大規模環境におけるHDFSの効率的なレプリカ再配置制御に向けた評価

日開　朝美†　　竹房あつ子††　　中田　秀基††　　小口　正人†

† お茶の水女子大学　〒112-8610 東京都文京区大塚 2-1-1
†† 産業技術総合研究所　〒305-8568　茨城県つくば市梅園 1-1-1
E-mail: †asami@ogl.is.ocha.ac.jp, ††{atsuko.takefusa,hide-nakada}@aist.go.jp, †††oguchi@computer.org

あらまし　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．分散ファイルシステムでは従来よりも大規模なシステムが構成されるようになり，複数のレプリカを分散配置することで可用性や耐故障性を維持している．ノードが故障すると，そのノードが管理していたレプリカが一時的に不足し，そのデータを保持している他のノードへのアクセス負荷が増加してシステム全体の性能が低下するため，不足レプリカの再配置の高速化が重要である．しかしながら，一般に広く利用されている Hadoop Distributed File System(HDFS) のレプリカ再配置では，データ移動に偏りが生じており，効率良く処理されていない．この問題を解消するために，我々はリング構造に基づく一方向のデータ転送を行い負荷分散を行う制御手法を提案した．本稿では，この制御手法を 0-1 整数計画問題として定式化した最適化手法を提案し，7 台からなる実クラスタを用いた小規模環境での評価とシミュレーションによる大規模環境での評価を行った．評価実験より，いずれの環境においても提案手法が有効であり，小規模実環境ではスループットが最大で 45%向上することを確認した．
キーワード　分散ファイルシステム，HDFS，レプリカ再配置，ノード削除

A Evaluation of Effective Replica Reconstruction Schemes on a Large-scale HDFS Environment

Asami HIGAI†, Atsuko TAKEFUSA††, Hidemoto NAKADA††, and Masato OGUCHI†

† Ochanomizu University　2-1-1 Otsuka, Bunkyouku Tokyo 112-8610 JAPAN
†† National Institute of Advanced Industrial Science and Technology (AIST)　1-1-1 Umezono, Tsukuba, Ibaraki, 305-8568, Japan
E-mail: †asami@ogl.is.ocha.ac.jp, ††{atsuko.takefusa,hide-nakada}@aist.go.jp, †††oguchi@computer.org

1. はじめに

近年，センサネットワークやソーシャルメディアなどから大量のデータが生み出されるようになり，高エネルギー物理学，生命情報工学などの科学技術分野や商業分野において，大規模データを効率良く管理，処理することが求められている．このような大規模データに対応した処理システムとして，汎用的なハードウェアを用いて高度な集約処理を可能にする分散ファイルシステムが広く利用されている．分散ファイルシステムは，データに対して複数のレプリカを生成し，大量のノードを用いて分散管理することで可用性や耐故障性を維持している．ノードが故障すると，そのノードが管理していたレプリカが一時的に不足し，そのデータを保持している他のノードへのアクセス負荷が増加して，システム全体の性能が低下する．そのため不足レプリカの再配置を高速に行い，データ処理システム全体の性能低下を防ぐことが重要である．

オープンソースの分散ファイルシステムでは，Apache Hadoop [2](以下 Hadoop) プロジェクトの Hadoop Distributed File System [3](以下 HDFS) が広く用いられている．しかしながら，HDFS のレプリカ再配置では，レプリカの生成元と生成先がランダムに選ばれるため，データ移動に偏りが生じて，効率良く処理が行われていない [1]．我々はこの問題を解消するために，リング構造に基づく一方向のデータ転送によって負荷分散を行うレプリカ再配置のスケジューリング制御手法を提案し，その有用性を示した [1]．

本稿では，これまで提案してきたレプリカ再配置のスケジュー

リング制御を，0-1 整数計画問題として定式化した最適化手法を提案し，マシン 7 台からなる実クラスタを用いた小規模環境と，シミュレーションによる大規模環境における提案手法の有効性を評価する．評価実験より，小規模実環境において提案手法により最大で 45%スループットが向上することと，ヒューリスティック手法が最適化手法と同程度の性能を得られることを示す．また大規模シミュレーション環境においてノード数が増加した場合にも提案手法が有効であることを示す．

