，16 プロセス，64 プロセスともに中から大メッセージサイズ域では put 型と get 型で同程度の性能となっている．

5.5　考察

式 1，式 2 を用いて put 型と get 型それぞれの実行時間の見積もりを行い，実測値との比較を行った．その中から，4 プロセス，16 プロセスと 64 プロセスにおける結果をそれぞれ図 8，図 9，図 10 に示す．各グラフの横軸はメッセージサイズ，縦軸は時間であり，破線がモデルによる見積もり，実線は実機による実測値を示している．図 8，図 9 から，4 プロセスおよび 16 プロセスにおいて，実測では

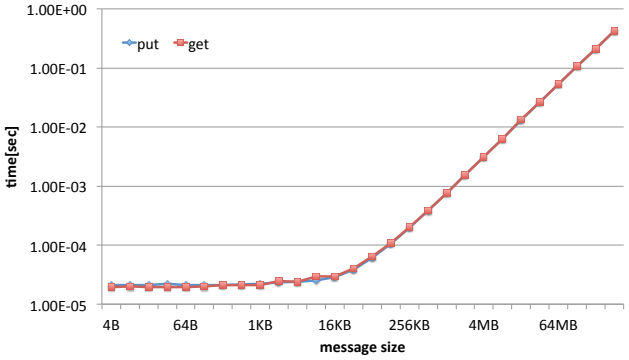

図 6　16 プロセスにおける put 型と get 型の実行時間

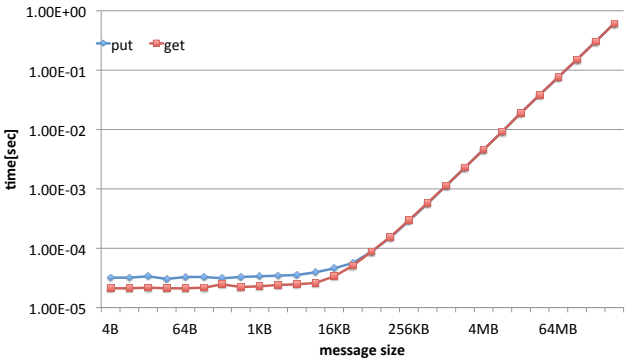

図 7　64 プロセスにおける put 型と get 型の実行時間

64KB まではわずかに put 型の性能が優れている事が分かる．そのため，このメッセージサイズの範囲では通信命令の選択肢として put 型を選ぶのが適切と考えられる．一方，見積もりでは性能の優劣が実測と逆になっており，見積もり結果からは適切な通信命令を選択する事が出来ない．

一方，64 プロセスでは図 10 に示す通り，実測においてメッセージサイズが 64KB までは get 型の性能が優れており，それ以降はほぼ同適度の性能となっている．見積もりでも同様の結果になっている事を確認できる．この事から，64 プロセスにおいては，メッセージサイズに応じた通信命令の優劣をモデルにより見積もれている事が分かった．

4 プロセスおよび 16 プロセスにおいて見積もりによる通信命令の優劣が実測と異なっている事は，性能モデルの見積もり精度を向上させる事で解決可能であると考えられる．見積もり精度向上のためには，まず性能モデルの作り方を変える事が挙げられる．既に LogP モデル [5] や PLogP モデル [7]，LogGP モデル [6] 等が提案されており，これらを用いる事により精度の向上が見込まれる．また，見積もりに必要な基礎データのより正確な測定方法の検討も必要である．

6. まとめ

本稿では，RDMA モデル上での永続型集団通信インタフェースにおける実装上の課題について考察し，永続型ブロードキャスト通信を例として，特に put 型と get 型の二

