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

日開朝美[†] 竹房あつ子[††]
中田秀基[††] 小口正人[†]

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，ノードの脱退もしくは削除が想定される．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，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に関するスループットを評価する．予備実験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を制御することでその偏りを解消し，処理の高速化を目指す制御手法を提案する．制御手法によりスループットが最大で 18%向上すること，およびデータ移動の偏りが若干解消することを示す．

Improvement of Replica Generation at Node Decommission and Deletion for Hadoop

Asami Higai,[†] Atsuko Takefusa,[††] Hidemoto Nakada[††]
and Masato Oguchi [†]

As a processing system corresponding to Big Data, Distributed File System, on which intensive transaction is executed with commodity machines, is drawing attention. In this study, we focus on Hadoop Distributed File System(HDFS), which is a foundational system of Apache Hadoop. Hadoop is a scale-out system and we assume node decommission or deletion occurs by the management depending on the system scale or by some troubles. In general, HDFS holds some replicas on several DataNode and replicates to the other node automatically if node decommission or deletion occurs. If this process takes longer, some DataNodes are overloaded and total performance decreases, so it is important to improve it. In this paper, we focus on node deletion and evaluate the throughput about the replica generation, which makes up for the deficit. From some preliminary experiments, we found the data transfer process is unfair and inefficient. We propose a method to get rid of the deflection by controlling the selection of the target node of replica generation and improve the process. As a result, we show that we can improve the throughput by 18% at most and get rid of the deflection to some extent.

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構成を取るようになった．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．

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

本稿では，HDFS において特にノード削除に注目してレプリカ生成時の性能評価を行った．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．そこでその偏りを解消するために，レプリカ生成先を制御する手法を提案し，実装した．評価実験から，提案手法によりスループットが最大で 18%向上し，ノード間のデータ移動の偏りも若干解消することができた．

2. ノード脱退・削除時の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．このとき，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 に転送する．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*`REPLICATION_WORK_MULTIPLIER_PER_ITERATION` 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` 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 各変数のデフォルト値

変数	デフォルト値
REPLICATION_WORK_MULTIPLIER_PER_ITERATION	2
dfs.max-repl-stream	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，ブロックサイズおよび同時転送ブロック数がスループットに与える影響を調査する．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．レプリカ生成処理では，生成元ではデータの読み込み，生成先ではデータの書き込みが行われることから，スループットは以下のように定義した．

$$\text{スループット (MB/sec)} = \frac{\text{削除ノードが保持しているデータ量 (MB)}}{\text{データの複製が開始してから完了するまでの時間 (sec)}}$$

加えてその結果を解析するために，DataNode の最適な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 3 つの予備実験を行う．

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

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 の同時処理するブロック数に関連する，`REPLICATION_WORK_MULTIPLIER_PER_ITERATION` と `dfs.max-repl-stream` 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 が小さいとき，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．本実験のスループットの上限値は約 138MB/sec であり，上限値に到達後は 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．NameNode の指示転送ブロック数は今回の実験と同様に 1〜8 に変化させる一方で，DataNode の転送ブロック数を固定させてレプリカ生成処理を行った他の予備実験からは，ブロックサイズが 64MB の時には，DataNode の転送ブロック数が 2 の時が最もスループットが高いことが分かっている．

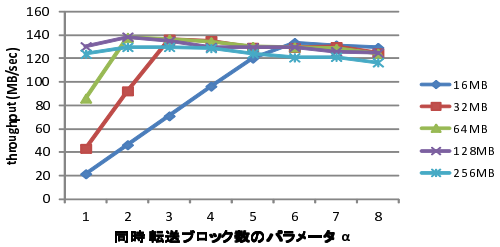

図 1　各ブロックサイズにおける
同時処理するブロック数に応じたスループット

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させるという理想的な場合を考えると，125MB/sec × (DataNode 数) 即ち，125MB/sec × 5 台=625MB/s 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 と理想値からは程遠い．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，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は 52MB/sec × 5(DataNode 数)=260MB/sec となり，本実験環境ではネットワークではなく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 の 5 秒間の移動平均と，Log から送受信しているブロック数を 1 秒間隔に取得したものを図 2 に示す．図 2 は上から順に各ノードの，ディスク read のスループット，ディスク write のスループット，受信ブロック数である．スループットのグラフは全て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 以下に制約されており，ACK を受け取るとすぐに NameNode から受け取ってあった転送指示のブロックの転送を始めるため，比較的 α に落ち着いているので，ここでは省略する．

