システム規模を伸縮させるHDFSのQoS向上

中田　良祐† 　Hieu Hanh Le† 　横田　治夫†

† 東京工業大学　〒 152–8550 東京都目黒区大岡山 2-12-1
E-mail: {nakata,hanhlh}@de.cs.titech.ac.jp, yokota@cs.titech.ac.jp

あらまし　近年、情報の蓄積、収集技術の向上により多くのデータが蓄積されている。このような膨大なデータを要求処理能力に応じた電力で処理するためにシステム規模を伸縮させる分散ファイルストレージシステムが研究されている。その一つにワークロードに応じて稼働ノード数を変化させる方法がある。この際にQoSを低下させる原因として、稼働ノード数変更に伴うデータ移動における一貫性保持のためのリクエスト処理待ち、段階的な規模伸縮によるワークロードの急激な変化への追随の不十分さ等が考えられる。それぞれに対して更新ブロックの移行状態を保持することでリクエストのアクセス先を判断し一貫性を保ちつつ応答する手法、複数段階の伸縮を一度に行うことで急激なワークロードの変化に対応する手法を提案、一部実装し評価した。

キーワード　分散インデックス, ストレージ,HDFS, ギアシフティング,QoS

1. はじめに

ビックデータに代表される近年の巨大な情報の利用はますます求められており、それらの処理にはオープンソースのHadoopのような分散処理環境が適している。分散処理環境とは、処理対象のデータを複数の計算能力をもつノードに分散させ、それらを並列に動作させることにより処理能力を上げる方法である、また複数のノードに複製を持たせることで障害発生時には複製からデータを復元することができ障害に対する耐性が強い。

Hadoop等の分散処理環境によりスループット向上を望めるが、複数ノードが常に稼働している状態になるため当然消費電力も大きくなる。実際にHadoopなどで処理されるワークロードの大きさは図2のように一定ではなく、ここで問題となることはワークロードが小さいときでもシステムが必要以上の性能を提供し余分に電力を消費することである。サーバーの省電力化を行うには図1のように性能とエネルギーが比例するような構成をする必要がある[1]。

そこで、分散ファイルシステムとしてワークロードの大きさに応じてギアシフティングと呼ばれるシステムの規模を伸縮させる仕組みを持つストレージが研究されている[2][3]。これらの方法では異なるギアのデータノードにデータの複製を持たせ、必要なとき、つまりワークロードが大きいときには高ギアのノードを稼働させ並列度をあげ、ワークロードが小さい時には高ギアのノードは停止させることによって消費電力を抑える事ができる(図3)。

データを複製するため、処理が行われると処理されたデータの複製との一貫性を保つために必要なノードには変更のあったデータを移行をさせる必要がある。この移行はギアアップする際に必要であり時間を要する。この移行時間を短縮するために効率的な移行を実現するメタデータ管理法NDCoupling[4]やデータ配置メソッドAccordion[5]が提案されているが、これらの方式ではデータ移行中のリクエスト応答を停止させており、リクエストが停止されている期間が長く明らかにQoSの低下

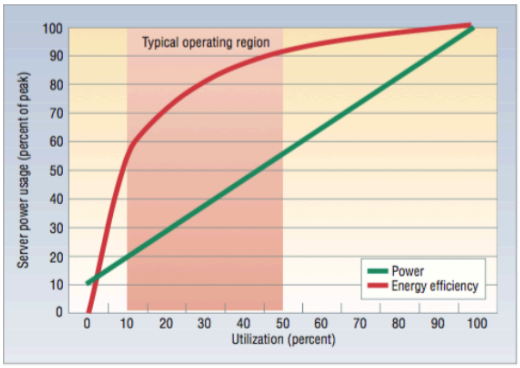

図1　性能と電力が比例したサーバのエネルギー効率[1]

