Hadoopのノード削除時のレプリカ生成の高速化手法の提案

日開　朝美[1]　竹房　あつ子[2]　中田　秀基[2]　小口　正人[1]

概要：大規模データに対応した処理システムとして，汎用なハードウェアを用いて高度な集約処理を行う分散ファイルシステムに注目が集まっている．本研究では，Apache Hadoop の基盤技術である Hadoop Distributed File System (HDFS) に着目した．Hadoop はスケールアウトするシステムであり，システム規模に応じた運用や故障の発生などにより，ノードを脱退および削除することが想定される．HDFS では通常複数の DataNode 上に複数のレプリカを保持し，ノード脱退時および削除時には他のノードにレプリカを自動的に生成する．この過程が長くなると一部の DataNode に負荷が掛かりスループットが低下してしまうため，その高速化が重要である．本稿では，ノード削除時に注目して不足分を補うレプリカ生成処理を高速化する制御手法を提案する．予備調査から，レプリカ生成時のデータ移動には偏りが生じ，効率の良い処理が行われていないことが分かった．そこでレプリカ生成先および生成元を制御することでその偏りを解消し，処理を高速化する制御手法を提案する．制御手法により偏りが解消され，スループットが最大59%向上することを示す．

A Proposal of Effective Replica Generation Schemes at Node Deletion for Hadoop

ASAMI HIGAI[1]　ATSUKO TAKEFUSA[2]　HIDEMOTO NAKADA[2]　MASATO OGUCHI[1]

1. はじめに

近年，通信技術や情報処理技術の発展と普及に伴い，データ量が爆発的に増加し，大規模データを効率良く処理することが求められている．そこで汎用のハードウェアを用いて大規模データの高度な集約処理を行う分散ファイルシステムに注目が集まっている．分散ファイルシステムは，高いスケーラビリティを持つため，システムの規模に応じた運用が見込まれる．また，従来では想定されていなかったような極めて大規模なシステムが構成されるようになった．しかしながら全てのマシンを正常に動作させることは難しく，常に一定の割合で故障や動作不安定なマシンが存在してしまうという問題が新たに生じる．

Hadoop Distributed File System(以下 HDFS)[1] は，Apache Hadoop(以下 Hadoop)[2] の基盤技術であり，Google の分散ファイルシステム GFS[3] に基づいて設計された．Hadoop はスケールアウトするシステムであり，システム規模に応じた運用や故障の発生などにより，ノードを脱退もしくは削除することが想定される．HDFS では通常複数の DataNode 上に複数のレプリカを保持し，ノード脱退時および削除時にはレプリカを自動的に生成する．この過程が十分に速くないと一部の DataNode に負荷がかかり性能が低下してしまうため，その高速化が重要である．

本稿では，HDFS においてノード削除時のレプリカ生成処理を高速化する手法を提案する．デフォルトの設定では，レプリカ生成処理のデータ移動にはノード間で偏りが生じ，効率の良い処理が行われていないことが分かった．そこでその偏りを解消するために，レプリカ生成先と生成元を制御する手法を提案し実装した．評価実験から，提案手法によりノード間のデータ移動の偏りが解消され，スループットが最大 59%向上することを示す．

[1] お茶の水女子大学
Ochanomizu University
[2] 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology(AIST)

2. レプリカ生成の流れ

2.1 Hadoop Distributed File System

Hadoop は Apache Software Foundation で開発されている分散コンピューティング関連のプロジェクト群で，様々なサブプロジェクトの集合体である．HDFS はそのサブプロジェクトの一つで，Google の分散ファイルシステム GFS に基づいて設計されたオープンソースの分散ファイルシステムである．HDFS は，ファイルのメタデータやクラスタ内のノード管理を行う一台の NameNode と，実際にデータを格納し，処理している複数台の DataNode からなる．ファイルを最小単位であるブロックに分割し DataNode 間で分散して保存することで，耐故障性と対多クライアントでの高い性能を維持している．