図 2 より，時間 80〜100 秒ではある一つの 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 の読み出し処理が停滞してしまうので，read のスループットの低下も招いている．またこれらの DataNode では，受信ブロック数が減少しているので，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．120，140 秒でも同様に，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偏りが発生してしまう．以上より，レプリカ生成先の偏りは，スループットの低下を招くので，この生成先の偏りを解消することで，各 DataNode のスループットが安定し，高いスループットが維持できると考えられる．

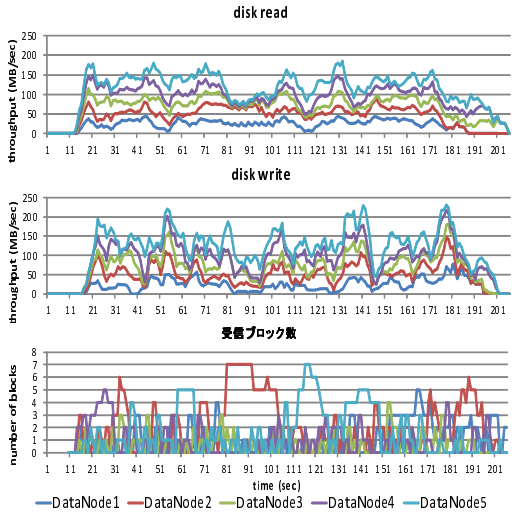

図 2　ディスクの各スループットと受信ブロック数の時系列データ

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特に生成先に偏りが発生し，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．従ってレプリカ生成先の偏りを解消することで，処理の高速化が見込める．そこで，レプリカ生成処理の高速化に向けて，まずはレプリカ生成先を制御する手法を提案し，制御前後でスループット，および各 DataNode の時系列データを比較して有効性を検証する．

4.1 提 案 手 法

ノードがリング状に配置されていると仮定し，レプリカ生成先を以下のように定義する (図 3).

(1) レプリカ生成先をリング状の次のノードに指定
(2) もしそのノードがすでにレプリカを保持している場合は，さらに次のノードをレプリカ生成先に指定

このようにレプリカ生成先を制御することで，データの流れを一方向に制御する．また各 DataNode がある時刻に受信するブロック数は，α=2 の時を例とすると，最大でも 1 つ前の DataNode からの 2 つと 2 つ前の DataNode からの 2 つの計 4 つに制限され，ある程度のデータの偏りの解消を期待できる．

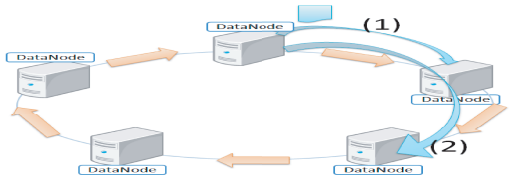

図 3　提案手法のレプリカ生成先

4.2 提案手法の評価

3.2 節と同様の実験を提案手法に関して行い，ブロックサイズ毎の制御前後のスループットの測定結果を図 4 に示す．128MB の結果は 256MB の結果とよく似ているのでここでは割愛する．図 4 より，DataNode が最大ストリーム数以上のレプリカ生成の指示を NameNode から受け取っている時には，全ての場合で提案手法のスループットが向上しており，最高で 18%向上している．16，32，64MB においてαが小さい時には，制御前後のスループットに変化がないが，これは 3.2 節で述べたように DataNode は NameNode からのレプリカ生成の指示を待っている状態であると考えられることから，妥当な結果である．また提案手法では，ブロックサイズが大きいほどスループットが向上している．ブロックサイズが大きい方が 1 つのブロックのレプリカ生成にかかる時間が長く，複数ノード間での通信効率を向上させる提案手法の効果が大きいためである．

