MapReduceシステムのネットワーク負荷分析

森　達哉，木村　達明，池田　泰弘，上山　憲昭，川原　亮一

MapReduceを実行する大規模分散システムの性能評価モデルの構築に向け，MapReduceによる大規模データ処理を実行した際にシステム全体に生じる負荷をネットワークの観点から分析したケーススタディを紹介する．

キーワード：MapReduce，ネットワーク計測，負荷分析

1. はじめに

MapReduce[4]はGoogleが開発した分散プログラミングフレームワークであり，Webデータに代表される非常に大規模なデータを効率的に処理する目的などに利用される．MapReduceフレームワークを用いたデータ処理の利点は，分散システムを構成するサーバを増やすことで処理能力の線形増加が期待できる点，すなわちスケール・アウト性を有する点にある．システムを構成する個々のサーバは必ずしもハイエンドである必要はなく，全体として高性能なシステムを比較的低コストで構築できる利点もある．MapReduceフレームワークの利用はGoogle社内のみに留まらず，MapReduceのオープンソース実装であるHadoop[1]の普及をきっかけとして今日では国内外の様々な企業・組織に利用されている．

スケール・アウト性を有するMapReduceの実行性能を向上させる最もシンプルな方法は，サーバの数を増やすことである．しかしながら，単純にサーバを追加するアプローチは非効率な結果を招く可能性がある．すなわち，目標とする性能に対して必要となる資源の適切な見積もりができない限り，分散システムを構成するハードウェアへの投資だけでなく，ハードウェアの利用に必要な電力を大量に浪費する恐れがある．

この課題を解決するためには，MapReduceを実行する分散システムの性能評価モデルを構築するアプローチが有望である．一方，MapReduceシステムの構成要素，およびそれらを特徴付ける変数は複雑かつ多岐にわたる．例えばサーバ一つをとっても性能に関係する変数としてメモリ，CPU，ディスクI/O，ネットワークI/O等が挙げられ，これらがMapReduceフレームワークの上で相互に影響を及ぼしている．さらにMapReduceが実行される分散システムは一般に多数のサーバから構成されるため，性能モデルの構築はチャレンジングな課題である．

本稿ではMapReduceシステムの性能評価モデルの構築に向け，システムの基本的な負荷分析のケーススタディを実施した結果を紹介する．負荷分析を行う目的は，複雑で大規模なシステムの挙動を観察することによって主要な変数間の相関関係を明らかにすることである．このようなアプローチにより，性能評価モデルに取り入れるべきエッセンスを明確にすることが狙いである．MapReduceシステムの構成要素は上述したように多岐にわたるが，本稿では大規模データを処理する分散システムにおいて，データ転送のボトルネックとなるネットワークI/Oに焦点を当てた分析を行う．

2. MapReduceフレームワークの概要

はじめにMapReduceの基本的なフレームワークについて概説する．MapReduceはMapおよびReduceと呼ばれる二つのシンプルなデータ処理を組み合わせて実行することにより，巨大なデータを効率的にバッチ処理するためのフレームワークである[4]．MapReduceフレームワークの利点のひとつは前述したスケールアウト性であるが，もう一つの特筆すべき利点は利用者にとっての敷居の低さである．MapReduceフレームワークはシステムレベルの詳細をプログラマーに対して隠蔽する．この抽象化により，プログラマーは並列分散処理の詳細を一切気にすることなく，MapとReduceの組み合わせを行うだけで大規模データの並列分散処理ができる[6]．

もり　たつや，きむら　たつあき，いけだ　やすひろ，かみやま　のりあき，かわはら　りょういち
NTTサービスインテグレーション基盤研究所
〒180-8585 武蔵野市緑町3-9-11