を招いている。この問題を解決するためにメタデータを用いた新しいリクエスト応答の手法が提案されている[6]。この手法は更新ブロックの移行状態を保持し、移行状態に応じてアクセス先を判断し一貫性を保っている。

また別のQoSを低下させる要因としてシステムの俊敏性を超える勢いでのワークロードの増加があった時にリクエスト応答時間が長くなること等が考えられる。ここでいうシステムの俊敏性とはギアシフトによりある一定時間にどれだけシステムの処理性能が上げられるかの程度を表している。本研究の二つ目の目標としてシステムの俊敏性を高める手法を提案する。

本論文では、2. 節で、本研究で着目する分散ファイルストレージシステムにおけるギアシフトについて説明し、3. 節で、関連研究を紹介し、4. 節で、ギアアップ中のリクエスト応答を可能にする仕組みと俊敏性向上の手法を説明し、5. 節で実験の環境と内容を説明し最後にまとめを述べる。

2. ギアシフティング方式

2.1 データの配置とギアシフト

ギアシフティング方式の分散ファイルストレージシステムではデータを複製して複数のノードに持たせワークロードの大き

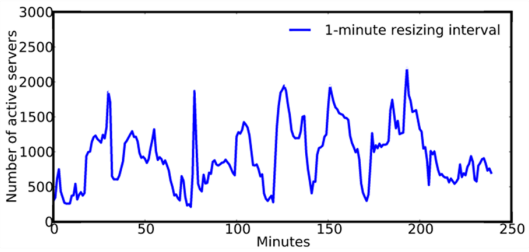

図 2　Facebook のワークロードトレース [7]
必要な性能を 1 分毎にトレースしたもの

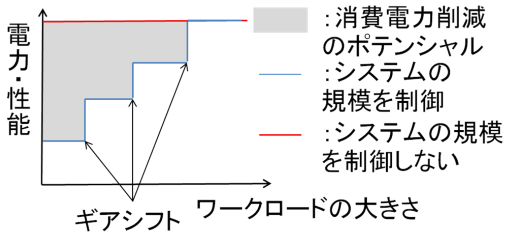

図 3　ギアシフティング方式

さに合わせて稼働ノード数を変化させることでシステムの規模を伸縮する。ギア 1 で動作するノードを $Group_1$ のノード、ギア 2 で新しく稼働するノードを $Group_2$ のノードとする。例えば図 4 で示したような四つのノードがあり 2 段階のギアのシステムを想定した場合、ギア 1 では $Group_1$ の Node 1、Node 2、ギア 2 では $Group_2$ の Node 3、Node 4 を加えた四つのノードで動作すると言った方式が考えられる。ギア 1 で動作する際に $Group_1$ のノードで全てのデータを処理できなくてはならないので、それらのノード内でデータを分散させる必要がある。$Group_2$ のノードには $Group_1$ のノード内のデータの複製（またはその一部）を持たせる。例では、$Group_1$ のノードのデータの半数を $Group_2$ のノードに複製することで 4 並列で処理を行うことができる。

ギア 1 でシステムが動作している時は Node 3 と Node 4 は停止しており、処理能力は小さくなっているが消費電力も小さくなっている。ギア 2 では全てのノードが動作しており、処理能力と消費電力は大きくなっている。このようにギアシフティング方式を使うことによって状況に応じて異なる性能を提供できる。大きな処理能力が必要なワークロード以外ではギア 1 での処理能力で十分だとすると、常に四つのノードで動いているシステムと比べて消費電力を抑えることができる。

2.2　ギアアップ処理に要する時間

低ギアで動作している時にリクエスト処理が行われるとノードのデータに書き込みが発生することがある。この時、書き込みがあったノードのデータと高ギアグループのノードの対応する複製の間にデータの相違が発生する。例えば図 4 で $B1$ が $P1$ の複製であったとする。この時ギア 1 で動作している a に $write$ が発生して a' に変わると、$a' \neq a$ になってしまうため、ギアアップした際に $Node3$ の a は最新のデータではないため使うことができない。そこでギアアップの際には a に a' にあった変更を反映させるためのデータの移行が必要になる。この