2.2 ノード脱退とノード削除

HDFS はクラスタからノードを切り離す際，該当ノードが保持しているデータのレプリカを残りのノードに移行することで，指定したレプリカ数を維持する．Hadoop にはクラスタからノードを切り離す方法として，ノード脱退とノード削除の二種類が存在する．

ノード脱退とは，事前に脱退ノードが保持しているデータのレプリカを他のノードに複製してからそのノードをクラスタから切り離す方法である．この時，脱退ノード自身もレプリカ生成元として処理に参加する．クラスタ規模の縮小やエラー率が非常に高いノードが存在する時など，意図的にクラスタからノードを切り離す場合を想定している．

ノード削除とは，クラスタからノードが離脱後，残ったノード間で削除ノードが保持していたデータのレプリカを複製し補完する方法である．削除ノードがレプリカ生成処理に参加することはない．ノードの故障や動作不良あるいはネットワークの切断により，想定外にノードがクラスタから離脱してしまう場合を表す．

2.3 レプリカ生成の流れ

ノード脱退もしくはノード削除処理が実行されると，その DataNode で管理されていたレプリカの再配置のために，レプリカ生成処理が行われる．NameNode がレプリカの生成元と生成先の DataNode を決定し，ブロック単位に処理が進む．NameNode が定期的にレプリカ生成指示を各 DataNode に送信する．各 DataNode はその受け取った指示をもとに，生成先の DataNode へデータの転送を行う．生成先の DataNode はデータの複製が完了すると，生成元の DataNode に ACK を返す．不足したブロックの全てが複製されるまで，このフェーズが繰り返される．

この時 NameNode が各 DataNode に指示する転送レプリカブロック数は，クラスタに接続されている DataNode 数と `REPLICATION_WORK_MULTIPLIER_PER_ITERATION` (N_work とする) の積で定義される．DataNode が生成完了の ACK を受け取らない状態のまま，レプリカ生成先の DataNode へ転送できるブロック数は `dfs.max-repl-stream`(N_stream とする) で定義される．`REPLICATION_WORK_MULTIPLIER_PER_ITERATION` は，FSNamesystem.java の ReplicationMonitor クラスで定義されている変数であり，`dfs.max-repl-stream` はプロパティで変更可能な変数である．各変数のデフォルト値は表 1 のようになっている．

表 1 レプリカ生成処理に関する変数のデフォルト値

変数	デフォルト値
N_work : NameNode が指示する転送ブロック数の一変数	2
N_stream: DataNode での同時転送ブロック数	2

3. 基本性能評価

本実験ではレプリカ生成の基本性能を把握するために，クラスタからノードを切り離す方法としてノード削除を用いてレプリカ生成処理のスループットを測定した．実験結果を解析するために，同時転送ブロック数，ディスク I/O の性能，データの偏りに関する予備調査を行う．その後その結果を踏まえて，レプリカ生成を高速化する手法を提案し，評価実験を行い，提案手法の有効性を検証していく．全ての実験において，ブロックのレプリカ数を 3 とし，1 台の DataNode をクラスタからノード削除する際のスループットを示す．なおスループットは以下のように定義した．

$$\text{スループット (MB/sec)} = \frac{\text{削除ノードが保持しているデータ量 (MB)}}{\text{レプリカ生成が完了するまでの時間 (sec)}}$$

3.1 実験環境

ローカルクラスタ上で Hadoop-1.0.3 をインストールしたマシン 7 台を用いた．そのうち 1 台を NameNode，残りの 6 台を DataNode とした．マシンのスペックは全て同一で表 2 に示す通りである．基本性能を把握するために，全ノードが Gigabit Ethernet で接続された単一のラックからなるシンプルな構成にしている．

表 2 マシンスペック

OS	Linux 2.6.32-5-amd64 Debian GNU/Linux 6.0.4
CPU	Quad-Core Intel(R) Xeon(R) CPU @ 1.60GHz
Main Memory	2GB
HDD	73GB SAS × 2(RAID0)
RAID Controller	SAS5/iR
Network	Gigabit Ethernet