2. 関連研究

2.1 レプリケーション手法

分散ファイルシステムにおいて，データのレプリカをどのように管理するかというレプリケーション手法が数多く提案されている．一般にデータを複製して，複数のレプリカを複数の異なるマシン上に保存することで，高いシステム性能や耐故障性，信頼性を提供しているが，この時，レプリカ数とレプリカの配置場所が重要になってくる．

ファイルクラスタリングに基づいて，アクセス時間やストレージ容量などの性能要件を満たし，レプリカの複製時間が最小になるようにレプリカの配置を決定する手法 [4] [5] [6] や，ファイルの人気度に基づいて，レプリカ配置場所と複製数を決定する手法 [7] など，アクセス時間やストレージ容量，複製時間を考慮したレプリカ配置手法は数多く提案されているものの，複製処理自体が各ノードに与える負荷についてはあまり考慮されていない．

2.2 ネットワークトポロジ

Felix [8] 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，n-ary spanning tree 型，multi-drop-chain 型の 3 つの論理ネットワークトポロジ (図 1) を取り上げ，調査している．評価実験から，star 型はノード数が増加すると，スイッチ部分で輻輳が発生してしまう．n-ary spanning tree 型はネットワーク帯域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，他のトポロジと比較するとノード数やネットワーク帯域に大きく影響されることなく，データの転送が可能であり，データの分配に適した方式であることが述べられている．

我々は，レプリカ再配置において multi-drop-chain 型と同様の転送処理が行われるように，一方向のリング構造をベースとしたスケジューリングを行う．

3. HDFS におけるレプリカ再配置

HDFS は Google の分散ファイルシステム GFS [9] に基づいて設計された分散ファイルシステムである．HDFS はマスタ・ワーカの構成をしており，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を行う複数台の 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 間で各ブロックのレプリカを複数個分散して保存することで可用性や耐故障性を維持している．

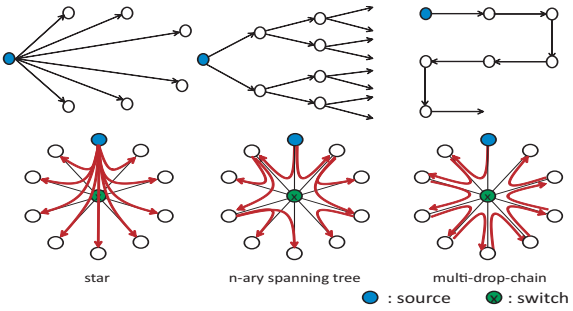

図 1 各ネットワークトポロジ

表 1 レプリカ再配置処理に関する変数のデフォルト値

変数	説明	デフォルト値
N_work	NameNode の指示ブロック数の一変数	2
N_stream	DataNode の同時転送ブロック数	2

3.1 レプリカ再配置処理

クラスタから DataNode が離脱すると，その DataNode で管理されていたデータの不足分を補うレプリカ再配置処理が行われる．NameNode がレプリカ生成元と生成先の DataNode を決定し，ブロック単位で処理される．NameNode が定期的にレプリカ再配置の指示を生成元の DataNode に送信し，各 DataNode は受け取った指示をもとに，生成先の DataNode へブロックの転送を行う．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 を返し，さらにその生成元の DataNode は NameNode に複製が完了したことを伝える．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 が生成元の DataNode に指示するブロック数は，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．この値を超えてレプリカ再配置のスケジューリングすることは出来ず，スケジューリングとデータ転送が並行して行われる．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．

4. レプリカ再配置の制御手法の提案

レプリカ再配置処理を効率良く行うには，適切にレプリカの生成元と生成先を選択して各 DataNode の送受信処理を均衡化することが必要である．そこで我々は，生成元が決定すると生成先も一意に決定する一方向のリング構造に基づいてデータ転送を行いながら，各 DataNode の転送データ量を均衡化するようなレプリカ再配置のスケジューリング方針を提案している [1]．この方針をもとに，0-1 整数計画問題として定式化して最適化ソルバで求めた最適解を用いる手法 (以下，最適化手法) を提案する．