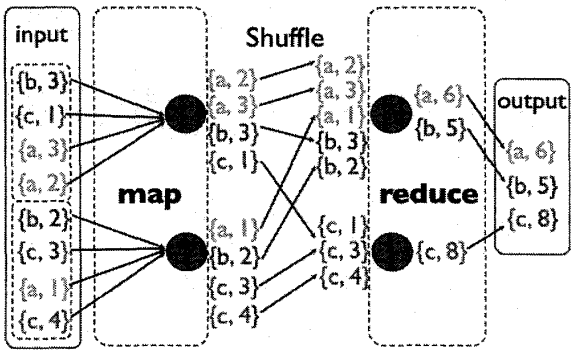

図1 MapReduceによるデータ処理の流れ

以下では図1を用い，MapReduceの動作を実際のデータ処理の流れに沿って示す．図における○はサーバを示す．MapReduceが処理する入力データはより細かい単位（ブロック）に分割され，後述するMapとReduceのそれぞれを実行する複数タスクによって並列分散的に処理される．ここでは便宜上4台のサーバが存在し，それぞれがMapおよびReduceタスクを1つずつ実行している例を図示しているが，単一のサーバが複数のタスクを同時に実行することも可能である．

MapReduceに対する入出力は一般にキー・バリューのペア$\{k, v\}$の形式で与える．図ではinput，すなわち$\{b, 3\}, \{c, 1\}, \cdots, \{c, 4\}$が入力であり，出力はoutput，すなわち$\{a, 6\}, \{b, 5\}, \{c, 8\}$である．この例では，個々のキーに対して与えられたバリューを，キーごとに総和をとる処理を行っている．このような処理の具体例として，ユーザ（キー）ごとの通信量（バリュー）を記録した複数のログデータを集計して，ユーザごとの総通信量を計数する処理などが挙げられる．

MapReduceに対する元の入力データは小さなブロックに分割され，各々のブロックはMapを実行するタスクに割り当てられる．図の例では入力は2つのブロックに分割され，各々がMap処理を実行するMapタスクに割り当てられている．さらに2つのMapタスクはそれぞれ1台のサーバに割り当てられている．各々のMapタスクはブロックを処理した結果を仕分けて中間出力として蓄積する．図中のMapタスクを実行するサーバの右に位置する$\{a, 2\}, \{a, 3\}, \cdots, \{c, 4\}$がMapの中間出力である．この例におけるMapの処理は入力をキーでソートすることに相当する．

Mapの中間出力はReduceを実行するReduceタスクへ転送される．このときに同じキーのデータは同一のReduceタスクにコピーされる．図の例ではキーa，bが上段のReduceタスクに，キーcが下段のReduceタスクにコピーされる．Mapの中間出力をReduceにコピーする処理は**シャッフル**と呼ばれ，中間出力が大きい場合はシャッフル時に大量のデータがネットワーク経由で転送される．この結果，Reduceタスクへの入力は，上段のReduceタスクが$\{a, 2\}, \{a, 3\}, \cdots, \{b, 2\}$，下段のReduceタスクが$\{c, 1\}, \{c, 3\}, \{c, 4\}$となる．

最後にReduceタスクは入力されたキー・バリューペアに対する処理を行い，最終的なキー・バリューペアを出力する．図中の例では各キーに対するバリューの総和をとった結果が最終出力となる．以上で示したMapとReduceによるデータ処理を複数組み合わせることによって，より複雑なデータ処理が実現できる[3][6]．

3. 実験方法

MapReduceシステムの負荷分析を行う方法として，実機を用いるアプローチを採用する．MapReduceの実装としては広く普及しているHadoopを用いる．Hadoopの詳細については本特集の「Hadoopを活用した大規模データ解析の動向と今後の展望」を参照されたい．以下ではMapReduceの設定とネットワーク負荷計測方法を示す．

3.1 MapReduceの設定

前節でみたように，Mapタスクの中間出力データからReduceタスクの入力データを構成するために必要となるデータはネットワーク上でコピーされる（シャッフル）．すなわちMapタスクの中間出力が巨大である場合にはシャッフル時に大量データが一斉にネットワーク転送されるため，スイッチングハブ等のネットワーク装置でトラヒック輻輳が生じる可能性がある．

MapReduceのシャッフル時の動作に関連する基本的なパラメタを調整し，ネットワーク負荷とシステムの全体性能にどのような相関関係があるかを実験的に分析する．この目的のために採用したHadoopのパラメタはmapred. reduce. tasksとmapred. reduce. parallel. copiesである．これらのパラメタは各々「MapReduceジョブに対して実行されるReduceタスク数」と「ReduceタスクがMap中間出力ファイルを取得する際の並列コピー数」を調節する．以降，本