3.2 同時転送ブロック数の違いによる比較

ブロックサイズ 16，32，64，128，256MB に対して，レプリカ生成時の NameNode と DataNode の同時転送ブロック数に関連する，表 1 の 2 つの変数を共に同じ値 α とし，α を 1〜8 に変化させてノード削除を行った際のスループットを図 1 に示す．

図 1 より，α が小さい時，すなわち同時転送ブロック数が少ない場合，ブロックサイズが小さい時にはスループットが低くなる．これはブロックサイズが小さいと DataNode はレプリカ生成の処理が完了して，NameNode から定期的に送られてくるレプリカ生成の指示を待っている状態が生じるからと考えられる．またこの状態にある間は，α に比例してほぼ線形にスループットが増加している．各ブロックサイズにおいて上限に到達後は，α の増加に伴い若干スループットが低下している．このスループットの若干の低下は，ストリーム数を増加させたことによって，スレッドの切り替えなどのオーバヘッドが発生し，性能低下につながったものと考えられる．

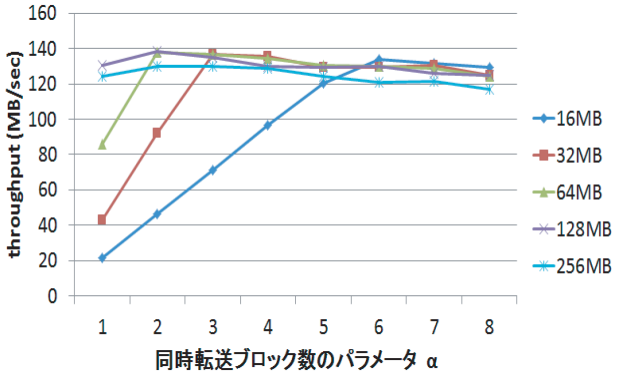

図 1 各ブロックサイズの同時転送ブロック数に応じたスループット

3.3 ディスク I/O の性能

Gigabit Ethernet を使用している本実験環境においては，ネットワークの性能面からは，各 DataNode がそれぞれ一意の DataNode を生成先として等分のデータを移行するという理想的な場合を考えると，125MB/sec × 5 台 (DataNode 数)=625MB/sec が理想的なスループット値と考えられる．しかし実際には高々 138MB/sec と理想値から程遠い．この原因としてディスク I/O，データ移動の偏り，スケジューリングなどが考えられる．そこで HDFS 上のディスク I/O の理想値を調査した．レプリカ生成時には各ノードで読み込みと書き込みが並行して行われるため，ディスクの性能調査の実験内容として，(a)30GB のファイルの書き込み時の 10GB のファイルの読み込み時間，(b)30GB のファイルの読み込み時の 10GB のファイルの書き込み時間，の 2 つの計測をマシン 1 台に対して各 12 回行いそれぞれのスループットを求めた．

(a) の実行時間は 12 回共に比較的安定しており，スループットは 67〜72MB/sec であった．一方 (b) の実行時間は試行毎に大きくばらつきがあり，スループットは 25〜52MB/sec であった．試行毎に実行時間にばらつきがあるが，ここではスループットの最大値を各ワークロードの理想値として用いる．レプリカ生成時には読み込みと書き込みが発生し低速な処理が足かせとなるため，ディスクの理想的なスループット値は 52MB/sec × 5 台 (DataNode)=260MB/sec となり，本実験環境ではネットワークの前にディスクがボトルネックになることが分かった．

3.4 データの偏り

3.3 節を踏まえて，ディスクに注目してデータ移動の偏りを調査した．ここではデフォルトの設定であるブロックサイズ 64MB，同時転送ブロック数のパラメータ α=2 の場合を取り上げる．各 DataNode 毎に iostat からディスク情報を取得し，ディスク I/O，ディスク read，ディスク write の 5 秒間の移動平均のスループットと，各 DataNode ログ (hadoop-$user-datanode.log) から送受信しているブロック数を 1 秒間隔に取得した時系列データを図 2 に示す．スループットのグラフは各 DataNode の積み上げグラフである．横軸が時間 (sec) で縦軸がスループット (MB/sec) もしくはブロック数 (個) である．送信ブロック数に関しては，各 DataNode が ACK 無しに送信できるブロック数は α 以下に制約されていて，α に満たないノードが優先的に送信元ノードに選出されるため，比較的安定しているのでここでは省略する．

