情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 2　126–137 (June 2010)

クラウド環境における大規模データブロードキャストの動的最適化

千葉立寛[†1,†2]　ティロ キールマン[†3]
マタイス デン バーガー[†3]　松岡聡[†1,†4]

クラウド環境において大規模データセットを用いた並列アプリケーションを実行する場合，巨大なデータを各ノードに効率良く配布する必要がある．従来のクラスタ・マルチクラスタ型並列計算実行環境におけるブロードキャスト通信の最適化手法では，ネットワークトポロジやバンド幅などのモニタリング情報をもとに効率的に転送可能なスパニングツリーを構築してブロードキャストを実現する手法など様々な手法が研究されているが，クラウドのような動的にネットワーク性能が変化するような環境においては，ボトルネックリンクの発生やトポロジが変化した場合，ツリーの再構築を行わなければならず，つねに最適な性能を維持し続けることは難しい．本研究では，クラウド環境として Amazon EC2/S3 を対象とし，S3 に保存された大規模なデータセットを EC2 ノードに高速にブロードキャストする手法を提案する．提案するアルゴリズムは，トポロジ情報を必要とせず，また，各ノードがダウンロードボトルネックリンクに対するロードバランスを行うことで，動的に各ノードのスループットを最大化することを可能とする．本稿では，まず Amazon EC2/S3 の性能を測定して問題点を明らかにし，提案アルゴリズムの実装を行い，EC2/S3 での性能評価を行った．その結果，単純な手法に比べてスケーラブルでかつ高い性能が得られることを確認した．

Dynamic Optimization for Large Data Broadcast on Clouds

Tatsuhiro Chiba,[†1,†2] Thilo Kielmann,[†3]
Mathijs den Burger[†3] and Satoshi Matsuoka[†1,†4]

Data-intensive parallel applications on clouds need to deploy large data sets from the cloud's storage facility to all compute nodes as fast as possible. Many optimal broadcast algorithms have been proposed for clusters and grid environments. The most common approach is, for example, to construct one or more optimal spanning trees, which can maximize available bandwith or avoid bottleneck links based on network topology and network monitoring data. Once available bandwidth changes dynamically, however, it is difficult to keep optimal performance. In this paper we focus on Amazon EC2/S3, which is most commonly used clouds, and we propose high performance broadcast algorithms; these algorithms make it possible to broadcast large data from Amazon S3 to multiple Amazon EC2 nodes. The salient features of our algorithms are to construct an overlay network on clouds without network topology information, to optimize node available throughput dynamically, and to increase the download throughput by letting nodes cooperate with each other. As a result, all nodes can download files from S3 quickly, even when the network performance changes while the algorithm is running. We evaluate our algorithms on EC2/S3, and show that they are scalable and consistently achieve high throughput. Both algorithms perform much better than each node downloads all data directly from S3.

1. はじめに

近年，Amazon Web Services（AWS）[1] をはじめ，Windows Azure [2]，Google App Engine などのクラウド環境の整備により，クラウドの利用が様々な場面で進められており，従来スーパコンピュータや大規模クラスタで行われてきた HPC アプリケーションもクラウドで実行することへのニーズが高まってきている．HPC アプリケーションの中でも ATLAS [3] や BLAST [4] のような大規模なデータを用いて処理を行うアプリケーションの実行を考えた場合，実験データをデータセンタやクラウド上のストレージから計算ノードへできるだけ高速かつ効率的に転送するためのマルチキャスト手法を開発することは，データサイズが増えるに従ってますます重要となる．

従来，クラスタなどの並列分散システム上での高速に通信可能な集団通信アルゴリズムや，P2P ネットワークでの耐故障性やスケーラビリティを重視するファイル共有・ビデオストリーミングにおいて，様々なマルチキャスト最適化手法が提案されており，それらの手法は，実行する環境の前提とアプリケーションの要求を満たすように設計されている．クラ

†1 東京工業大学
Tokyo Institute of Technology
†2 日本学術振興会特別研究員 DC
Research Fellow of the Japan Society for the Promotion of Science
†3 アムステルダム自由大学
Vrije Universiteit Amsterdam
†4 国立情報学研究所
National Institute of Informatics

ウド環境は，従来の並列分散システムと共通点の多い環境ではあるものの，多数のユーザが仮想マシン環境を通してクラウドを共有・利用することで，動的にトポロジやバンド幅が変化することが考えられるため，従来の前提に基づいたマルチキャスト手法では，つねに最適な通信ができるとは限らない．

我々は，クラウド環境上でも効率的にマルチキャストを行う最適化手法を提案する．分散並列システムと P2P システムのアイディアを取り入れており，高スループットでかつノード数・データサイズに対してもスケーラブルにブロードキャストを実行可能となるように設計した．提案する最適化手法を実装し，Amazon EC2/S3 上での性能比較を行い，非常に高い性能であることを確認した．

2. 関 連 研 究

本章では，様々な立場から研究が行われている大規模データマルチキャスト手法についての特徴をまとめ，目指す方向性について言及する．

2.1 並列分散システムでのマルチキャスト

クラスタに代表される並列分散システムでのマルチキャストは，MPI をはじめとするメッセージパッシングシステムでの集団通信アルゴリズムにおいて，静的または動的な最適化手法を用いた様々なアルゴリズムが提案されてきた．HPC アプリケーションの実行をターゲットとしている並列分散システムでは，全ノードに対していかに高速にデータをマルチキャストするかに焦点を当てた最適化がなされている．システムの前提として，**(1)** ネットワークが安定して高性能かつ Fat Tree などのネットワークバンド幅を最大限活用できる構成であり，**(2)** トポロジやルーティングは変化せず，**(3)** ノードが利用できる送受信のバンド幅は対称性があることを仮定している．

これらのアルゴリズムの基本的な方針としては，ネットワークの物理的なトポロジや利用可能なバンド幅，ノード数などの情報をもとに最適なスパニングツリーを構築する手法が一般的である[5]．データはルートノードを基点とするツリーに沿って転送されていくため，並列分散システムでのマルチキャスト手法は，sender が主体の push 型通信であるといえる．

さらに，大規模なメッセージをマルチキャストする場合，スパニングツリーを複数構築し利用可能なバンド幅の中でスループットを最大化し，データを複数のチャンクに分割してパイプライン転送を行うことでより高速にマルチキャストを実現する手法も提案されている[6)–9)]．たとえば，Stable Broadcast[8] では，ネットワークトポロジがツリーであるとして，バンド幅情報が付加されたトポロジの情報を用いて，参加ノード間のルーティングを深さ優先でたどってパイプラインを構築し，ノードが得られるバンド幅を最大化することで高速化している．