稿ではこれら2つのパラメタを表現する変数をr（Reduceタスク総数）およびp（並列コピー数）とする．rがクラスタを構成するサーバ数よりも小さい場合，シャッフルのデータ転送負荷は一部のサーバに集中する．pが大きい場合，シャッフル時にMapタスクの中間出力をコピーする通信の並列数が大きくなる．その他のMapReduceおよびHDFSのパラメタはデフォルト値を固定的に用いた．

本稿で分析対象とするMapReduceジョブは，特にデータ転送負荷が高い，すなわちI/Oバウンド高いものとする．そのような条件を満たすMapReduceプログラムとして，Hadoopの標準的なベンチマークプログラムのひとつである，TeraSortを採用した．TeraSortプログラムを用い，100 GB（1 TBではない）のデータをソートする．100 GBのデータ構成はサイズが100バイトのランダムかつユニークな文字列10億個を並べたものである．

3.2 ネットワーク負荷の監視

Hadoopはシステムレベルの様々なイベントをログとして記録するが，ネットワークI/Oに関する詳細な情報，例えば時間精度が高いデータ転送流量の変動値や，TCP通信[1]における再送数といった詳細な情報は記録しない．しかしながら，ネットワークI/Oに関連した性能評価や通信輻輳の原因究明には詳細なネットワーク負荷情報の把握が必要不可欠である．

詳細なネットワーク負荷情報を収集するひとつの手段として，パケットヘッダ情報をキャプチャする方法がある．パケットヘッダ情報を詳細に分析することにより，例えば1秒間隔のスループット変動や，TCPの再送回数の把握が可能となる．一方，今回の実験に用いたHadoopクラスタを構成するサーバ数は12であるので，クラスタ内で生成し得るトラヒック流量はin/outの合計で$12\times2\times1$ Gbps$=24$ Gbpsとなる．したがって，すべてのインタフェースで送受信されるパケットヘッダ情報をスイッチングハブのポートミラーリング機能等を利用して集中的に収集する方法はコストが高い．そこで，各々のサーバでパケットヘッダキャプチャを分散的に行い，後からすべてのヘッダ情報を集約するアプローチを取る．パケットヘッダ情報をキャプチャするプログラムとしてはtcpdump[5]を用い，各サーバでパケットヘッダ情報に印加される時刻情報が同期するように設定した．簡易な実験により，パケットキャプチャプロセスによるCPU負荷は無視できるものであることを確認した．またパケットヘッダの取得サイズは1パケットあたり64バイトに抑えたので，パケットヘッダ収集に伴うディスクI/Oへのオーバーヘッドも小さなものであった．