図 2 より，時間が 80〜100 秒のとき，ある一つの DataNode の受信ブロック数が増加していることから，生成先が一つのノードに集中していることが分かる．この時のディスクのスループットを見てみると，read，write 共に低下している．これは，受信が集中した DataNode ではディスクの書き込みが干渉して write のスループットが低下し，その結果，そのノードにデータを送信している他の DataNode の読み出し処理も停滞してしまうので，read のスループットの低下も招く．またこれらの DataNode では受信ブロック数が減少するので，書き込み完了後は次のデータを受信するまでは，write のスループットが低下していると考えられる．時間が 140 秒と 180 秒のときも同様に，レプリカ生成先の偏りが発生し，それによりスループットが低下していることが分かる．HDFS では同一ラックに全てのノードが配置されている場合，レプリカの生成先は該当ブロックを保持しない DataNode からほぼランダムに選出されるため，場合によってはこのようなレプリカ生成先の偏りが生じてしまう．

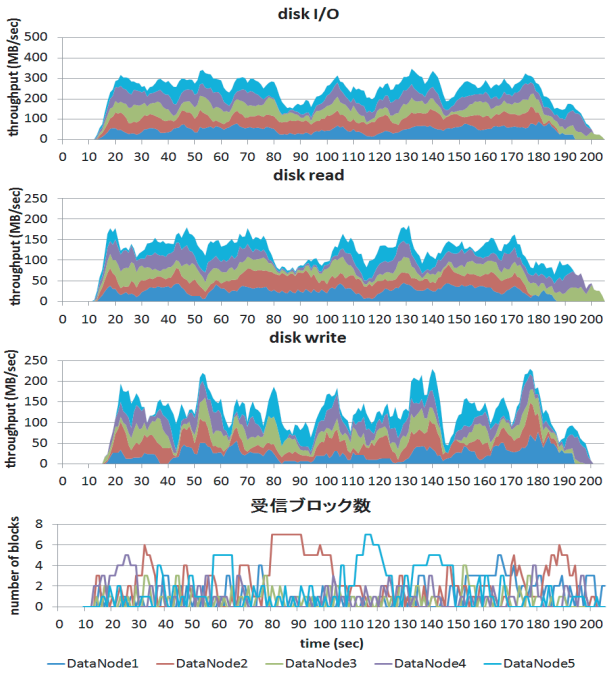

図 2　ディスクの各スループットと受信ブロック数の時系列データ

4. レプリカ生成先・生成元を考慮した制御手法の提案

前述の実験結果より，レプリカ生成時の処理にはデータ移動に偏りが生じ，それに伴いスループットが低下していることが分かった．従ってこの偏りを解消することで，レプリカ生成処理の高速化が見込める．HDFS では同一ラックに全てのノードが配置されている場合，レプリカの生成先は該当ブロックを保持しない DataNode から，そしてレプリカ生成元は該当ブロックを保持する DataNode からほぼランダムに選出される．そこで本稿では，データの偏りを解消するためにレプリカ生成先と生成元を制御する以下の改良を行った．

制御手法 1　レプリカ生成先の制御

制御手法 2　レプリカ生成先の制御＋送信ブロック数の制御 (2 つ先のノードへのデータ送信有)

制御手法 3　レプリカ生成先・生成元の制御 (2 つ先のノードへのデータ送信無)

4.1 提案手法の概要

制御手法の手順を以下に述べる．本制御手法では，ノードをリング状に配置しそのリング構造に従って一方向にデータを送信する (図 3).