2.2 P2P でのオーバレイマルチキャスト

P2P システムにおけるマルチキャスト，BitTorrent[10] に代表されるファイル共有や SplitStream[11] などにおいてもデータストリーミングを行うアプリケーションの核として様々な最適化手法が研究されている．P2P ネットワークにおけるこれらのアプリケーションでは，まず，ロバスト性に焦点を当てており，**(1)** 十分なネットワーク構成やトポロジ情報がない状態からでも自律的にオーバレイを構築することや，**(2)** 動的なノードの参加・脱退が起こってもシステムが稼働し続ける耐故障性，**(3)** 動的に変化するネットワーク性能への対応が必要とされている．データを拡散させることで P2P システムの安定性を高め，動的なネットワーク性能の変化に対応するため，ブロードキャストデータは小さなチャンクに分割されて多数のノードに分散されるが，一般的には，どのチャンクをどのノードが持っているかをシステム全体として知ることは難しく，チャンクの送信側から転送経路を決定することができないため，受信側から送信側への不足チャンクのリクエストを行い，その情報をもとに送信側がチャンクを送信する．したがって，P2P でのマルチキャスト手法は，receiver が主体の pull 型通信であるといえる．

ノード間の近接性を考慮したクラスタリングを行い，WAN のような RTT が大きいネットワークを経由したノード間でのリクエスト回数を減らしたり，同一チャンクが何度も WAN を転送されないように工夫をしたりすることで，スループットの最適化が行われ，pull 型通信でも高い性能を得ることが可能となる．このような手法は，MOB[12] でも用いられている．本提案アルゴリズムでもノード内でのデータ転送では pull 型通信を用いており，同様の最適化手法を行っている．

2.3 広域環境におけるデータ転送

地理的に分散した複数の計算資源を利用するグリッド環境においても，巨大なデータをグリッドを構成するノード間・ストレージ間で効率良く転送するための様々な手法が研究されているが，高性能かつ高信頼な転送を行うことが可能な GridFTP[13] は，デファクトスタンダードなデータ転送プロトコルとしてグリッド環境でよく用いられており，複数のデータソース，または，同一のデータソースからの複数の TCP コネクションを束ね，さらに，個々の TCP コネクションでの最適なウィンドウサイズを調整することで高スループットな転送を実現している．

文献 14) では，送受信ノード間に直接 TCP コネクションを張るのではなく，複数の中間

ノードを経由してパイプラインで転送を行う手法（multi-hop path splitting）と，ネットワークバンド幅を考慮して複数の通信パスを用いて転送を行う手法（multi pathing）に対して，ヒューリスティックを用いて送受信ノード間での通信スループットを最大化するアルゴリズムを提案し，従来の GridFTP よりも高い性能を達成することを可能としている．

複数の通信パスを用いて独立した複数のコネクションをアグリゲートして合計の通信スループットを稼ぐアイデアは，クラウドストレージからノードへのデータ転送部分において本稿で提案するアルゴリズムでも用いられている．しかしながら，上記のアルゴリズムでは，帯域確保の保証ができた各ノード間でのバンド幅マップを用いて最適化した通信スケジューリングを行っており，個々のノードのスループットが通信のたびに動的に変動することを想定したうえで通信スループットの最大化を行う本提案アルゴリズムとは異なる．

2.4　クラウドでのマルチキャスト

Amazon EC2/S3 に代表されるアプリケーション実行基盤をユーザに提供するクラウドサービスの登場により，HPC アプリケーションもクラウド上で実行する要求が高まっている．文献 15)，16) では，EC2/S3 の性能面・コスト面から評価を行い，データインテンシブアプリケーションの実行に適しているかどうかを探っている．また文献 17) では，NPB を用いて EC2 の性能を評価している．これらの先行研究では，EC2 内の通信性能が低いため，計算インテンシブアプリケーションでの性能は既存のクラスタ環境と比べて現状では期待できないと述べている．データインテンシブアプリケーションについては，ストレージの利用コストやスケーラビリティ，可用性の観点からクラウドでの実行が期待できるが，データアクセスでのパフォーマンスチューニングを行う必要性があることについて言及しており，効率的なマルチキャスト手法を開発することは，より重要な技術の 1 つである．

クラウドで提供される計算資源が動作する物理的な環境は，P2P のように広域に配置された疎結合なノード群ではなく，物理ノード群が密に配置された巨大なクラスタで構成されるデータセンタである．しかしながら，従来からあるクラスタ環境とは対照的に，これらの計算資源は完全に，または一部が仮想化されてユーザに提供され，ユーザ透過に見えない部分も多いため，クラウドでのマルチキャストでは，それらを十分に考慮した最適化を行う必要がある．クラウドストレージにあるデータを各計算ノードに配置するためのマルチキャスト実行を考えているため，以下では，(1) クラウドストレージからクラウド計算ノードへの転送と，(2) クラウドの計算ノード間での転送，という 2 つのパターンで考慮すべき点をあげる．

2.4.1　クラウドストレージからの転送

クラウドストレージは，ユーザがファイルシステムとして利用することを想定したものではなく，REST や SOAP の API を通してデータにアクセスすることを想定して設計されているが，大量のデータを保存し，多数のユーザからのアクセスへの対応や信頼性を確保するためには，Gfarm v2[18] や HDFS[19]，GFS[20] に代表される並列分散ファイルシステムでクラウドストレージを構成し，スケーラビリティとデータ永続の信頼性を高めていると推測することができる．そのため，並列分散ファイルシステムで起こりうる問題が，潜在的にクラウドストレージにも存在すると考えられる．これらの並列分散ファイルシステムでは，データを複数に分割し，複数の I/O ノードに分散させ，また，レプリカを作ることでファイルの冗長性を高め，ノード数の増加に対しても高いスケーラビリティで I/O スループットを向上させることが可能だが，I/O ノードでのコンテンションや，レプリカの位置によって，個々のリンクごとのパフォーマンスは低下する．たとえば，GFS[20] などでも，同一のチャンクサーバにアクセスが集中した場合，個々のリンクのパフォーマンスが低下する現象が確認されている．

2.4.2　計算ノード間での転送

クラウド内のネットワーク性能は P2P 計算環境と同様に動的に変化することが推測され，様々なレイヤでボトルネックリンクが発生する可能性がある．たとえば，同一の物理マシンを共有する他のユーザの VM の影響によって自分が利用しているノードのアップリンクとダウンリンクの性能に影響を及ぼす場合や，クラウドシステム側での動的なロードバランスによりルーティングや通信帯域が変更される場合である．2.1 節で述べた既存の分散システムに適応したパイプラインツリーを用いたマルチキャスト手法や，2.3 節で述べた広域環境でのデータ転送手法は，モニタリングデータやトポロジ情報，利用可能なバンド幅マップを用いて転送のルーティングツリーを決定することを前提としている．しかしながら，クラウド上でこのような情報をつねに最新の状態として知り続けることは難しく，マルチキャストごとにネットワーク性能が変化しうるクラウドでは，最適なパイプラインツリーを 1 度構築してもつねに最適な発揮することは難しい．