4. 負荷分析

本節ではMapReduceジョブとネットワークの負荷の関係を分析したケーススタディを示す．図2に

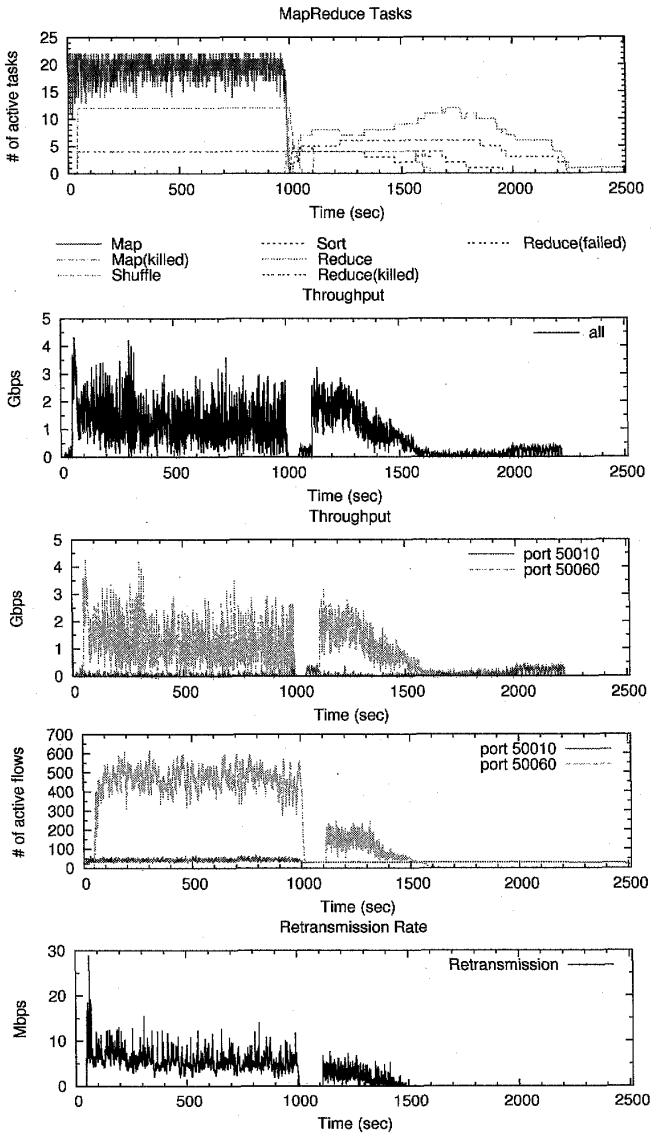

図2 MapReduceタスクの負荷（最上段），ネットワーク負荷（2段目），主要なポート番号ごとのネットワーク負荷（3段目），主要なポート番号ごとのアクティブフロー数（4段目），TCPによる再送トラヒックレート（5段目）．設定したMapReduceパラメタは$r=16$および$p=10$．

[1] TCP（Transmission Control Protocol）はインターネットで広く使われる伝送制御方式であり，パケット損に対する再送機能やウィンドウ制御による輻輳回避機能を兼ね備えている．

表1　図2に示すMapReduceジョブのパラメタ設定値

パラメタ (プロパティ名)	意味	変数名	値
mapred.tasktracker.map.tasks.maximum	サーバ毎の同時実行 Map タスク数	c_m	2
mapred.tasktracker.reduce.tasks.maximum	サーバ毎の同時実行 Reduce タスク数	c_r	2
mapred.map.tasks	ジョブあたりの合計 Map タスク数	m	1,504
mapred.reduce.tasks	ジョブあたりの合計 Reduce タスク数	r	16
mapred.reduce.parallel.copies	シャッフル時の同時データ転送数	p	10

MapReduceジョブを構成するタスクレベルの負荷と詳細なネットワーク負荷情報の関係を示す．MapReduceジョブに対して設定したパラメタは表1の通りである．

4.1　MapReduceジョブの負荷

はじめに，MapReduceジョブのタスクレベルでの負荷に着目する．MapRduceジョブのログを分析することによって，ジョブを構成するすべてのタスクの状態とその変化を追跡することができる．例えばあるタスクがスタートしたらアクティブタスク数を加算し，あるタスクが終了したらアクティブタスク数を減算することによってアクティブな同時実行タスク数を把握することが可能である．ジョブを構成するタスクは大まかにMap，シャッフル，Reduceの順に実行される．後に明らかになるように，シャッフルおよびReduceにおいてネットワークの負荷が高くなる傾向にある．

Hadoopの実装ではタスクの投機的実行，すなわち将来の失敗をみこした冗長なタスクの実行が可能である．Hadoopの投機的実行では通常よりも時間がかかっているタスクを検出した場合，同一処理を行うバックアップのタスクが起動され，先にどちらかのタスクが終了するともう一方のタスクが強制終了される．したがって，投機的実行が行われる場合にはタスクが実行される順番は必ずしも上記で示したようにMap，シャッフル，Reduceの順番とはならないケースがある．タスクの実行がどのような順番となるかはその時々のシステムの状態に依存するため，非決定論的な振る舞いとなる要素を含んでいる．

以下では各パラメタがMapReduceジョブにどのような影響を与えるかを示す．図2の最上段は今回実行したMapReduceジョブのうち，表1のパラメタ設定を行ったジョブについて，各タスクの同時実行状態を示した図である．最初にジョブの開始時点では，スレーブノード数10にノードごと同時Mapタスク実行数c_mを乗じた20個のMapタスクが起動する．これらのMapタスクは比較的小さいため，個々のタスクはすぐに終了するが，すぐに次のMapタスクが起動し，アクティブなMapタスク数がつねに20となるようにスケジューリングされている．ジョブ開始後40秒付近でr=16のReduceタスクが起動されている．ここで16の内訳は，凡例にある“Shuffle”と“Reduce (killed)”であり，このうち12のタスクは問題なく完了したが，残りの4つのタスクは投機的実行によって途中で強制終了されたことを示している．投機的実行によりバックアップのReduceタスクがジョブ開始から約1,100秒に起動されていることもわかる．シャッフル時に各々のReduceタスクはMap中間ファイルをp個の同時セッションを使って取得する．したがって，pが増加するにつれ，単一のReduceタスクはより多くのデータ取得のための通信セッションを起動する．