(1)　レプリカ生成先は，そのリング構造に従って 1 つ先のノードに指定.

(2)　もし既に 1 つ先のノードがレプリカを保持している場合

(2a)　生成先は 2 つ先のノードに指定

(2b)　そのノードをレプリカ生成元の候補から除外

(3)　レプリカ生成元の候補の中から，累計送信ブロック数が少ないノードを生成元に選出

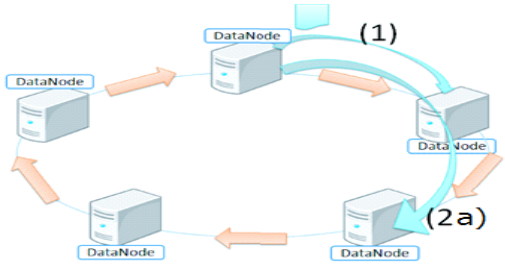

図 3　レプリカ生成先制御手法のレプリカ生成先

制御手法 1 は (1)(2a) の手順でレプリカ生成先を制御する．制御手法 2 は (1)(2a) の手順に各 DataNode の送信ブロック数を均一化する制御 (3) を加えたものである．制御手法 3 は (1) の手順に 2 つ先のノードへのデータ送信を回避する制御 (2b) を行った上で (3) の制御を加えたものである．

レプリカ生成のスケジューリングは定期的に行われるが，制御手法 2，3 の制御を加えるにあたり，最初に一括してレプリカ生成の必要な全てのブロックのスケジューリングを行うように変更した．一括してスケジューリングするため，NameNode の同時転送ブロック数に関する変数 N_work を十分大きな値に設定する．

制御手法 1 により，ある時刻にある DataNode が受信するブロック数は最大で 1 つ前の DataNode と 2 つ前の DataNode からそれぞれ N_stream 個受信する場合の $2 \times N\_stream$ 個に制約され，受信ブロックの偏りが幾分か解消されることが期待できるが，2 つ先のノードへのデータ送信が発生した場合には，データの偏りを招きかねない．制御手法 2 は，制御手法 1 に送信ブロック数を均一にする制御を加えるため，送信データ量の偏りが解消され，制御手法 1 の時よりも効率良く処理が進むと考えられる．制御手法 3 は，送信ブロック数の均一化に加え，2 つ先のノードへの送信を回避するので，結果として受信のブロック数も均一になり，最も効果的な制御手法であると考えられる．

4.2 DataNode の最適な同時転送ブロック数

3.2 節ではレプリカ生成時の NameNode と DataNode の同時転送ブロック数に関連する，表 1 の 2 つの変数，N_work，N_stream を共に同じ値 α としていたが，制御手法 2，3 では N_work を十分大きな値に設定している．そこで制御手法 3 を用いた際の DataNode の最適な同時転送ブロック数 N_stream を調査した．その結果各ブロックサイズにおける最適な同時転送ブロック数 N_stream は表 3 のようになった．

表 3　最適な同時転送ブロック数

	16MB	32MB	64MB	128MB	256MB
最適な同時転送ブロック数	8	4	2	2	1

4.3　提案制御手法の評価

ブロックサイズ 16，32，64，128，256MB に対して，4.2 節で調査した DataNode の最適な同時転送ブロック数 N_stream をそれぞれ用いて，提案した制御手法に関してノード削除を行った際のレプリカ生成のスループットを測定した．以下の 4 つのスループットを図 4 に示す．

(a)　デフォルト
(b)　制御手法 1：生成先制御手法
(c)　制御手法 2：生成先制御手法＋送信ブロック数制御 (2 つ先のノードへのデータ送信有)
(d)　制御手法 3：生成先・生成元制御時 (2 つ先のノードへのデータ送信無)

それぞれ (a)〜(d) の N_stream が等しい場合を比較する．

図 4 より，全てのブロックサイズで提案した制御手法 ((b)〜(d)) によりスループットが向上していることが分かる．デフォルト時のスループットと比較すると (b)，(c)，(d) ではそれぞれ最大 18%，26%，59%向上している．特に (d) のスループットの向上率が高く，2 つ先のノードへの送信を回避し，生成先・生成元の制御を行う制御手法 3 が非常に有効であることが分かる．