我々が 4 章で提案するアルゴリズムは，動的なネットワーク性能の変化にも対応するため，主に P2P で用いられる pull 型で転送を行い，かつ，クラウドストレージからのデータ取得を複数のコネクションを集約して，また，個々のリンクのパフォーマンスを動的に最適化することで，マルチキャスト実行の全体を高速化を可能とする．

3. Amazon EC2/S3

本研究では，現実のクラウド環境として Amazon EC2/S3 を対象としている．本章では，EC2/S3 の性能と特徴を列挙し，大規模データを扱う HPC アプリケーション実行における EC2/S3 利用モデルを定義し，注目する課題点について述べる．

3.1 特　　徴

Amazon Elastic Compute Cloud（EC2），Simple Storage Serivice（S3）は，Amazon Web Services（AWS）[1] によって提供されるクラウドサービスである．EC2 はユーザに対して仮想化された計算資源を提供するサービスで，必要に応じて 1 ノードから数千ノードを起動してアプリケーションの実行を可能にする．提供される計算資源は，Xen を用いた仮想マシンであり，EC2 ではこれをインスタンスと呼ぶ．ユーザは，S3 に保存されている Amazon Machine Image（AMI）と呼ばれる OS イメージと，CPU やメモリサイズなど決定するインスタンスタイプを選択して仮想マシンを起動する．

S3 は，高い信頼性や可用性，データの永続性などの特徴を持ったクラウドストレージサービスである．ユーザは *bucket* と呼ばれるフォルダに似た一意の *name space* を S3 上に作成し，その中に 1 Byte〜5 GBytes までの *object* を保存する．*bucket name* と *object name* を *key* として，ユーザは REST や SOAP API を用いることでデータの追加・取得・削除が可能である．

3.2 クラウド利用モデル

本稿で扱うデータインテンシブなアプリケーション実行環境としてのクラウド利用モデルを以下で定義する．想定する大規模データを扱う HPC アプリケーションは，クラウド上のデータストレージに保存された大きなデータセットを利用する．計算を実行する各ノードは，初期状態ではこのデータセットを持っておらず，個々のローカルスペースにクラウドストレージからデータをダウンロードした後，計算を実行すると仮定する．

上記のようなクラウド利用モデルを実現する実行環境として，本稿では，Amazon EC2/S3 を用いる．また，計算の実行前に S3 上の大規模データを EC2 の各ノードにできるだけ高速に配置することを目的とする．現在 EC2/S3 は EU サイト，US サイトの 2 つのリージョンが利用できるが，本稿では利用するリージョンは 1 つとし，リージョン間をまたぐ EC2/S3 の組合せは利用しない．

3.3 予 備 評 価

S3 にあるファイルを EC2 ノードへブロードキャストする最も単純な手法は，個々のノードがそれぞれ直接 S3 からファイルをダウンロードすることである．本稿では，以下このアルゴリズムを flat tree と呼ぶ．S3 からのダウンロードスループットが十分に得られるならば，EC2 ノード間でのデータ転送を含まず，同一のリージョン内の EC2 と S3 間のデータ転送には料金がかからないため，S3 をルートとする flat tree もブロードキャストを実現する手法として妥当な選択肢の 1 つである．本節では，予備評価として S3 から EC2 ノードへのダウンロード性能を測定し，flat tree アルゴリズムが適しているかどうかを探る．

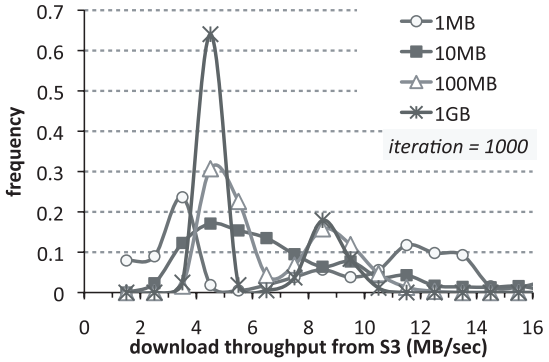

図 1　1 ノードアクセス時のダウンロードスループットの相対頻度
Fig. 1　Download throughput frequency from S3 to one EC2 node.

3.3.1 単一ノードからのダウンロード性能

まずはじめに，1 ノードから S3 上のファイルをダウンロードする性能を測定した．**図 1** は同じファイルに対して同じノードから連続して 1,000 回のダウンロード試行を行ったときのファイルサイズごとに得られるダウンロードスループットの相対頻度を表している．たとえば 100 MB のときの相対頻度を見ると，試行全体の 60%程度は 6 MB/sec 以下，8 MB/sec 以上でダウンロードすることができたのは 30%程度という結果となっている．また得られるスループットは，ファイルサイズが大きくなるにつれて，速い場合と遅い場合の 2 極化していくことが確認される．

3.3.2 複数ノードからのダウンロード性能

次に，同じファイルに対して複数ノードから同時にダウンロードしたときの flat tree アルゴリズムの性能を測定した．**図 2** は EC2 上の 8 ノードが同じ *bucket* にある 1 GB のファイルを 10 回ダウンロードしたときの，試行ごとの通信完了時間の最大値，平均値，最小値を示したものである．3.3.1 項と同様に各ノードが得られる S3 からのダウンロードスルー

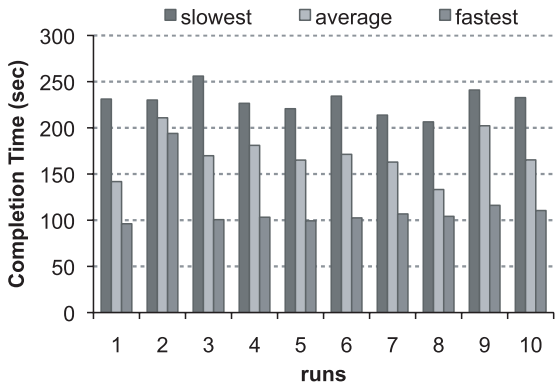

図 2　flat tree アルゴリズムでの通信完了時間（8 nodes，1 GB）
Fig. 2　Download completion time with flat tree algorithm for each experiment (8 nodes, 1 GB).

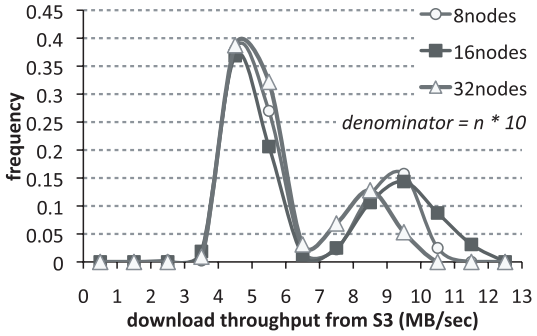

図 3　複数ノードアクセス時のダウンロードスループットの相対頻度（flat tree，1 GB）
Fig. 3　Download throughput frequency from S3 to 8, 16, 32 EC2 nodes (flat tree, 1 GB).