4.2　ネットワーク負荷

つぎにMapRedeceジョブによって発生したネットワーク負荷の詳細を分析する．図2の最上段と二段目を比較すると，アクティブなMapReduceタスク数とネットワーク負荷の間にある程度の相関があることがわかる．特にトラヒック流量が増加する期間はシャッフルタスクが存在する期間と一致していることが見て取れる．今回実行したTeraSortジョブにおいて，Mapタスクが出力する中間ファイルはユニークなキー・バリューペアであるため，Reduceタスクはシャッフル時に入力と同じ数のキー・バリューペア（すなわち10億のペア）をネットワーク経由でコピーする必要が生じる．このため，シャッフル時にはReduceタスクによる中間出力ファイルの同時コピーの通信が発生し，結果としてトラヒック流量が増大する．

3段目の図はTCPのポート番号別のトラヒック流量であり，特に流量が多かった2つのポート50010と50060に着目している．ポート50010はHDFSへの読み書きに，ポート50060はタスクトラッカー間通信に用いられる．後者は特にシャッフルに伴うデータ転送に用いられる．最上段の図と合わせることにより，最初と2番目のトラヒックバーストはシャッフルタス

クと対応し，最後のバーストはReduceタスク，すなわちHDFSへの書き込みに対応することがわかる．これはHadoopの実装としても正しい挙動である．なお，第2のバーストは投機的実行によるバックアップのReduceタスクによるものである．

4段目の図は上述の2つのTCPポート番号に対応するアクティブなフロー数の遷移である．ここでフローとは同一の属性を持つパケットの集合であり，例えば｛VLAN ID，発着IPアドレス，発着ポート番号，プロトコル番号｝のようなtupleによって定義される．シャッフル（ポート50060番）についてはタスク数とアクティブなフロー数が相関していることがわかる．一方，HDFSに関しては必ずしも相関関係は無い．HDFSの通信では長期継続するフローが通信チャネルとして用いられるため，フロー数は大まかに$32(=c_m \times r)$となっている．

最後に，最下段の図はTCP転送における再送パケットのトラヒック流量を示している．シャッフルフェーズにおいてより多くのパケット再送が生じていることが見て取れる．ここで注目すべき事実は，タスク数は，タスクが生成する通信を実行するTCPフロー数および再送レートと比例関係にない点である．例えば第2のトラヒックバーストは第1のバーストよりもサイズが大きいが，トラヒックを構成するシャッフルのタスク数やトラヒックを構成するTCPフロー数は少ない．さらにTCP再送は第1のバーストにおいてより顕著であることがわかる．次節で考察するように，MapReduceジョブのパラメタがネットワークI/Oの特性にどのような影響を与えるかを理解することは，MapReduceジョブ全体の性能評価モデルの構築をする上で必要不可欠なパーツである．

4.3 パラメタと性能

図3にMapReduceジョブのパラメタと性能の関係を調べた結果を示す．バイトロス率とは全通信量（バイト）のうち，パケットがロスしたことによって欠損したとみなすことができる通信の割合である．上段の図よりReduceタスク数rの増加により，TeraSortのMapReduceジョブの実行時間が短くなる傾向にあることがわかる．この結果は並列化に伴う性能向上を示すものであり，直感と一致する．一方，シャッフル時の同時データ取得数pの増加は必ずしも性能の向上にはつながらないことがわかる．pはネットワーク通信の並列性を調節するパラメタであるが，この結果