4.1 提案するスケジューリング方針の概要

提案するレプリカ再配置のスケジューリング方針を以下に述べる．全ノードが同一ラック上に存在するものとする．

1) DataNode を論理的にリング状に配置し，そのリング構造に従って一方向にデータを転送する．

2) 各 DataNode が送信するデータ量を等しくするために，生成元に選出される回数を等しくする．

4.2 最適化手法

前述で提案したスケジューリング方針を 0-1 整数計画問題として定式化し，その解をレプリカ再配置のスケジューリングに用いる．定義する 0-1 整数計画問題では，レプリカ再配置に要する時間を短縮するために，各 DataNode の送信ブロック数の差を最小化することを目指す．

まず，記号の定義を与える．DataNode i の集合を D，レプリカ再配置が必要なブロック j の集合を B とする．DataNode の総数を N_{dn}，レプリカ再配置が必要なブロックの総数を N_b，レプリカ数を $N_{replica}(\geqq 2)$ とすると，DataNode あたりの平均転送ブロック数 N_{avg} は，$N_{avg} = N_b/N_{dn}$ となる．現在のブロックの配置を行列 $Current_{i,j}(i \in D,\ j \in B)$ とする．$Current_{i,j}$ の値は，DataNode i にブロック j が存在する場合は 1，存在しない場合は 0 とする．また DataNode の隣接関係を行列 $Adj_{from,to}(from,\ to \in D)$ とする．$Adj_{from,to}$ の値は DataNode $from$ から DataNode to への転送が可能な場合は 1，そうでない場合は 0 とする．レプリカ再配置のスケジューリング結果を格納する変数を $X_{from,to,j}$ で表す．$X_{from,to,j}$ の値は，DataNode $from$ から DataNode to へブロック j の転送をする場合は 1，ない場合は 0 とする．各 DataNode i の転送ブロック数の差を最小化するために利用する変数を z_i とする．この時，レプリカ再配置のスケジューリングは以下のように定式化される．

$$Minimize \quad \sum_{i \in D} z_i \tag{1}$$

$$Subject\ to \quad All_{i,j} = Current_{i,j} + \sum_{from \in D} X_{from,i,j} \quad \forall i \in D, \forall j \in B \tag{2}$$

$$All_{i,j} \leqq 1,\ \forall i \in D,\ \forall j \in B \tag{3}$$

$$\sum_{i \in D} All_{i,j} = N_{replica},\ \forall j \in B \tag{4}$$

$$X_{from,to,j} \in \{0,1\},\ \forall from, \forall to \in D, \forall j \in B \tag{5}$$

$$Current_{i,j} - \sum_{to \in D} X_{i,to,j} >= 0,\quad \forall i \in D, \forall j \in B \tag{6}$$

$$\sum_{j \in B} X_{from,to,j} <= M \cdot Adj_{from,to} \quad \forall from, \forall to \in D \tag{7}$$

$$\sum_{j \in B} X_{from,to,j} - N_{avg} \geqq -z_i,\quad \forall from, \forall to \in D \tag{8}$$

$$\sum_{j \in B} X_{from,to,j} - N_{avg} \leqq z_i,\quad \forall from, \forall to \in D \tag{9}$$

$$z_i \geqq 0,\ \forall i \in D \tag{10}$$

上記を満たす，$X_{from,to,j}$ を求めレプリカ再配置のスケジューリングに用いる．式 (1) は，各 DataNode の転送ブロック数の差を最小化するための目的関数を表す．式 (2) は，転送後の配置を $All_{i,j}$ で表す．式 (3) は，転送後の配置において同じ DataNode に同じブロックが 2 つ以上配置されないことを表す．式 (4) は，各ブロックのレプリカの総数が $N_{replica}$ になることを表す．式 (5) は，$X_{from,to,j}$ は 0 か 1 の値をとることを表す．式 (6) は，ブロックの生成元となる DataNode がそのブロックを持っていることを表す．式 (7) は，DataNode $from$ と DataNode to が転送リングにおける生成元 DataNode と生成先 DataNode の関係にあることを表す．隣接関係がない DataNode 間の転送ブロック数は 0 で，ある場合は正の値となる．M はある程度大きい値であり，ブロック総数を超えることはないので，ここでは $M = N_b$ とする．式 (8)(9) は，各 DataNode が転送するブロック数とその平均値 N_{avg} の差の下界と上界を表す．式 (10) は，z_i は 0 以上の値をとることを表す．