プットはばらつくため，ダウンロード完了時間はノードによって大きなずれが生じることが分かる．最も早く通信を完了するノードは 10 MB/sec 以上でダウンロードしているが，最も遅いノードの場合，4 MB/sec 程度のスループットしか得ることができていない．また，**図 3** はこのときの各ノードで得られた平均ダウンロードスループットの頻度を表している．この図から，ノード数が増加しても 1 ノードのときのダウンロードスループットの頻度と同様の傾向が得られることが確認できる．

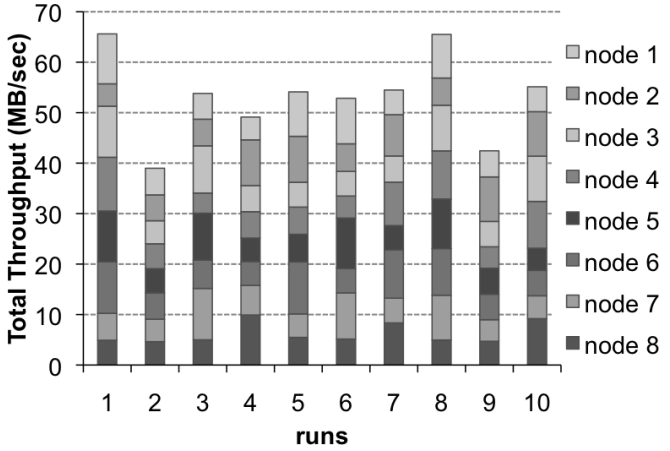

図 4　flat tree での各ノードのダウンロードスループットの総和の推移（1 GB）
Fig. 4　Sum of each node throughput with flat tree (1 GB).

図 4 は，試行ごとの各ノードのスループットの総和を示している．この図から，ノードが得るスループットは前回にダウンロードしたときのスループットに関係なく，そのときどきによって得られるスループットが変動することが確認できる．たとえば，node 5 は 3 回目の試行時に 10 MB/sec であるが，4，5 回目は 5 MB/sec 程度となり，6 回目の試行で再び 10 MB/sec のスループットを得ている．試行ごとの全ノードのスループットの総和は，8 ノード時でたかだか 50 MB/sec 程度であることも確認できる．**図 5** では，ノード数を増やしたときの全ノードの S3 からのスループットの総和の平均を表しているが，少なくとも 32 ノードまでは得られるスループットの総和は，ノード数に比例しており，S3 から EC2 ノードへの利用可能なバンド幅を満たすことはなかった．

3.3.3　flat tree アルゴリズムの課題

ノードが S3 からファイルを直接ダウンロードするときに得られるスループットは，システムの状況により動的に変化し，また，同一ノードから同一ファイルに対するアクセスであっても，アクセスごとにスループットが変化して性能が安定しないことが確認された．1,000 回の試行により，だいたい 20〜30%の確率で比較的高いスループットを，60〜70%の確率で低いスループットしか得られないことも確認された．ノード数によらずに同程度の頻度で性能が律速されていることが図 3 の頻度分布から分かる．このため，flat tree を用いて

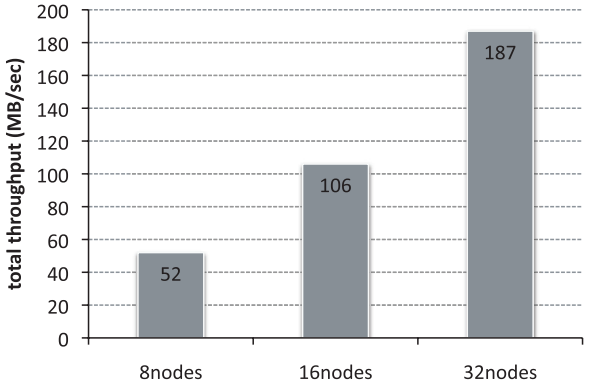

図 5　ノード数に対する平均ダウンロードスループットの総和の推移
Fig. 5　Average total throughput from S3 to EC2 node on 8 nodes, 16 nodes, 32 nodes.

ブロードキャストを行った場合，ノードごとに通信完了時間がばらつき，結果としてアプリケーション全体の性能に影響する．

その一方で，S3 から引き出すことができるスループットの総和は，少なくとも 32 ノードまではノード数に対して増加している．このため，現時点では S3 から EC2 ノードへのバンド幅限界を知ることはできなかった．各ノードはシングルコネクションで S3 からデータを取得しているが，マルチコネクションで S3 からデータをダウンロードしたときの各ノードが得られるトータルスループットの性能限界や，これを複数ノードから行った場合の S3 からのダウンロードバンド幅の限界を調べることは今後の課題の 1 つである．

また，このように各コネクションごとのスループットがばらつく原因としては，S3 を構成するストレージシステムによる動的な帯域制限やデータが保存されている I/O サーバでのコンテンションなどが考えられる．一般的な並列分散ファイルシステムや他のクラウドストレージにおいても，同様の理由によりスループットの変動が起こると予測されるため，このスループット変動を考慮して性能を向上・安定させる手法を探ることは，解決すべき問題の 1 つである．

4. 提 案 手 法

4.1 方　　針

flat tree アルゴリズムを用いてクラウドストレージからファイルを取得する際の問題点と，2.3 節で述べたクラウドで起こりうる一般的な問題点をふまえ，以下のような方針で提案アルゴリズムを設計する．

- **クラウドストレージからのダウンロードスループットの最大化**：できるだけ速く S3 から各ノードにデータを転送するため，ノードごとのダウンロード性能の差異を吸収するように，全ノードが協調して S3 からのスループット総和を最大化することを目指す．
- **全ノードの通信完了時間の総和を最小化**：ノード間でのダウンロード性能に差がでることで，計算開始が遅れるのを防ぐため，個々のノードの通信完了時間ではなく，全ノードトータルでの通信完了時間の最小化を目指す．
- **モニタリング情報・特定のツリーに依存しない通信**：利用可能なバンド幅の動的な変化に対応するため，計算ノード間の通信には，パイプラインを構築してノード間を転送する方式ではなく，各ノードが自律的に動作してスループットを最大化するような P2P 型の転送を用いる．また，利用するノードが数百，数千ノードにスケールした場合でも，高いスケーラビリティを保つことを目指す．