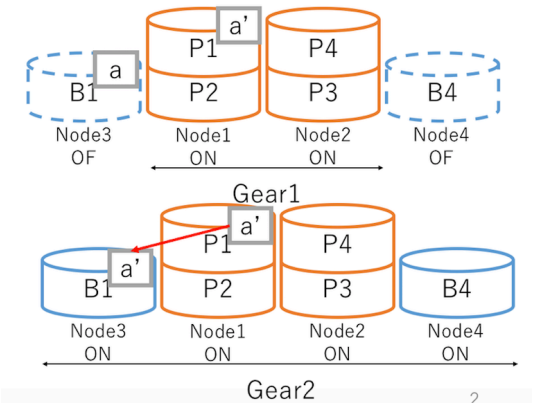

図 4　ギアシフティング HDFS

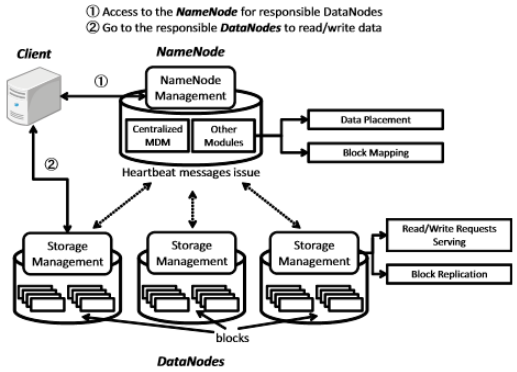

図 5　一般的な HDFS

移行がギアアップ処理に時間がかかる原因の一つであり、ギアアップ中にリクエスト応答ができない原因である。データの移行効率化のためにはシステムのメタデータ管理とデータ配置が重要である。

3.　関 連 研 究

本節では効率的なギアアップ処理の観点から、Hadoop 用ギアシフティングシステムにおいて重要なメタデータ管理とデータ配置についての関連研究について説明する。

3.1　メタデータ管理

3.1.1　NDCoupling

一般的な HDFS は図 5 のように NameNode と呼ばれるノードが一つ存在し、リクエストはまず NameNode にアクセスする。その後、DataNode と呼ばれるデータを分散させているノード群の中から対応するノードに処理を受け渡す。そのため、リクエスト応答の処理が NameNode に集中している。その他、メタデータの管理やデータ配置、ブロックマッピングなどを一括して NameNode が受け持つのでギアアップ時は特に大きな負荷がかかる。

一方 NDCoupling [4] という手法を用いた HDFS では図 6 のように、すべての DataNode に NameNode の仕事を分散させる。各ノードのメタデータ管理モジュールやブロックマッピングは自分のノードのメタデータとブロックを管理すればよい。リクエストは最初ランダムに選ばれたノードへ行き、その後分散されたメタデータ管理モジュールが Fat-Btree [8] の構造を用