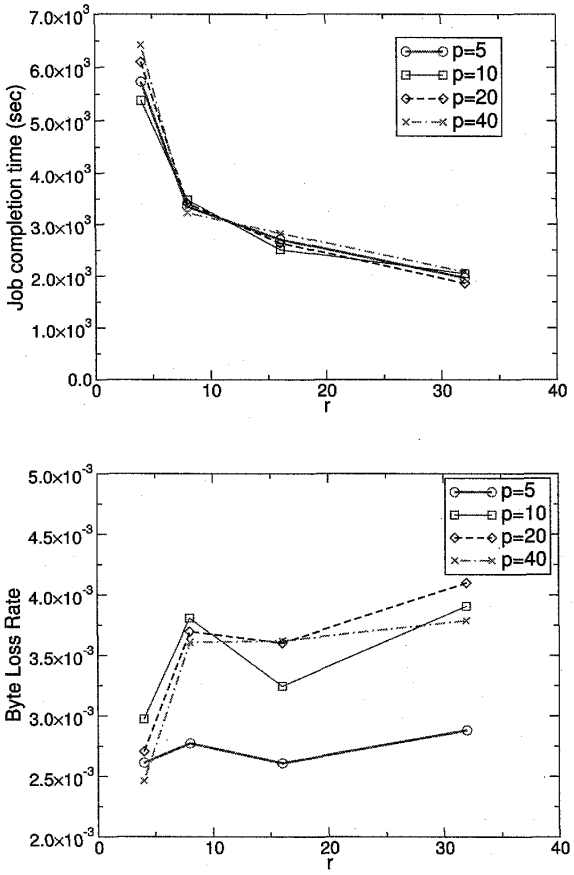

図3 MapReduceパラメタと性能の関係．ジョブ完了時間（上段）とバイトロス率（下段）．

は通信の並列性を高めることが必ずしも性能向上につながらないケースを例示している．

下段のグラフからはrとpの増加に伴いTCP通信におけるバイトロス率が増加することが読み取れる．ただしバイトロス率の値そのものはそれほど高くないため，MapReduceジョブの実行性能に与える影響は大きくないことがわかる．すなわち，シャッフル時にはトラヒック流量が増えるものの，今回実験したパラメタ空間ではネットワークI/Oがボトルネックにはなっていないことがわかる．

パラメタpをさらに増加するかノード数を増加させた上でさらにrを大きくすることにより，性能の向上が見込まれる一方，ネットワーク輻輳が起こる可能性もある．例えば文献[2][7][8]では多対1の通信を行う分散システムにおいてデータリクエストのタイミングが同期することによってTCPの輻輳崩壊が生じる現象—*TCP Incast*—が報告されている．MapReduceのような分散システムはそのような現象を引き起こすシステムの一例と考えられているが，今回実施した実験の範囲内ではTCP Incastの存在は確認され

なかった．MapReduceを実行するシステムにおいてTCP Incastが生じ得るパラメタ空間や条件を明らかにするためには大規模なノードを利用した広範な実験あるいはシミュレーションによる実証的な手法が必要であり，今後の課題である．

4.4 ネットワーク・フロー特性分析

最後に，MapReduceシステムにおけるネットワーク負荷をフローレベルで分析した結果を示す．フローは多くの場合アプリケーションレベルで発生する通信の最小単位であり，トラヒックのモデル化を構成するための基本的な単位である．また，フローはネットワーク監視，流量制御，フィルタリング等，様々な用途で用いられるため，フローの特性を分析することは応用上も意義深いものである[9]．

図4はHDFSおよびTaskTrackerの通信に対するフローサイズの累積補分布である．フローサイズはフローごとの合計通信量をバイト数でカウントしたものである．はじめにフローのサイズはある特定値に偏っていることがわかる．例えばHDFS（ポート50010番）の場合，フローサイズは500+B，1+KB，2+KB，4+KB，…，64+MBにピークがある．サイズが小さなフローはHDFSデーモンの通信メッセージ