表 1 は，クラスタや P2P におけるマルチキャスト手法の特徴と，それらをふまえて我々が提案するクラウドにおけるマルチキャスト手法の特徴をまとめたものである．クラスタ環境においては，モニタリングデータを用いてそのネットワーク性能を活かすための高スループットなスパニングツリーを用いて高速なマルチキャストを実現するが，動的かつ不安定なネットワーク性能の変化に対応することはできない．また，P2P 環境においては，ネットワークの変化やスケーラビリティの高いアルゴリズムであるが，高い性能をつねに発揮することは難しい．我々の提案するアルゴリズムは，クラスタや P2P システムで用いられるアルゴリズムの両方の利点を取り入れ，高スループットとスケーラビリティの高いアルゴリズムである．

表 1　マルチキャスト実行環境におけるアルゴリズムの特徴
Table 1　Features of general and proposed multicast algorithms on each environment.

	cluster	P2P	clouds
multicast topology	spanning tree(s)	overlay	tree + overlay
communication type	push	pull	pull
network performance	high	low	middle
node proximity	dense	sparse	dense
cost of communication algorithm	high	low	low
node-to-node performance	homogeneous	heterogeneous	heterogeneous
underlying network topology	stable	unstable	(un)stable
correspond to dynamic change	bad	good	good

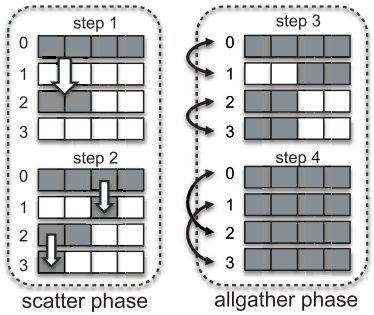

図 6　van de Geijn アルゴリズムによるブロードキャスト
Fig. 6　Broaccast with van de Geijn algorithm.

4.2　non-steal アルゴリズム

クラスタ・マルチクラスタ環境の MPI におけるブロードキャストアルゴリズムとして van de Geijn らによって提案された手法が高い性能を可能とするアルゴリズムの 1 つとして知られている．このアルゴリズムは，**(1)** root ノードがデータを等分割して，recursive harving でデータを転送し（Scatter），**(2)** 分割されたデータを各ノードが recursive doubling で交換し全データを回収する（AllGather）の 2 フェーズでブロードキャストを実現する（**図 6**）．このアルゴリズムは，多くの MPI 実装において利用されており，たとえば，GridMPI [21] では，これを応用し，Scatter フェーズ終了後，WAN をまたいだクラスタ間でコネクション数を制限しつつ別クラスタへ対応する複数のノードが同時に転送を行い，個々のクラスタ内で AllGather フェーズを行ってブロードキャストを行う．

クラウドでのブロードキャストとしてこのアルゴリズムをそのまま適用するとすると，ノード数が n のときに，クラウドストレージからのダウンロードを $\log n$ ステップに分けて行うことになり，スループットを向上させる観点からは望ましくない．また，その後の AllGather で各ノードが通信ステップごとに決まったノードとデータ交換をする方式では，偶発的に発生する通信コンテンションやボトルネックリンクによるスループット低下を避けることができず，また，それらに対処するためにノード間のトポロジを考慮したとしても，数千ノードの利用を考えた場合，マルチキャスト実行前に通信相手を決定する静的な通信方法では，つねに最適な性能を発揮することは難しい．

non-steal アルゴリズムでは，scatter と allgather の 2 フェーズに分けてクラウドストレージから計算ノードへ転送を行うという観点で上記の van de Geijn アルゴリズムのアイデアを利用しているが，それぞれのフェーズでのデータ転送方法は，クラウドの動的にネットワーク性能が変化することへの対応を意識したものとなっている．**図 7** は提案アルゴリズムの概要であり，アルゴリズムの詳細は以下のとおりである．

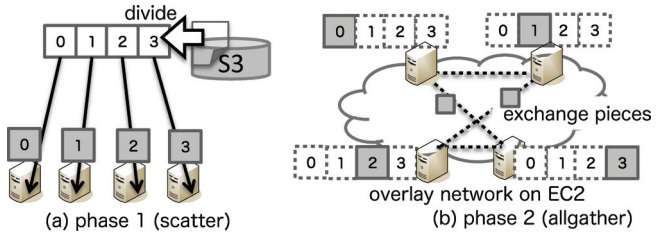

図 7　提案アルゴリズムの概要
Fig. 7　Abstract of proposed algorithm.

phase 1　S3 上のファイルを 32 KB ずつの論理的なピースに分割し，計 P ピースに分割する．ノード数が N のとき，それぞれのノード i に対して $(\frac{iP}{N}, \frac{(i+1)P}{N} - 1)$ の範囲のピースのダウンロードを割り当てる．担当する範囲のピースを各ノードが同時にダウンロードし，すべてを取得し終わったノードは他のノードがダウンロード終えるまで待機する．

phase 2　EC2 ノード間にオーバレイネットワークを構築し，BitTorrent-like なプロトコルを用いて各ノードは未取得のデータを集める．

アルゴリズムを詳細に説明するために例をあげて示す．EC2 上で 3 ノード（A，B，C）を利用するとし，S3 にある 300 MB のファイルを各ノードにブロードキャストをすることを考える．このファイルを 32 KB のピースサイズに分割すると，300 MB のファイルは論理的に 9,600 個のピースで構成される．A，B，C は，それぞれ，0〜3,199，3,200〜6,399，6,400〜9,599 の範囲に該当するピースを S3 からダウンロードする．

各ノードの S3 からのダウンロードスループットが，A は 10 MB/sec，B と C は 2 MB/sec であるとする．このとき，A，B，C が担当する範囲のピースのすべてを S3 からダウンロードする時間は，10 sec，50 sec，50 sec となる．non-steal アルゴリズムでは，他のノードがデータをダウンロードし終えるまで待つので，phase 1 にかかる時間は B，C の性能に律速され 50 sec である．

その後，各ノードは BitTorrent-like なプロトコルでデータの転送を行う．各ノード i はネットワーク的に近いノードとコネクションを確立し，自身が持っているピースリスト

$possession(i)$ を通知する．あるノード j は，未取得のピース p がリストに含まれている場合，$request(p)$ をノード i へ送る．ノード i は該当するピース p をノード j に送る．ノード j は，p を受信後，自分が新たに取得したピース p を他のノードへのコネクションを通じて通知し，受け取ったノードは，ノード j のピースリスト $possession(j)$ を更新する．また，BitTorrent のオリジナルなプロトコルでも提案されている choking/unchoking アルゴリズムを用いて，スループットが高いノードに動的にコネクションを切り替えてネットワーク性能の変化に適応させる．この通信を繰り返すことで，最終的に全ノードがピースを回収してブロードキャストが完了する．