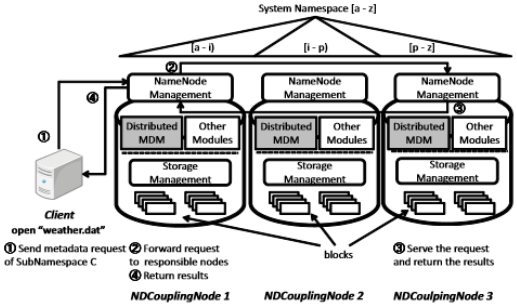

図 6　NDCoupling+HDFS

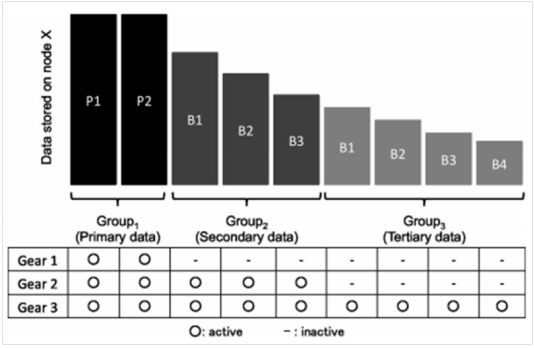

図 7　3 ギア構成の Rabbit データ配置の例

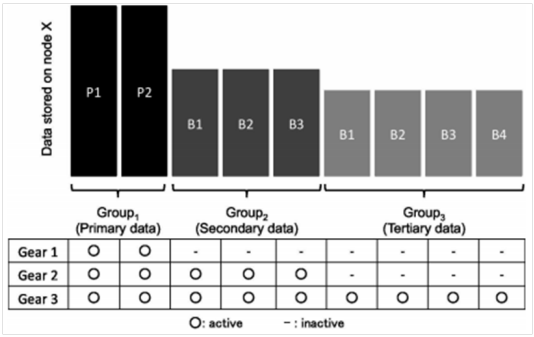

図 8　3 ギア構成の Sierra データ配置の例

いたメタデータ管理を利用して対応するノードへ渡される。処理されたデータは最初にアクセスしたノードを経由しクライアントに返される。この手法によってリクエスト処理が分散せれた。また、データと対応するメタデータなどをノードごとに管理することができるようになり、ギアアップの際のデータの移行をスムーズに行うことが可能になる。

3.2　データ配置

3.2.1　Rabbit, Sierra

分散ファイルストレージにギアシフティング方式を採用した研究の先駆けとなったものが Rabbit [2] と Sierra [3] である。Rabbit では主に Read の性能を上げることを目的として equall-workdata-layout policy という各ノードが同程度の量の仕事を受け持たせることを考慮した配置メソッドを提案している。図 7 のように、policy を達成する計算式に基づいてギア 1 で動くノード上にある primary data の複製を高ギアで動くノードに持たせる。

後に続く Sierra では Rabbit でギア 1 のノードに集中していた write の処理の負荷を各ノードに分散させる方式などを取り入れ、さらに図 8 のように、同ギアのノードの持つデータのブロックサイズを同じにすることによってノード数の制約を緩和した。

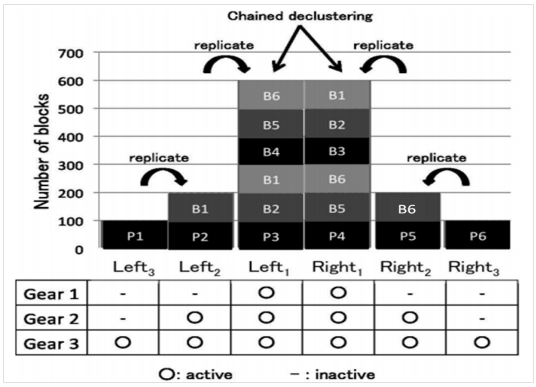

図 9　3 ギア構成の Accordion データ配置

両者ともデータの移行を考慮したデータ配置メソッドではないため、ギアアップ時のデータ移行量が大きく、データ移行は効率的に行うことができない。

3.2.2　Accordion

Accordion [5] では、Rabbit や Sierra とは異なる新たなデータ配置メソッドを提案しギアアップの際の効率的なデータ移行などを実現した。続く論文 [9] で NDCoupling と組み合わせたシステムを提案している。このデータ配置メソッドは図 9 のように、全てのノードに primary data を分割して持たせ、分割ブロック単位で低ギアのノードに複製していく配置方法により、データをブロックで扱えるようになり、ブロックごとに移行やメタデータの管理をすることができる。Accordion は Rabbit に比べて 30%スループットが高く、NDCoupling と Accordion の構成では HDFS と Accordion の構成と比べて 22%スループットが高い。