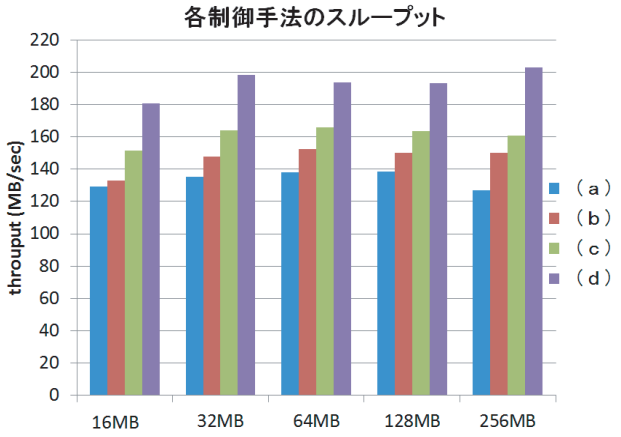

図 4　各制御手法によるスループットの比較

(a)　デフォルト
(b)　生成先制御手法
(c)　生成先制御手法＋送信ブロック数制御 (2 つ先のノードへのデータ送信有)
(d)　生成先・生成元制御手法 (2 つ先のノードへのデータ送信無)

また各 DataNode 間のデータの偏りを 3.4 節と同様にブロックサイズ 64MB，N_stream=2 の場合のディスクの各スループットと受信ブロック数の時系列データを用いて評価する．ここでは，制御手法 1 と制御手法 3 を用いた場合をそれぞれ図 5，6 に示す．

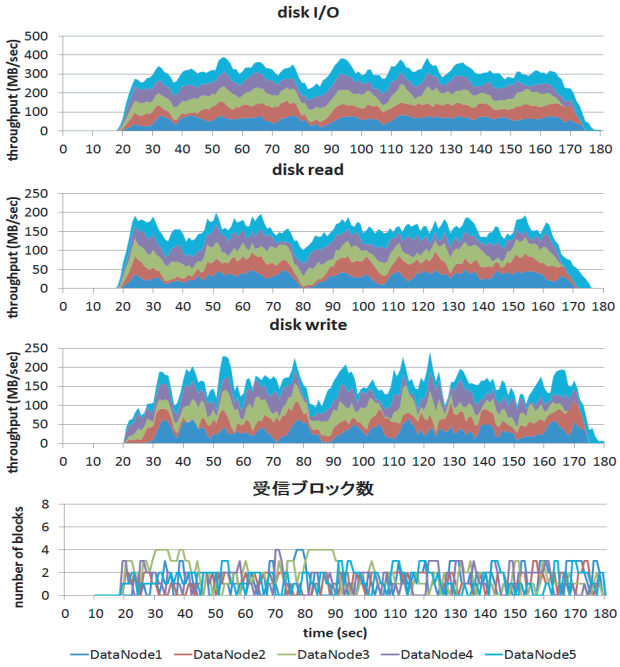

図 5　ディスクの各スループットと受信ブロック数の時系列データ (制御手法 1)

図 5 より，制御手法 1 を用いた場合，時間が 40 秒と 85 秒のとき受信ブロック数が最大の 2 × N_stream 個となる DataNode が存在しており，スループットが低下している．これは 4.1 節の制御手法の手順 (2a)，つまり 2 つ先のノードへのデータ送信が発生した場合に生じることがある．レプリカ数を 3 とし，削除ノードを除く 5 台でレプリカ生成を行っている本実験環境では，1 つ先のノードが該当ブロックを保持している可能性は理論的には 25%である．つまりデータの偏りの原因となる，2 つ先のノードへのデータ送信が 25%の高い確率で発生する．実際にある 1 回の試行の各 DataNode のレプリカ生成先の内訳を調査したところ，表 4 のようになり，ノードにより若干異なるものの，全体では理論値の 25%に一致した．このことから，2 つ先のノードへのデータ送信はスループットに大きく影響を及ぼしていると考えられる．