4.3 steal アルゴリズム

non-steal アルゴリズムでは，phase 1 において各ノードが同時に担当範囲のデータを取得することで全体のスループットを向上させているが，各ノードはデータ取得完了後，他のノードが取得完了するまでストールする．図 4 に示したように，EC2 ノードの S3 からのダウンロードスループットがノードごとに異なるため，同じデータサイズであってもノードによって phase 1 の完了時間にばらつきが生じてしまう．そこで steal アルゴリズムでは，自身が担当する範囲のデータをダウンロードし終えたノードが，S3 からのスループットが遅いノードの担当するデータをそのノードの代わりに積極的にダウンロードして S3 からのスループットのロードバランスを行うよう改良した．

non-steal と同様に，A，B，C の担当する範囲のピースはそれぞれ 0〜3,199，3,200〜6,399，6,400〜9,599 である．これをさらに論理的に 100 ピースごとの単位で区切ったものを $work$ と呼ぶ．たとえば，A に関しては，[0-99]，[100-199] のように区切ることができ，$work\ id$ を一意に割り振ると，A が担当する $workList_A$ は，$[0, 1, 2...31]$ になる．各ノードは，$workList$ の先頭から順にリストが空になるまで S3 からダウンロードを行う．すべての $work$ を受信し終えたノードは，他のノードに対して $giveWork$ メッセージを通知する．受信したノードは $workList$ に $work$ が余っている場合，リストの末尾から $\lceil length(workList)/2 \rceil$ 個の $work$ をそのノードに送信する．受信したノードは，自分の $workList$ の末尾に追加し，S3 からのダウンロードをスタートする．今，A が最初の自身に割り当てられた $work$ をすべて受信し終えて，B に $giveWork$ を送信する．10 sec 経過時点での B は 100 MB のうち 20 MB を受信し終えている．これはすなわち，[3,200-3,839] に該当し，$work\ id$ としては [32...37] を消化したことになる．B が持っている残りの $work$ は [38...63] であるので，このとき残っている $work$ の半数の 13 個 [51...63] のダウンロードを A に依頼する．新たな $workList$ はそれぞれ $workList_A = [51...63]$，$workList_B = [38...50]$ となってダウンロードをスタートさせる．

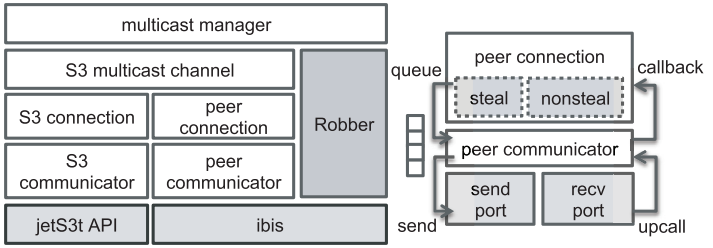

図 8　実装したソフトウェアレイヤ（左）と non-steal，steal プロトコルのメッセージングプロトコル（右）
Fig. 8　Overview of implemented software layer (left) and messaging protocol (right).

steal アルゴリズムによるダウンロードスループットのロードバランスは，Windows Azure ストレージサービスや，他の多くの並列分散ファイルシステムにおいても，個々のリンクのダウンロードスループットが変動する場合，同様の手法を用いて個々のリンクのダウンロードスループット性能を向上させることができるため，その他のシステムに幅広く適用可能であり，このようなシステムにおけるスループット向上のための一般的な解決策として有効であると考える．

4.4 実　　装

提案アルゴリズムを java ベースのグリッドプログラミングプラットフォーム：ibis[22] 上に実装した．**図 8** は，実装したソフトウェアレイヤの概要（左）と non-steal，steal メッセージングプロトコルの詳細（右）を示している．

phase 1 では，各ノードは S3Communicator オブジェクトを介してデータを S3 から取得する．S3 への実際のアクセスには，jetS3t API[23] で提供される REST を用いてダウンロードを行っている．non-steal，steal アルゴリズムは PeerConnection オブジェクトに実装されており，ibis で提供される通信プリミティブである *send port*，*receive port* を介して他のノードと通信を行う．ibis で提供される registry service により，各ノードは現在マルチキャストに参加しているノードの hostname，IP，一意に決定される ID を互いに知ることが可能となる．*receive port* に到着したメッセージは，ibis で提供される upcall によりすべてプログラムに通知される．個々のメッセージに対応したハンドラを呼び出し，メッセージの処理を行う．メッセージの送信は，デッドロックを防ぐため，いったんキューに格納し，*send port* を通じて個々の *receive port* へ送信される．

phase 2 の BitTorret 型の通信は，同じく Ibis 上に実装されている Robber[24] を用いた．

Robber は，WAN で結合されたマルチクラスタ環境での高スループットなマルチキャストを達成するために開発されたソフトウェアで，ネットワークモニタリングデータを必要とせずに他のノードとのスループットを動的に最適化することが可能である．

5. 性能評価

5.1 提案アルゴリズムの性能と安定性

利用する Amazon EC2/S3 環境はともに Europe サイトを用い，EC2 のインスタンスには，*small* を用いて実験を行った．**図 9**，**図 10** は，提案アルゴリズム（steal，non-steal）を用い，8 ノードに対して 1 GB のファイルを 10 回連続してブロードキャストを実行したときの試行ごとの通信完了時間の平均値（average），最大値（slowest），最小値（fastest）を示している．steal，non-steal ともに slowest，average，fastest の通信完了時間の差が flat tree（図 2）に比べて小さく，全ノードが同程度の時間で通信を終えていることが分かる．一方，実験回数を重ねたときの 1 回 1 回の通信時間を比較すると，steal アルゴリズムは，安定して高い性能を達成しているが，non-steal アルゴリズムでは，10 回目の通信時に著しく性能が低下している．これは，8 ノードのうち少なくとも 1 台のノードの S3 からのダウンロードスループットが低く，他のノードがダウンロード終了までストールした影響で，phase 2 の開始が遅れたためである．steal アルゴリズムで同じ状況となった場合，先に S3 からのダウンロードを終えたノードがボトルネックとなっているノードが担当しているダウンロードタスクを動的に譲り受けて S3 からファイルを取得するため，突発的に発生する S3 ダウンロードスループットの低下にも非常に強いことが分かる．

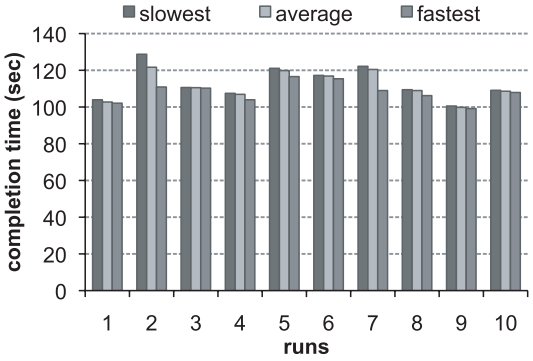

図 9　steal algorithm の通信完了時間（1 GB, 8 nodes）
Fig. 9　Completion time with steal algorithm for each experiment (1 GB, 8 nodes).