4.　提 案 手 法

関連研究ではギアアップ中はリクエストをブロックしてデータの移行を行なっている。したがって、性能を上げたい時にギアアップを要請するとしばらくリクエスト応答が停止してしまう。またシステムの俊敏性を超えたワークロードの増加が起こるとしばらくリクエストの応答時間が遅くなってしまう。これらを改善することでシステムの QoS を向上させることを目的に、本研究では以下の手法を提案する。

4.1　提案手法 1:低ギア構成のメタデータ保持

4.1.1　アプローチ

2.2 節で述べたことが原因でギアアップ要請があってからデータの一貫性を保持するためにデータ移行とそのデータのメタデータの修正が行われる。当然、修正中のメタデータを使い、データ移行中にアクセスしても、そのデータが最新のものである保証はない。このため、その間のリクエスト応答を止めている。しかし、ギアアップが開始される前の低ギアのノードのメタデータを使い、低ギアのノードのデータにアクセスすればそれは必ず最新のデータである。そこでギアアップ処理を行う間も低ギアでの構成を保持する。ギアアップ中は保持している

低ギアでの構成を使い低ギアのノードのデータにアクセスさせることで、高ギアのノードに格納されている一貫性の保証されていないデータに触ることなく常に最新のデータにアクセスさせることが可能になる。

4.1.2 低ギア構成を保持しながらのギアアップ

step 1:高ギアのノードへメタデータを複製して移行 (リクエスト応答停止)

この時、高ギア構成に変更するための修正などはまだせず、低ギアの構成を維持

step 2:データの移行、このとき低ギアの構成を使いリクエスト応答

step 3:メタデータの修正を行い、高ギアの構成に変更（リクエスト応答停止）

この手法は step 2 でリクエスト応答が可能であるので、ギアアップ中のリクエスト応答可能時間が増え QoS が向上する。

4.2 提案手法 2:移行状態のメタデータ管理

提案手法 1 はギアアップ中のリクエスト応答を可能にするが、低ギアの構成を使うため提供される性能は定義あのものである。提案手法 2 はギアアップ中に徐々に性能が高ギアのものに近づくのでより、リクエスト応答時間を短縮することができる。

4.2.1 アプローチ

ブロック毎に、メタデータにデータの移行状態を持たせてアクセス先のデータノードを決定させる。移行状態は**移行未**と**移行済**があり、**移行済**の場合のみ高ギアのノードもすでにデータは最新のものである。この時、メタデータの指す低ギアのノードにアクセスする代わりに高ギアのノードにアクセスさせる。こうすることによってギアアップ中に処理性能が徐々に上がりシステムの柔軟性が向上する。