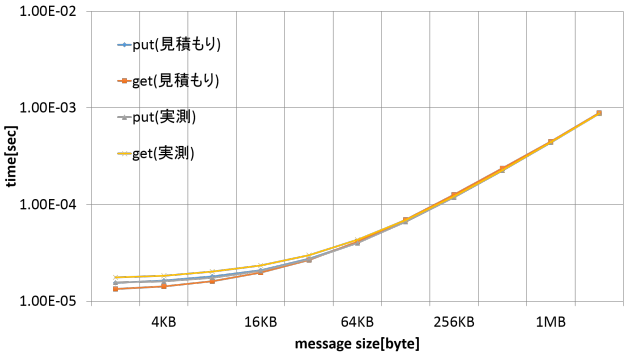

図 8　4 プロセスにおける put 型と get 型の実測値と見積もり

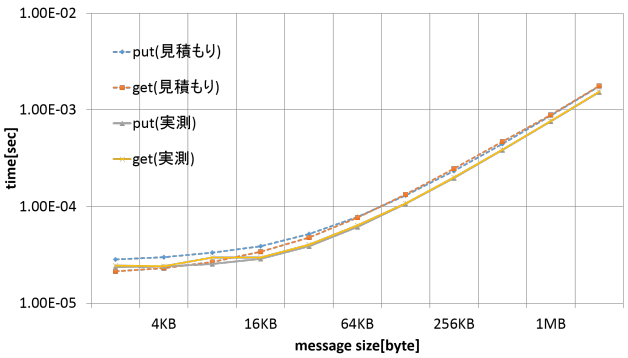

図 9　16 プロセスにおける put 型と get 型の実測値と見積もり

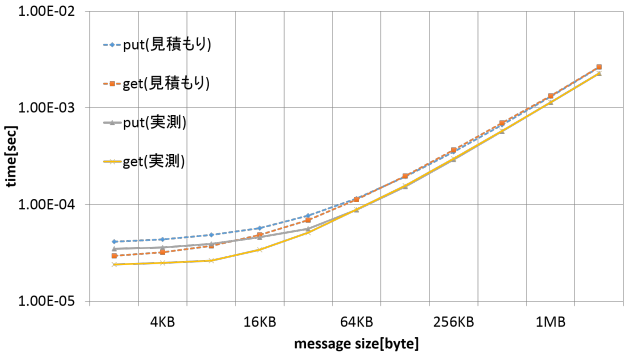

図 10　64 プロセスにおける put 型と get 型の実測値と見積もり

つの実装について性能モデルの作成と性能評価を行った．実験の結果，プロセス数が 64 では，性能モデルによりメッセージサイズ毎の通信命令の優劣について正しく判断出来ている事が分かった．一方，4 プロセスおよび 16 プロセスの場合は，性能モデルにより適切な通信命令を判断する事は出来なかった．

今後の課題としては，性能モデルの作り方，見積もりに使用する基礎データの計測方法の見直しによる見積もり精度の向上が挙げられる．

参考文献

[1] MPI Forum, http://www.mpi-forum.org/
[2] ACE Project, http://ace-project.kyushu-u.ac.jp/main/jp/index.html

[3] A. Faraj, X. Yuan, and D. Lowenthal, “STAR-MPI: self tuned adaptive routines for MPI collective operations.” Proceedings of the 20th annual international conference on Supercomputing. ACM, 2006.
[4] T. Nanri, and M. Kurokawa, “Efficient Runtime Algorithm Selection of Collective Communication with Topology-Based Performance Models.”, Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications, 2012.
[5] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian and T. von Eicken, “LogP: Towards a Realistic Model of Parallel Computation.”, Proc. Symposium on Principles and Practice of Parallel Programming (PPoPP), pp. 1-12, 1993.
[6] A. Alexandrov, M. F. Ionescu, K. E. Schauser and C. Scheiman, “LogGP: Incorporating Long Messages into the LogP Model - One Step Closer Towards a Realistic Model for Parallel Computation.”, Proc. Symposium on Parallel Algorithms and Architectures (SPAA), pp.95-105, 1995.
[7] T. Kielmann, E. B. Henri , and G. Sergei, “Bandwidth-efficient collective communication for clustered wide area systems.” Parallel and Distributed Processing Symposium, IPDPS 2000. Proceedings. 14th International. IEEE, 2000.