5.2 ノード数に対するスケーラビリティ

次に各アルゴリズムのノード数に対するスケーラビリティを確認するため，同様の実験をノード数を変化させて行った．**図 11** では，このときの各アルゴリズムのスループットを比較している．flat tree がノード数が増えるにつれてノード全体としての性能を低下させるのに対して，提案アルゴリズムは，flat tree よりも高い性能で，かつ，ノードの増加によらず一定の性能を得ていることが分かる．32 ノード使用時で，flat tree に対して non-steal

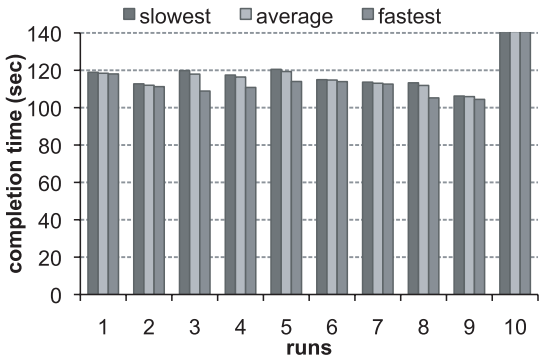

図 10　non-steal algorithm の通信完了時間（1 GB, 8 nodes）
Fig. 10　Completion time with non-steal algorithm for each experiment (1 GB, 8 nodes).

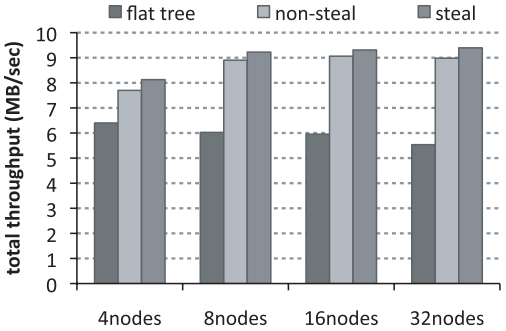

図 11　ノード数に対する合計スループットの平均（1 GB）
Fig. 11　Average total throughput for the number of nodes (1 GB).

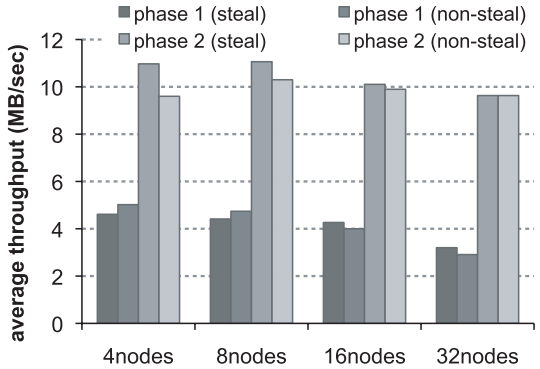

図 12　steal，non-steal アルゴリズムにおける phase 1，phase 2 の平均スループットの内訳（file size: 1 GB）
Fig. 12　Average throughput details of phase 1 and phase2 with steal and non-steal algorithm (file size: 1 GB).

は 1.6 倍，steal で 1.7 倍程度の性能向上が確認された．提案アルゴリズムでは，EC2 内のノード間のデータ転送に BitTorrent-like な通信を用いており，ノード数の増加に対するスケーラビリティの高さを可能としている．

5.3　提案アルゴリズムの性能解析

最後に，提案する steal，non-steal アルゴリズムの性能をより詳細に解析して提案手法の有効性を示す．**図 12** は，steal，non-steal アルゴリズムを用いて 1 GB のファイルを S3 から全ノードに対してブロードキャストした場合の各 phase の平均スループットを示した図である．phase 1 に関しては，steal，non-steal ともに，4，8 ノード使用時で 4，5 MB/sec 程度のスループットを，16，32 ノード使用時では，3，4 MB/sec 程度のスループットが得られた．phase 1 でのスループットはノード数が増えるに従って低下している．これは，ノード数が増えるに従って個々のノードが S3 からダウンロードするデータ量が減るためである．たとえば，4 ノード時では各ノードは初期状態で 256 MB 分のデータを割り当てられてダウンロードするが，32 ノード時では 32 MB 分のデータとなる．一方，phase 2 に関しては，つねに安定して高性能に通信を実行しており，steal，non-steal ともに 10 MB/sec 程度のスループットを達成しているという結果になった．

また，**図 13** は，steal アルゴリズムを用いて 1 GB のファイルをブロードキャストした場合における，各 phase での通信に要した時間の平均値をノード数に応じて示した図である．ノード数が増えるに従って，phase 1 に費やされる時間は減少する一方，phase 2 に費やされる時間は増加していく．16，32 ノード程度を使用した場合，phase 2 の通信時間の全体に対して占める割合が 90%程度にまで達し，ノード数によらず一定の通信時間に近づくことが分かる．ノード数が増えるに従って steal と non-steal アルゴリズムによる性能差は減少するが，steal アルゴリズムはスケーラブルにブロードキャストを実行することが可能となっていることが示されている．

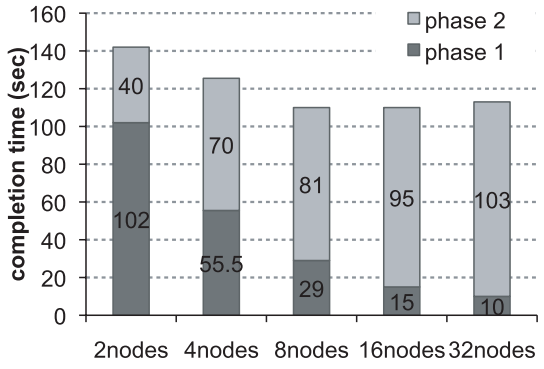

図 13　steal アルゴリズムを用いたときの phase 1 と phase 2 の通信完了時間の比率（file size: 1 GB）
Fig. 13　Completion time ratio of phase 1 and phase 2 with steal algorithm (file size: 1 GB).