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

表 3 実験に用いた各パラメータ

レプリカ数	3
HDFS のデータ量	50GB×3(レプリカ数)
ブロックサイズ	16,32,64,128,256MB
DataNode 数	6 台から 1 台削除

5. 実クラスタを用いた小規模環境における評価

最適化手法を HDFS のレプリカ再配置モジュールに実装した．最適化ソルバには無償で提供されている GLPK [10] を用いる．デフォルトの手法と最適化手法，そして [1] で既に提案している制御手法 (以下，ヒューリスティック手法) を用いて，ノード削除時のレプリカ再配置処理の性能を比較する．評価では，以下を調査する．

(1) 再配置のスループット

(2) 各 DataNode の転送ブロック数

(2) の各 DataNode が転送するブロック数の調査では，送受信処理の偏りを評価する．

5.1 実験概要

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 台からなるクラスタを用いた．その内 1 台を NameNode とし，残りの 6 台を DataNode とする．マシンのスペックは全て同一で表 2 に示す．全ノードが Gigabit Ethernet で接続された単一のラックからなる．

実験に用いたパラメータを表 3 に示す．レプリカ数を 3 とし，各試行毎に HDFS に約 10GB のファイルを 5 つ put した．レプリカを含めた全データ量は約 150GB となっており，これはクラスタ全体の容量の約 25%に相当する．ファイルを put した後に，バランサーを起動し，各 DataNode が保持するデータ量を均一にした．そのため試行毎にデータ配置は異なるが，各 DataNode が保持するデータ量は同程度である．

各手法を用いて，ブロックサイズを 16〜256MB(デフォルト 64MB) まで変化させ，1 台の DataNode 削除時のレプリカ再配置処理の性能を調査する．

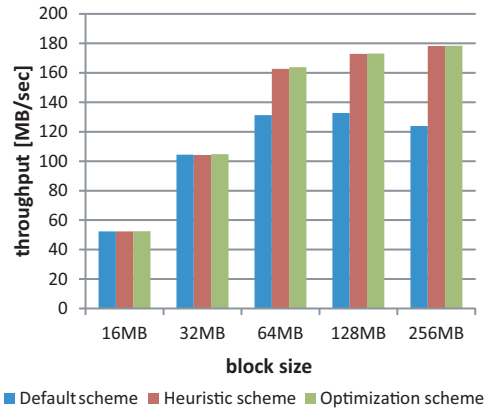

図 2　各手法におけるノード削除時のスループット

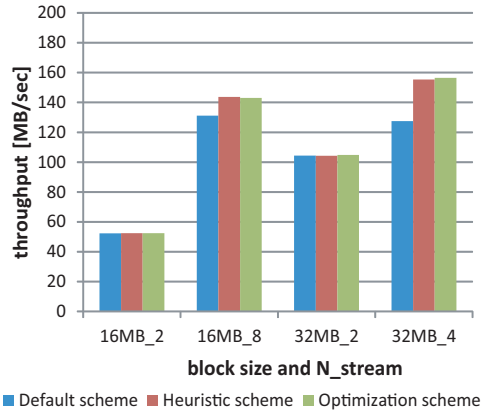

図 3　N_stream を変化させた時のノード削除時のスループット (16MB,32MB)

表 4　16MB,32MB に対する N_stream の値

ブロックサイズ	N_stream
16MB	8
32MB	4

5.2 実験結果

5.2.1 再配置のスループット

各手法を用いて，ノード削除を行った時のスループットを図 2 に示す．縦軸は再配置のスループット [MB/sec] で，横軸はブロックサイズである．図 2 より，ブロックサイズが 64MB 以上の場合は提案手法によりスループットが向上しており，デフォルトの手法と比較するとヒューリスティック手法では最大で 44%，最適化手法では最大で 45%向上していた．ヒューリスティック手法と最適化手法の再配置のスループットはほぼ等しく，本実験環境においてヒューリスティック手法が十分有効であることが確認できた．