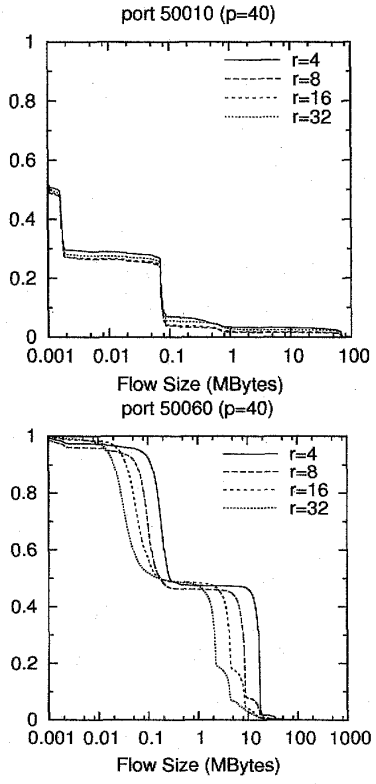

図4 フローサイズの累積補分布．HDFS通信（左）とTaskTracker通信（右）．

であるが，最大値である約64 MBは今回の実験系におけるHDFS上のファイルを構成するブロックのサイズと一致する．これらのブロックサイズと一致するフローはHDFSによる通信トラヒック量の88%以上を占めている．すなわち，HDFSの通信は大部分がブロックの転送に費やされている．ブロックサイズを変更することにより，フローサイズの特性も変わることがわかる．

HDFS通信のフローサイズ分布はジョブごとの総Reduceタスク数であるrとは無相関である．一方，TaskTracker通信のフローサイズ分布はrと明らかな相関がある．すなわち，HDFSのブロックサイズをC [MB]とすると（今回の実験系ではC=64），フローサイズの分布はC/r [MB]に集中していることがわかる．この結果は以下のように説明ができる．今回の評価で採用したTeraSortプログラムにおいてMapタスクが出力する中間ファイルのサイズは，Mapタスクへの入力となるブロックサイズとほぼ一致する．なぜならプログラムへの入力データは重複のないレコードの集合であり，MapReduceジョブが実行する処理はレコードのソートであるため，入力と出力のデータサイズがほぼ同じになるからである．また，今回のプログラムではMapの中間出力をReduceに割り当てるパーティション関数はランダムなハッシュ関数であるため，Mapタスクの中間出力はほぼ均等にReduceタスクへと転送される．以上をまとめると，1つのMapタスクの出力サイズは約C [MB]であり，r個のReduceタスクへと転送される．TaskTrackerによるシャッフル時の転送量は約C/r [MB]となる．

つぎにフローサイズがしきい値を超えるフローについて，フロー継続時間とフロー転送レートの分布を分析した．しきい値はHDFS通信についてはC [MB]，TaskTrackerについてはC/r [MB]とした．図5と6に結果を示す．どのケースにおいてもReduceタスク数rの増加にともない，フロー継続時間は長くなり，転送レートが低くなることがわかる．特にTaskTracker通信についてはこの傾向が顕著である．前節でみたように，rの増加は分散処理の並列度を高めることであり，本研究のパラメタ空間の範囲では実行性能が向上した．しかしながら同時に多数のReduceタスクが動作すると，中間ファイルをMapから取得する際にネットワークリソースの奪い合いとなる．このため個々のフロー転送レートは低下し，転送完了までに時間がかかるようになる．このような競合による通