6. おわりに

本稿では，Amazon EC2/S3 において，S3 からのデータ転送時に発生しうるボトルネックコネクションに対し，他のノードが動的にワークスティーリングを行うことでノード全体のスループットのロードバランスを行い，さらに，計算ノード間で BitTorrent-like な通信を行うことにより，動的なネットワークトポロジの変化や利用可能なバンド幅が変化しても高いスループットを得ることを可能とするブロードキャスト手法を提案した．単純な手法と比べて，ノード数によらず，1.7 倍程度高いスループットを得ることが確認された．今後の課題として，phase 1 と phase 2 をオーバラップさせることや，マルチコネクションを用いて S3 から得られるダウンロードスループットを向上させること，*steal* する *work size* をノードのダウンロード速度に応じて動的に変更することによる最適化を考えている．

謝辞　本研究の一部は，文部科学省科学研究費補助金（特別研究員奨励費 20・8911，特定領域研究 18049028）の支援，また，Amazon による研究・教育支援プログラム（AWS in Education）の支援によって行われた．

参 考 文 献

1) Amazon Web Services. http://aws.amazon.com/
2) Windows Azure. http://www.microsoft.com/windowsazure/
3) A Toroidal LHC ApparatuS Project (ATLAS). http://atlas.web.cern.ch/
4) Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J.: Basic Local Alignment Search Tool, *Journal of Molecular Biology*, Vol.215, No.3, pp.403–410 (1990).
5) den Burger, M., Kielmann, T. and Bal, H.E.: Balanced Multicasting: High-throughput Communication for Grid Applications, *ACM/IEEE Conference on Supercomputing* (*SC '05*) (2005).
6) Izmailov, R., Ganguly, S. and Tu, N.: Fast Parallel File Replication in Data Grid, *Future of Grid Data Environments workshop* (*GGF-10*) (2004).
7) Chiba, T., Endo, T. and Matsuoka, S.: High-Performance MPI Broadcast Algorithm for Grid Environments Utilizing Multi-lane NICs, *7th IEEE International Symposium on Cluster Computing and the Grid* (*CCGrid*) (2007).
8) Takahashi, K., Saito, H., Shibata, T. and Taura, K.: A Stable Broadcast Algorithm, *8th IEEE International Symposium on Cluster Computing and the Grid* (*CCGrid*) (2008).
9) 柴田剛志，田浦健次朗：トポロジ情報を用いた効率的かつ漸近安定な大容量ブロードキャスト，情報処理学会論文誌：コンピューティングシステム（ACS），Vol.2, No.3, pp.47–57 (2009).
10) Cohen, B.: Incentives build robustness in BitTorrent, *Workshop on Economics of Peer-to-Peer Systems* (2003).
11) Castro, M., et al.: Splitstream: High-bandwidth multicast in cooperative environments, *19th ACM Symposium on Operating Systems Principles* (2003).
12) den Burger, M. and Kielmann, T.: MOB: Zero-configuration High-throughput Multicasting for Grid Applications, *16th IEEE International Symposium on High Performance Distributed Computing* (*HPDC '07*) (2007).
13) Allcock, W.: GridFTP: Protocol extensions to FTP for the Grid, *Global Grid Forum GFD-R-P.020* (2003).
14) Bresnahan, J., Link, M., Kettimuthu, R. and Foster, I.: GridFTP Multilinking, *Proc. 2009 TeraGrid Conference* (2009).
15) Garfinkel, S.L.: An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS, Technical Report, Center for Research on Computation and Society School for Engineering and Applied Sciences, Harvard University (2007).
16) Palankar, M.R., Iamnitchi, A., Ripeanu, M. and Garfinkel, S.: Amazon S3 for science grids: A viable solution?, *DADC '08: Proc. 2008 international workshop on Data-aware distributed computing*, New York, NY, USA, ACM, pp.55–64 (2008).
17) Walker, E.: Benchmarking Amazon Ec2 for high-performance scientific computing, *LOGIN*, pp.18–23 (2008).
18) 建部修見，曽田哲之：広域分散ファイルシステム Gfarm v2 の実装と評価，情報処理学会研究報告（2007-HPC-113），pp.7–12 (2007).
19) Apache Hadoop. http://hadop.apache.org/
20) Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google File System, *19th ACM Symposium on Operating Systems Principles* (*SOSP-19*) (2003).
21) Matsuda, M., Kudoh, T., Kodama, Y., Takano, R. and Ishikawa, Y.: Efficient MPI Collective Operations for Clusters in Long-and-Fast Networks, *IEEE International Conference on Cluster Computing* (*Cluster 2006*) (2006).
22) van Nieuwpoort, R.V., et al.: Ibis: A Flexible and Efficient Java based Grid Programming Environment, *Concurrency and Computation: Practice and Experience*, Vol.17, No.7-8 (2005).
23) jetS3t. http://jets3t.s3.amazonaws.com/
24) den Burger, M. and Kielmann, T.: Collective Receiver-initated Multicast for Grid Applications, *IEEE Trans. Parallel and Distributed Systems* (*TPDS*) (2009).

(平成 21 年 10 月 2 日受付)
(平成 22 年 2 月 10 日採録)

千葉　立寛（学生会員）

1983 年生．2006 年東京工業大学理学部情報科学科卒業，2008 年同大学大学院情報理工学研究科数理・計算科学専攻修士課程修了．同年より同博士課程在学中，および，同年より日本学術振興会特別研究員 DC1．主に広域分散並列環境での通信最適化に関する研究に従事．ACM 会員．

ティロ キールマン

1997 年ドイツ・ダルムシュタット工科大学コンピュータサイエンス専攻・博士（情報工学）．2001 年同大学，およびドイツ・ジーゲン大学からコンピュータサイエンス分野の大学教授資格を得る．1998 年よりオランダ・アムステルダム自由大学コンピュータサイエンス専攻・准教授．主に高性能計算，分散コンピューティング，グリッド・クラウドアプリケーションのプログラミング環境やランタイムシステムに関する研究に従事．OpenGrid Forum，および Gridforum Nederland の各運営委員．

マタイス デン バーガー

2002 年オランダ・アムステルダム自由大学コンピュータサイエンス専攻修士課程修了，2009 年同大学同専攻・博士（情報科学）．2006〜2007 年，電子メールの署名と暗号化に関する仕事にソフトウェアエンジニアとして携わる．現在，アムステルダム自由大学・コンピュータサイエンス専攻の博士研究員．主に高性能計算，分散システムの設計・性能評価に関する研究に従事．

松岡　　聡（正会員）

東京大学情報科学科・情報科学専攻・博士（理学）（東京大学，1993 年）．2001 年より東京工業大学学術国際情報センター・研究教育基盤部門・教授．専門は高性能システムソフトウェア・グリッド・並列処理等．最近のプロジェクトは NAREGI プロジェクト（サブリーダー，2003〜2007 年），東工大キャンパスグリッド（2002〜2006 年），科研特定領域・情報爆発（柱長，2006〜2010 年），JST-CREST Ultra Low Power HPC（2007〜2011 年）等．2006 年に構築したスパコン TSUBAME は，わが国トップスパコンの地位を 2 年間維持，世界最大の GPU スパコン等多くの話題を呼ぶ．ACM OOPSLA 2002，IEEE Grid2003/2006，ACM/IEEE Supercomputing 2009 の論文委員長等，数多くの国際学会の大会委員長・プログラム委員長を歴任．情報処理学会坂井記念賞（1997 年），学術振興会賞（2006 年），ISC Award（2009 年）等を受賞．