ブロックサイズ 16MB，32MB と小さい場合は，制御前後でスループットに変化がない．これは，ディスク帯域を全て使い切っておらず，処理に余力があるためである．3.1 で述べたように，各 DataNode が ACK を受け取らない状態のまま転送出来るブロック数がデータ量ではなくブロック数 N_stream で指定されているために，ブロックサイズが小さいと処理するデータ量も少なくなる．よって，DataNode では NameNode から指示されたレプリカ再配置処理が完了して，定期的に送られてくるレプリカ再配置の指示を待っている状態がデフォルト手法時から既に生じていたからと考えられる．そこでブロックサイズ 16MB，32MB に対して，処理の負荷が大きい場合には提案手法が有効であるかを確認するために N_stream の値を表 4 に示す値に変化させて，ノード削除の実験を行った際の再配置のスループットを図 3 に示す．縦軸は再配置のスループット [MB/sec] で，横軸はブロックサイズと N_stream の値である．図 3 より，N_stream の値が大きい場合にはブロックサイズが小さくても制御手法によりスループットが向上することが確認できた．

またデフォルト手法とヒューリスティック手法を用いた際のディスク I/O のスループットの時系列データ (ブロックサイズ 64MB) を図 4 に示す．図 4 は縦軸がディスク I/O のスループット [MB/sec] で，横軸が時間 [sec] である．図 4 より，デフォルト手法では，ディスク I/O のスループットが不安定であるのに対し，ヒューリスティック手法では，比較的安定した高い値が全 DataNode で維持されている

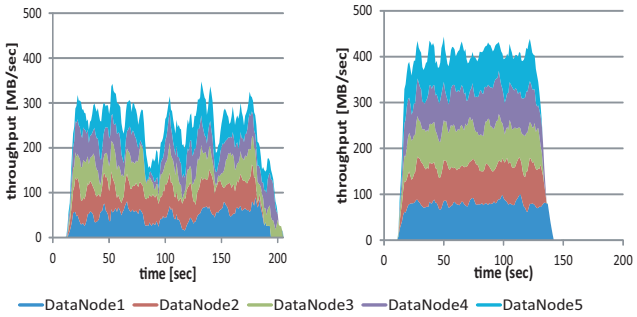

図 4　各 DataNode(計 5 台) のディスク I/O のスループット (積み上げグラフ)
(左：デフォルト手法，右：ヒューリスティック手法)

5.2.2 各 DataNode の転送ブロック数

デフォルト手法とヒューリスティック手法を用いた際のディスク I/O のスループットと受信ブロック数の時系列データ (ブロックサイズ 64MB) を図 5 に示す．図 5 は縦軸が受信ブロック数で，横軸が時間 [sec] である．図 5 より，デフォルト手法では各 DataNode が受信しているブロック数には大きく差があるのに対し，ヒューリスティック手法では各 DataNode の受信ブロック数が安定している．

また各手法を用いて，ノード削除を行った時の各 DataNode が受信したブロック数とその偏差を表 5 に示す．ここではブロックサイズ 64MB のある 1 回の試行を取り上げている．データ配置が試行毎に異なるため，受信ブロック数や偏差の値は各試行で若干異なり，各試行の偏差を平均すると表 6 になる．提案手法により各 DataNode が受信するブロック数が均衡化され，送受信処理の偏りが解消されていることが分かる．ヒューリスティック手法と最適化手法の受信ブロック数の結果はほぼ同じで，送受信処理の偏りという点からも，ヒューリスティック手法が有効であることが分かる．

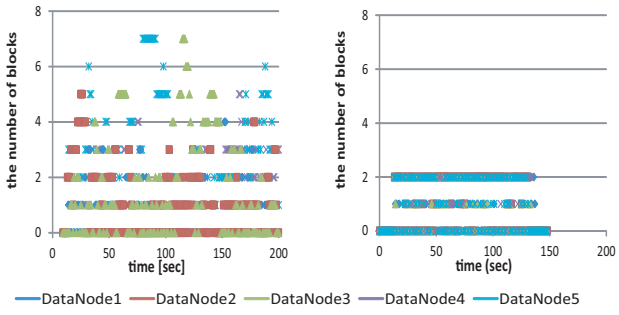

図 5　各 DataNode の受信ブロック数
(左：デフォルト手法，右：ヒューリスティック手法)