4.2.2 手　　法

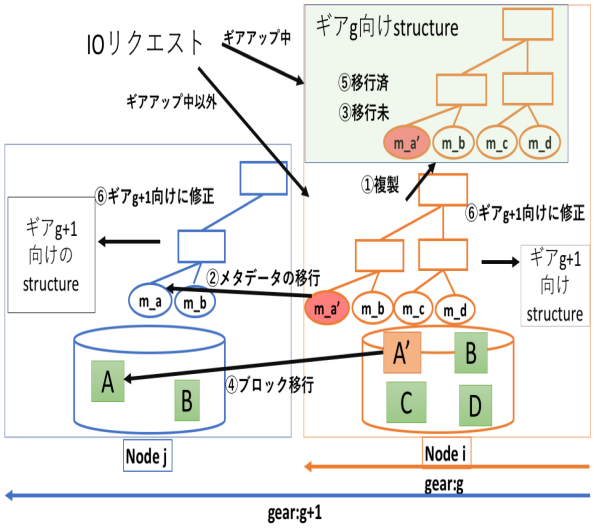

図 10　移行状態を管理しながらのギアアップ処理

提案手法の実現手順について図 10 を使い説明する。Node i はギア g のノード、Node j はギア g+1 のノードである

step 1:メタデータを複製してギア g のメタデータとする。元のメタデータは更新のあったメタデータを移行するなどの修正を行う。

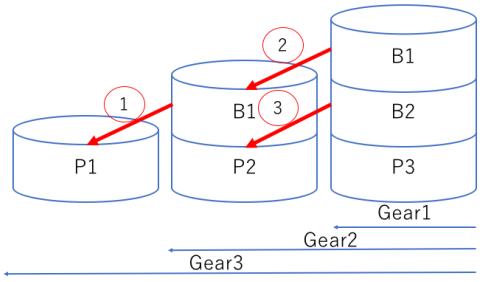

図 11　1 段階ずつギアシフト

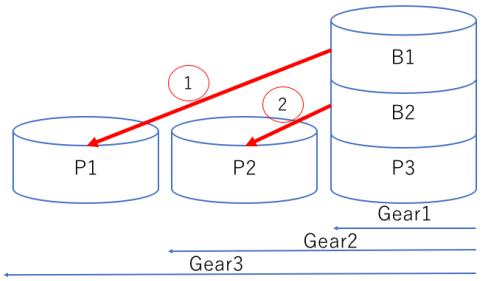

図 12　複数段階同時ギアシフト

step 2:移行状態を持たせる。

データの書き込みがあった場合、各ノードのログファイルに記録される。稼働中のノードのログファイルを読み移行の必要があるデータを選択してまず移行未の状態を持たせる。図の例では Node i のデータ A' が移行の必要があり、そのメタデータ m_a' に**移行未**の状態を持たせる。

step 3:データブロック移行のコマンドを発行する。

各ノードのストレージマネジメントモジュールのキューにデータブロックと移行先のノードを挿入する形でコマンドが発行される。ストレージマネジメントモジュールはキューからコマンドを取り出しデータ移行を実行する。

step 4:書き込みのあったブロックの移行

移行コマンドが取り出され指定されたノードへの移行が実行される。

step 5:移行状態の更新

ブロックが移行されたら step 2 で持たせた移行状態を**移行済**に変更する。

step 6:メタデータを修正してギア g+1 向けのメタデータに変える。

step 7:リクエスト応答処理をギア g 向けのものからギア g+1 向けに変更する。

このギアアップ処理の間は、複製したギア 1 向けのメタデータを使いリクエストに応答することで、ギアアップ中のリクエスト応答を可能にする。

4.3 提案手法 3:複数段階同時ギアアップ

Accordion のデータ配置でギア g からギア g+n までの n 段階のギアアップを一度にすることを考える。ギア g+n で primary として扱われるデータのみが最新の状態にできればギア g+n

	Node1	Node2	Node3	Node4
ギア 1		○	○	
ギア 2	○	○	○	○

図 13　ギ ア 構 成

ノード	ASUS EeeBox EB1007
OS	Linux Ubuntu 11.10
CPU	Intel (R) Atom CPU D410 1.66GHz
ディスク	HDD 50GB
メモリ容量	4GB
Java version	1.7
Network	100Mb/s