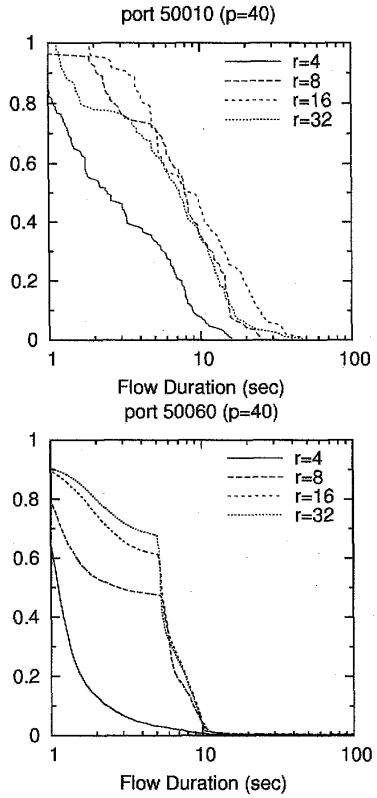

図5 フロー継続時間の累積補分布．HDFS 通信（左）と TaskTracker 通信（右）．

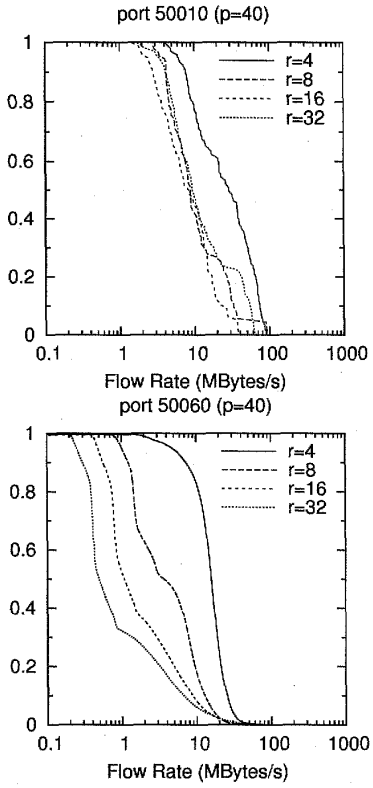

図6 フロー転送レート時間の累積補分布．HDFS 通信（左）と TaskTracker 通信（右）．

信のオーバーヘッドが全体の性能に与える影響はパラメタ依存であるが，この結果は少なくとも単純に処理の並列度を高めることが必ずしも性能の最適化にはつながらない可能性を示唆している．MapReduce ジョブの全体的な性能をモデル化するためには，個々のノードにおける容量，処理能力に加えてジョブの種別，およびネットワークレベルでのリソース競合，およびリソース競合時における通信の挙動を司るトランスポートプロトコルの特性やネットワーク機器の容量等を考慮する必要がある．

5. おわりに

大規模なリソースを消費する MapReduce システムの運用を最適化するためには性能評価モデルの構築が必要である．その課題に向けた最初の第一歩として，本稿では MapReduce システムの基本的な負荷分析を実施した結果をケーススタディとして紹介した．特にネットワーク I/O に焦点を当てた分析を行い，データシャッフル時のデータ転送に伴うネットワーク I/O 負荷の増大や，データ処理・転送の並列度を高めることに伴うネットワーク I/O のオーバーヘッドの増大を示した．これらの結果は，ネットワークリソースの競合やトランスポートプロトコルの挙動が MapReduce システム全体の性能と相関することを示唆する．本稿で紹介したネットワーク負荷分析をベースの一つとしてより詳細な性能評価モデルを構築し，評価することは今後の課題である．

参考文献

[1] Hadoop, http://hadoop.apache.org/

[2] Y. Chen, R. Griffith, J. Liu, R.H. Katz and A.D. Joseph, Understanding tcp incast throughput collapse in datacenter networks, In *WREN '09 : Proceedings of the 1st ACM workshop on Research on enterprise networking*, pp. 73-82, New York, NY, USA, 2009, ACM.

[3] C.T. Chu, S.K. Kim, Y.A. Lin, Y. Yu, G.R. Bradski, A.Y. Ng and K. Olukotun, Map-Reduce for Machine Learning on Multicore, In B. Schölkopf, J.C. Platt and T. Hoffman, editors, *NIPS*, pp. 281-288, MIT Press, 2006.

[4] J. Dean and S. Ghemawat, MapReduce : Simplified Data Processing on Large Clusters, *Commun. ACM*, 51 (1) : 107-113, 2008.

[5] V. Jacobson, C. Leres and S. McCanne, TCPDUMP, http://www.tcpdump.org/

[6] J. Lin and C. Dyer, *Data-intensive Text Processing with*

MapReduce, Morgan and Claypool Publishers, 2010.

[7] A. Phanishayee, E. Krevat, V. Vasudevan, D.G. Andersen, G.R. Ganger, G.A. Gibson and S. Seshan, Measurement and analysis of TCP throughput collapse in cluster-based storage systems, In *Proc. USENIX Conference on File and Storage Technologies*, San Jose, CA, Feb. 2008.

[8] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D.G. Andersen, G.R. Ganger, G.A. Gibson and B. Mueller, Safe and effective fine-grained tcp retransmissions for datacenter communication, *SIGCOMM Comput. Commun. Rev.*, 39(4): 303-314, 2009.

[9] 川原，森，上山，IP フロー計測技術の応用，電子情報通信学会誌，93(4): 287-292, 2010.