表 5　ある 1 回の試行における各 DataNode の受信ブロック数とその偏差

	デフォルト手法	ヒューリスティック手法	最適化手法
DataNode1	78	78	78
DataNode2	79	79	79
DataNode3	84	78	79
DataNode4	85	79	79
DataNode5	67	79	78
偏差	7.162	0.548	0.548

表 6　各試行における各 DataNode の受信ブロック数の偏差の平均

	デフォルト手法	ヒューリスティック手法	最適化手法
平均偏差	8.337	0.697	0.481

表 7　ノード数とブロック数のパラメータ

	ノード数	ブロック数
ブロック数：固定 ノード数：変化	5〜25	800
ブロック数：変化 ノード数：固定	10	800〜4000

6. シミュレーションによる大規模環境における評価

HDFS の実際の運用では，数十〜数千ノードを用いてラックを構成して運用される．そこで大規模環境における提案手法の有効性を検証するために，シミュレーションにより以下を調査する．

(1)　最適化手法の実用性
(2)　レプリカ再配置の実行時間

6.1　最適化手法の実用性

一般に最適解の求解には時間がかかり実システムでは非実用的であると言われているため，ノード数やブロック数を変化させて最適化手法の実用性を評価する．

6.1.1　実験概要

ノード数とブロック数を変化させて，最適解の求解にかかる時間をシミュレーションを用いて評価する．実験に用いたパラメータを表 7 に示す．ブロック数 800 個は，ブロックサイズを 64MB とした時の表 3 のデータ量に相当する．

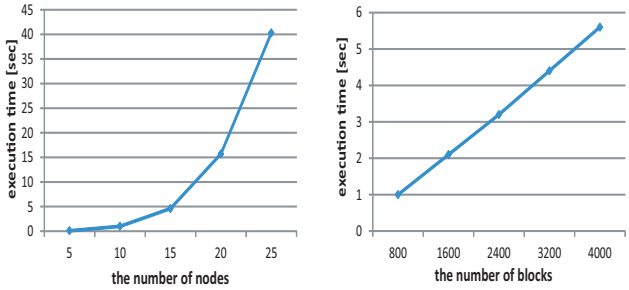

図 6　左：最適解の求解時間 (ブロック数固定，ノード数変化)
図 7　右：最適解の求解時間 (ブロック数変化，ノード数固定)

6.1.2　実験結果

最適解の求解に要した時間を図 6，7 に示す．図 6 はブロック数固定でノード数を変化させた時の結果で，図 7 はノード数固定でブロック数を変化させた時の結果である．図 6，7 より，求解時間はノード数の増加に伴い指数的に，そしてブロック数の増加に伴い線形に増加していることが分かる．ノード数を d，ブロック数を b とすると，最適化手法の計算量は，$O(d^2 \times b)$ である．一方デフォルト手法とヒューリスティック手法の計算量は，$O(b)$ である．最適化手法がスケールしない一方，ヒューリスティック手法は $O(b)$ の計算量で最適化手法と同等の再配置スループットを達成することが可能であり，大規模環境においても非常に有効であることが示唆された．

6.2　レプリカ再配置の実行時間

ヒューリスティック手法が最適化手法と同程度の性能を示すことと，大規模環境において最適化手法を用いることが非実用的であることから，大規模環境の評価では，提案手法としてヒューリスティック手法を用いる．デフォルト手法とヒューリスティック手法に対して大規模環境におけるレプリカ再配置の基本性能を把握するために，単一ラックにおいてノード数を変化させた場合のレプリカ再配置の実行時間をシミュレーションを用いて評価する．

6.2.1　実験概要

DataNode 数を 5〜30 台まで変化させて，レプリカ再配置の実行時間をシミュレーション環境を用いて評価する．評価には，分散システムのシミュレータ SimGrid [11] を用いた．また基本性能を把握するために単一のラック構成とする．評価に用いるパラメータを表 8 に示す．DataNode 数はノード削除後の DataNode 数を示す．ブロック数は，5 節の実験においてレプリカ再配置時に各 DataNode が転送したデータ量に近い値として，DataNode 数に比例する値とした．ブロック一つあたりの再配置にかかる時間は実環境の実験結果から 1.5sec とし，複数の複製処理が共存する場合には CPU リソースは等しく割り当てられるものとする．