図 14　ノードスペック

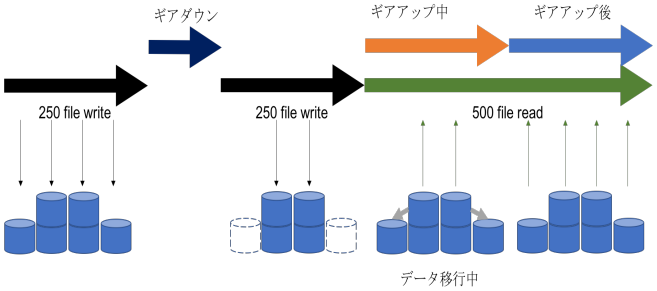

図 15　実 験 手 順

の性能を提供できるため。backup となるデータの更新を後回しにすることで更新回数を減らしギアアップにかかる時間を減らす。例として、３段階のギアを持つシステムを考える。ギア１で動作している時に、急激なワークロードの増加が起きてギア３の処理性能を要求されたというような場合にこの複数段階ギアアップが役に立つ。１段階ずつのギアシフトの場合は図 11 の赤矢印が示すように３ブロックの移行が必要だが、提案する手法の複数段階同時ギアシフトは図 12 のようにギア２の状態を経由しないため移行するブロックの数が２つに減るのでより早くギア３にシフトすることが可能である。データの移行が行われなかったギア２のノードへはギアアップ処理が終了した後、データの移行が行われデータが最新のものとなる。

5.　実　　験

本論文では提案手法 1 の効果を検証する実験を行う。提案手法 2,3 に関する実験は今後、発表予定である。

5.1　実 験 環 境

ギア構成は図 13 に指名したように、Accordion + NDCoupling 構成で４ノード、２段階ギア、ギア２では全４ノードが稼働し、ギア１では２ノード稼働とした。ノードスペックは図 14 のようになっている。

5.2　実 験 手 順

実験を図 15 が示す以下の手順で行なった。

step １：ギア２で２５０ファイル書き込み

step ２：ギア１で２５０ファイル書き込み

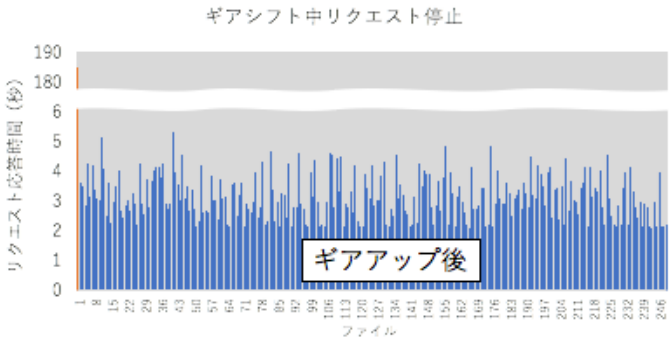

図 16　従来のギアアップ

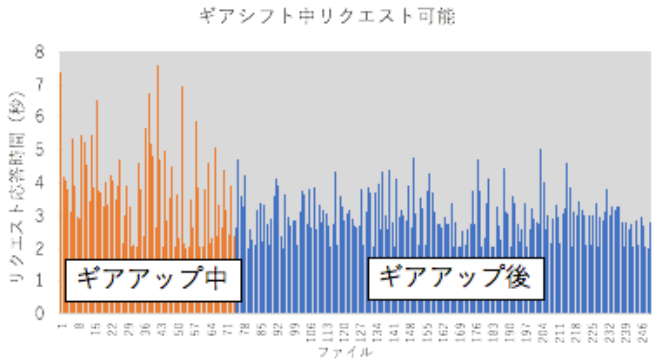

図 17　提案手法 1 のギアアップ

step ３：ギア１で５００ファイル読み込みと同時にギアアップ開始

step ４：ギアアップ終了後、ファイル読み込み完了

提案手法の一部を実装したギアアップとリクエスト応答を受け付けない（従来の）ギアアップをそれぞれリクエスト処理中に行い、読み込みにかかる時間を比較した。

5.3　実 験 結 果

図 18 は従来のギアアップと提案手法 1 を実装したギアアップを比較した表でありる。ギアアップとリクエスト応答が同時に行われている時（図 16 のオレンジの線が示す時間）は従来手法はリクエストがギアアップが終了するまで待たされるため 186 秒かかる。提案手法 1 を実装したギアアップではギアアップとリクエスト応答が同時に実行されていることが確認でき、図 17 のオレンジ線がそのリクエスト応答時間である。リクエスト応答時間は平均約 3.84 秒であり通常時のリクエスト応答時間 (図 17 の青線) の平均 3.01 秒と比較しても約 0.8 秒の性能低下に抑えられている。500 ファイルの read にかかる時間（図 15 の緑の矢印が示す時間）も提案手法 1 を実装した方では 808 秒と従来手法と比べて約 160 秒短縮されている。このように提案手法 1 の有効性が確認できた。この原因は今回の前提ではノード間の通信速度が 100Mbps であり従来のギアアップでは cpu に余裕があったためと考えられる。