表 4　各 DataNode のレプリカ生成先のブロック数の内訳

	1 つ先のノードへ送信	2 つ先のノードへ送信	2 つ先のノードへ送信する確率
DataNode1	55	21	28%
DataNode2	58	18	24%
DataNode3	66	20	23%
DataNode4	61	18	23%
DataNode5	61	22	27%
合計	301	99	25%

図 6 より制御手法 3 を用いた場合，デフォルト (図 2) や制御手法 1(図 5) の時と比較すると，ディスクの全てのスループットに関して非常に安定していることが分かる．また送信先が一意なので，送信ブロック数を均一にした結果，

受信ブロック数も均一になるので各 DataNode が常に安定した等しい値を維持しており，データの偏りが解消され，効率良くレプリカ生成処理が行えることが示された．

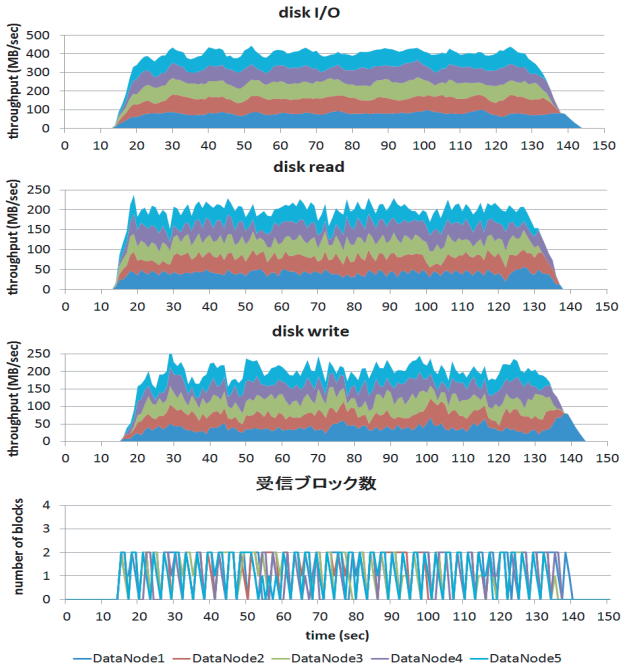

図 6 ディスクの各スループットと受信ブロック数の時系列データ (制御手法 3)

5. 関連研究

5.1 ネットワークトポロジ

Felix[5] らは大規模クラスタにおいて OS イメージを全てのマシンに高速に分配する方法として，star 型，3-ary spanning tree 型，Multi-drop chain 型の 3 つの論理ネットワークトポロジを取り上げ，調査している．star 型はノード数が増加すると，link 部分で輻輳が発生してしまい，3-ary spanning tree 型はリソースの限界により複数のストリームを効率良く処理できない．一方で，Multi-drop chain 型はノード数が増加しても処理にほとんど影響がなく，またネットワーク性能が低下しても，他のトポロジに比べて処理速度に与える影響が少ないことから，データの分配には Multi-drop chain 型が優れた方式であることが述べられている．

5.2 Power Proportional Storage

Sierra[6]，Rabbit[7]，JackRabbit[8] は，power proportionality の概念に基づき，必要な性能に応じた電力消費量の比率が一定に保たれるように，アクティブサーバ数を変更する分散ストレージシステムである．負荷に応じてストレージ規模を変更することを前提に，レプリカ配置に制御を加え，可用性に影響を及ぼすことなくサーバを素早くインアクティブ状態にすることを可能にしている．これは特定のサーバ群に primary のレプリカを配置し，ストレージを縮小する時にはそれ以外のサーバをインアクティブにすることにより，全てのレプリカに対して常に 1 つ以上はストレージに存在するようにすることで実現されている．インアクティブサーバへの書き込みが発生した場合は，代わりに一時的にアクティブサーバに書き込み，後にサーバが再びアクティブ状態になった際に，書き込みやデータの再配置などを行うことで対応している．