6.2.2　実験結果

デフォルト手法とヒューリスティック手法を用いて，ノード数を変化させた際のレプリカ再配置の実行時間を図 8 に示す．図 8 は縦軸が実行時間 [sec] で，横軸が DataNode 数である．ひげは 95%信頼区間を表している．図 8 より，DataNode 数が増

表 8　シミュレーションの各種パラメータ

DataNode 数	5, 10, 15, 20, 25, 30 台
ネットワーク帯域幅	1Gbps
ネットワーク遅延	0.05msec
CPU 性能	4.2GFLOPS
ブロックサイズ	64MB
ブロック数	80×DataNode 数

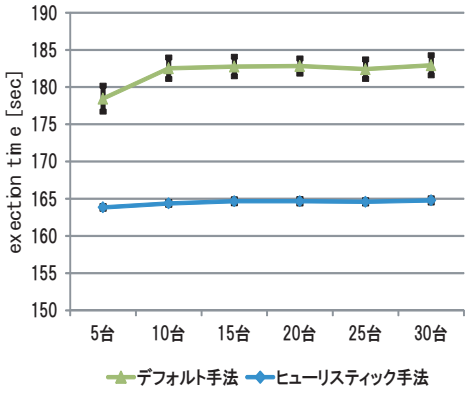

図 8　レプリカ再配置の実行時間

加した場合にも提案手法の方が再配置に要する時間が短く，有効であることが分かる．DataNode 数に関わらず実行時間はほぼ一定で，両手法ともスケールしていることが分かる．また信頼区間に関して，デフォルト手法では若干の幅があるのに対して，ヒューリスティック手法では非常に小さいことから，データ配置に関わらず安定して効率良く処理することができると考えられる．以上より単一ラックにおいては，DataNode 数の増加に関わらず，ヒューリスティック手法が有効であることが分かった．

7. ま と め

HDFS において効率良くレプリカ再配置を行うために，リング構造に基づく一方向のデータ転送を行い，各ノードの負荷を均衡化するスケジューリング方針をもとにした最適化手法を実装し，ヒューリスティック手法とともに単一ラック環境において評価した．評価実験から実クラスタを用いた小規模環境においては，提案手法により各 DataNode の負荷が均衡化し，スループットが最大で 45%向上することと，ヒューリスティック手法が最適化手法と同程度の性能が得られることを示した．シミュレーションによる大規模環境においては，最適化手法は DataNode 数が増加すると非実用的であるが，ヒューリスティック手法は DataNode 数やデータ配置に関わらず有効であることが分かった．

今後の課題は，複数のラックが存在する場合において，各手法の基本性能や有効性を検証することである．

文　献

[1] 日開朝美，竹房あつ子，中田秀基，小口正人，”Hadoop のノード削除時のレプリカ生成の高速化手法の提案” Multime-dia, Distributed, Cooperative, and Mobile Symposium(DICOMO2013), 7H-2, July 2013.

[2] Tom White, Hadoop: The definitive guide, trans. Ryuji Tamagawa. O'Reilly JAPAN, 2010.

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

[4] Rashedur M.Rahman, Ken Barker, Reda Alhajj, ”Study of Different Replica Placement and Maintenance Strategies in Data Grid,” In Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, pp.171-178, 2007.

[5] Y. Wang and D. Kaeli, ”Load balancing using grid-based peer-to-peer parallel I/O,” In Proceedings of IEEE International Conference on Cluster Computing, pp.1-10, 2005.

[6] Hitoshi Sato, Satoshi Matsuoka, and Toshio Endo, ”File Clustering Based Replication Algorithm in a Grid Environment,” In Proceedings of the 9th IEEE International Symposium on Computing and the Grid (CCGrid2009), pp.204-211, Shanghai, China, May 2009.

[7] K. Sashi, Antony Selvadoss Thanamani, ”A New Replica Creation and Placement Algorithm for Data Grid Environment,” International Conference on Data Storage and Data Engineering, pp. 265-269, 2010.

[8] 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.

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung (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.

[10] GLPK. http://www.gnu.org/software/glpk/

[11] SimGrid. http://simgrid.gforge.inria.fr/