6.　ま　と　め

本研究ではギアシフティングする HDFS における QoS の低下の原因となる二つの問題を提示し、それぞれに解決のための

	ギアシフト中 Read不可	ギアシフト中 Read可能
ギアシフト中 平均応答時間(s)	186	3.84
ギアシフト後 平均応答時間(s)	3.15	3.01
500file合計read 時間(s)	971	808

図 18　実 験 結 果

手法を示している。

一つは、ギアアップ中のリクエスト応答を止めることが問題であり、その解決法として低ギアのメタデータの保持と更新するデータの移行状態をメタデータとして管理し、状態に合わせてアクセス先ノードを選択する方法 [6] を示した。

一つは、急激なワークロードの増加が起こった時、システムが提供する性能の向上が間に合わないという問題があり、解決法として速く性能を向上させる手法として段階を飛ばしてギアアップする手法を示した。

低ギア構成のメタデータを保持したギアシフトの実験では応答時間が短くなっていることが確認できた。

今後の課題として提案手法 1 をより大規模で行えるよう修正し、提案手法 2、3 を評価実験すること、今回の手法ではギアアップ中は read リクエストの応答のみ可能なので write リクエストへの応答方法を検討する必要がある。また、提案手法 3 の複数段階ギアシフトは一段階のギアシフトとの使い分けの基準を確立する必要が挙げられる。

文　　献

[1] L.A. Barroso and U. H olzle, “The Case for Energy-proportional Computing,”Computer, vol.40, pp. 33-37, 2007

[2] A.Hrishikesh, C.Hames, G.Varun, G.R.Ganger, .Michael A., and S.Karsten, “ Robust and flexible power-pproportional storage.. ”Proc, 1st Symposium on Cloud Computing, pp.217-228, 2010.

[3] E.Thereska, A.Donnelly, and D.Narayanan, ”Sierra: Practical power-proportionality for data center storage, ”Proc. 6th European Conference on Computer Systems, pp.169-182, 2011.

[4] H.H. Le, S. Hikida, H Yokota, ”NDCouplingHDFS: A Coupling Architecture for a Power-Proportional Hadoop Distributed File System,”IEICE Trans. inf. & Syst., vol.E97D, no.2, pp.213-222, Feb.2014.

[5] H.H. Le, S. Hikida,and H. Yokota, Accordion: ”An Efficient Gear-Shifting for a Power-Proportional Distributed Data-Placement Method ”IEICE Trans. inf. & Syst., Vol.E98-D No.5 pp.1013-1026, 2015.

[6] 杉山翔一郎, レーヒェウハン, 横田治夫, “システム規模を伸縮させる HDFS におけるデータ移行中のリクエスト応答調整“ DEIM Forum 2017 H5-1.

[7] L. Xu, J. Cipar, E. Krevat, A. Tumanov, N. Gupta, M. A. Kozuch and G. R. Ganger, Slide of ”SpringFS: Bridging Agility and Performance in Elastic Distributed Storage”, Proc. of the 12th USENIX FAST, 2014, pp. 243-255.

[8] H. Yokota, Y. Kanemasa, and J. Miyazaki, ”Fat-Btree: An update-conscious parallel directory structure, ”Proc. of the 15th ICDE, pp.448-457, March 1999.

[9] H.H. Le, S. Hikida,and H. Yokota, ”An Efficient Gear- Shifting Power-Proportional Distributed File System. ”International Conference on Database and Expert Systems Applications. Springer International Publishing, 2015.