6. まとめと今後の課題

分散ファイルシステムの一つである HDFS 上で，ノード削除時のレプリカ生成処理を高速化する手法を提案した．デフォルトの設定ではレプリカ生成時のデータ移動に偏りが生じ，スループットの低下を招いている．そこでその偏りを解消するために，レプリカ生成先および生成先元の制御を提案し，有効性を検証した．ノードをリング状に配置しそのリング構造に従って 1 つ先のノードを生成先に選出するという生成先制御では，デフォルト時のスループットと比較すると最大 18%の向上に留まり，データの偏りは若干解消したものの，2 つ先のノードへのデータ送信が依然としてデータの偏りを招いていた．生成先制御に各 DataNode の送信ブロック数を均一化する制御を加えると，デフォルト時のスループットと比較すると最大 26%向上した．さらに最も良いと考えられる，生成先と送信ブロック数の均一化の制御に加え，2 つ先のノードへの送信を回避するとスループットが大幅に向上しデフォルトの時よりも最大 59%向上した．またデータの偏りも解消され，本提案手法が非常に有効であることが示された．

本実験環境では生成元の候補となる DataNode は 2 つであるが，2 つ先のノードへのデータ送信を回避した場合，生成元となる DataNode は一意に決まってしまう．現在の制御はレプリカ生成の必要なブロックを無作為に順々にスケジューリングしていっている．そこでまず始めに 2 つ先のノードへの送信を回避しなければならないブロックを先にスケジューリングして，その後生成元として 2 つの候補の中から選択できるブロックをスケジューリングすることで，スケジューリングに幅を持たせられるように制御を加えていこうと考えている．

またレプリカ生成処理のスケジューリングを 0-1 整数計画法を用いてモデル化することにも取り組んでいきたい．スケジューリングの最適解を得るのに要する時間や現在の制御がどの程度最適解に近いものかなどを調査し，最終的には，求めた最適解をレプリカ生成処理のスケジューリングに組み込めるように実装していきたい．

参考文献

[1] Dhruba Borthakur, ”HDFS Architecture,” 2008 The Apache Software Foundation.

[2] Tom White(著), 玉川竜司，兼田聖士 (訳):Hadoop, オライリー・ジャパン,2010

[3] Ghemawat, Sanjay; Gobioff, Howard; Leung, Shun-Tak (October 2003), ”The Google File System,” 19th Symposium on Operating Systems Principles (conference), Lake George, NY: The Association for Computing Machinery, CiteSeerX: 10.1.1.125.789, retrieved 2012-07-12.

[4] 日開朝美, 竹房あつ子, 中田秀基, 小口正人”Hadoop のノード脱退時および削除時のレプリカ生成の高速化” SACSIS 2013，2013 年 5 月.

[5] Felix Rauch, Christian Kurmann, Tomas M.Stricker, ”Partition Cast — Modelling and Optimizing the Distribution of Large Data Sets in PC Clusters,” Euro-Par 2000, LNCS 1900, pp.1118-1131, 2000.

[6] Eno Thereska, Austin Donnelly, and Dushyanth Narayanan. ”Sierra: Practical Power-proportionality for Data Center Storage.” In Proceedings of the Sixth Conference on Computer Systems, EuroSys’11, pp. 169-182, New York, NY, USA, 2011. ACM.

[7] Hrishikesh Amur, James Cipar, Varun Gupta, Gregory R.Ganger, Michael A. Kozuch, and Karsten Schwan. ”Robust and Flexible Power-Proportional Storage.” In Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC’10, pp. 217-228, New York, NY, USA, 2010. ACM.

[8] James Cipar, Lianghong Xu, Elie Krevat, Alexey Tumanov Nitin Gupta, Michael A. Kozuch, Gregory R. Ganger. ”JackRabbit: Improved Agility In Elastic Distributed Storage.” Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-12-112, October 2012.