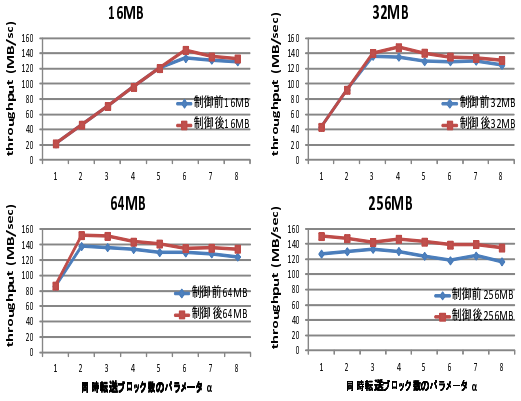

図 4　ブロックサイズ毎の制御前後のスループット

また各 DataNode 間のデータの偏りを評価するために，3.4 節と同様にブロックサイズ 64MB，同時転送ブロック数のパラメータ α=2 の時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図 5 に示す．図 5 より，制御後は各ノードの受信ブロック数の偏りが幾分か解消され，デフォルトと比較すると高いスループットが安定して出ていることが分かる．しかし，制御後もなお時間 40，85 秒では受信ブロック数が 4 となる DataNode が存在しており，レプリカ生成先に偏りが生じ，スループットが低下している．これは 4.1 節の提案手法の (2) が発生している状況である．

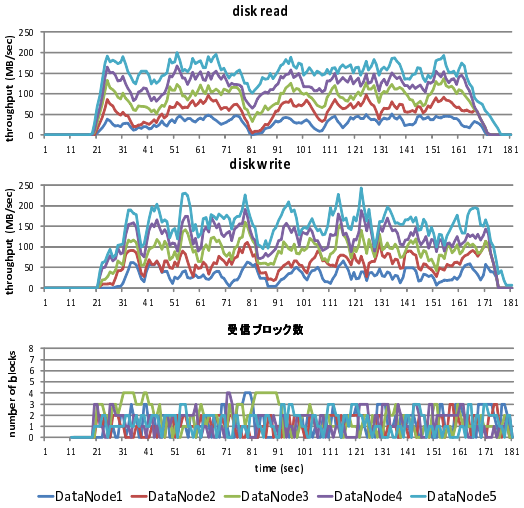

図 5 ディスクの各スループットと受信ブロック数の時系列データ

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 つ次のノードが該当ブロックを保持している可能性は 25%なので，生成先の偏りの原因となる，2 つ先のノードがレプリカ生成先になる事態が 25%の確率で発生すると考えられる．実際に各 DataNode のレプリカ生成先の内訳を調査したところ，表 3 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，制御後もデータの偏りに関して更なる改良の余地があることが分かった．

表 3 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%

5. 関 連 研 究

Felix[4] 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．

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に関する基本性能評価を行った．デフォルトの設定ではレプリカの生成先に偏りが生じ，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，ノードをリング状に配置し，レプリカ生成先をそのリング構造に従って 1 つ先のノードに指定することで，各ノードが等しくデータを受信するような制御手法を提案した．評価実験から提案手法ではデフォルトの時よりも最高で 18%スループットが向上し，各ノードの受信ブロック数の時系列データからデータ移動の偏りが大分解消できていること，そして各ノードのスループットも比較的高い値を維持できていることを確認した．

今後の課題は，現在の制御では依然としてレプリカ生成先に偏りが生じ，ディスクリソースも使い切っていないので，レプリカ生成元，スケジューリングやデータ移動の制御を加えて更なる高速化を目指すことである．また Hadoop では，複数のレプリカは信頼性と読み書きに必要な帯域のトレードオフを考慮して異なるラック上に分散して配置されるため，ラックを意識した制御にも取り組んでいきたい．

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) 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.