Vol.2013-HPC-138 No.23
2013/2/22

MapReduce ジョブ実行時の不要計算資源解放手法

成瀬宏輔[†1]　　合田憲人[†2 †1]

大規模データを処理するフレームワークとして Hadoop が注目されている．Hadoop では，ジョブ実行中にアイドル状態の計算ノードが発生し，資源の利用効率が下がる場合がある．本稿では，利用者が予め定めたジョブの実行時間（締切）を守りながら，アイドル状態にある不要な計算ノードを解放する（離脱させる）ことを目的として，Hadoop ジョブの実行時間の解析結果を示すとともに，不要な計算ノードの解放手法を提案する．

1. 序論

近年，インターネットや通信機能を持つセンサ等の電子機器の普及によって大規模データが日々生成されている．こうした大規模データを現実的な実行時間で処理するためのソフトウェアフレームワークとして Hadoop[1]が注目され，プライベートクラウドやパブリッククラウド（IaaS）上で利用されている．Hadoop では，入力データが HDFS と呼ばれる分散ファイルシステムに保存される．ジョブの実行時には，Map タスクと呼ばれる複数のタスクが HDFS から読み込んだデータを並列に処理し，Map タスクの出力を Reduce タスクと呼ばれる（複数の）タスクがまとめて最終的な結果を得る．

Hadoop ジョブを実行した場合，一般的に Reduce タスクの並列度は Map タスクに比べて小さいため，実行途中からアイドル状態の計算ノードが現れる場合がある．これらのアイドル状態の計算ノードは，ジョブの実行終了まで利用されないため，計算ノード全体の利用効率を低下させるほか，パブリッククラウド（IaaS）を利用する場合は，無駄な利用料が発生するという問題がある．

本稿では，利用者が予め定めたジョブの実行時間（締切）を守りながら，アイドル状態にある不要な計算ノードを解放する（離脱させる）ことを目的として，Hadoop ジョブの実行時間の解析結果を示すとともに，不要な計算ノードの解放手法を提案する．Hadoop ジョブの実行時間解析では，ジョブ内の各タスクの実行時間をベンチマークプログラムにより評価し，タスクの実行時間の特徴を示す．本解析結果は，次に述べる提案手法において，ジョブの実行時間を予測するために用いられる．提案手法では，Map フェーズを締切までに終了させるために必要な計算ノードをジョブ実行時に予測し，不要な計算ノードを離脱させる．

計算ノードの離脱については，考慮すべき問題がある．例えば，ジョブの実行中に Reduce タスクを担当する計算ノードを離脱させてしまうと，Reduce タスクが完了しないという問題や，Map タスクの出力のコピーが再度別の計算ノードに対して行われ，性能が低下してしまうという問題が起こる．また，ジョブの実行性能低下を防ぐためには，データのローカリティや各計算ノードの実行状況を考慮しながら離脱させる計算ノードを決定する必要がある．提案手法では，これらの問題を考慮しながら，適切に離脱させる計算ノードを決定する．

Hadoop のスケジューリング機構については，複数のジョブ間で効率よく計算ノードを割り当てる手法 [2][3] や，単一ジョブ内でのタスクがアクセスするデータのローカリティを向上させる手法[4]が提案されている．しかし，これら既存手法では，不要な計算ノードを離脱させることにより，計算資源全体の利用効率を向上させることには言及されていない．

以後，2 節では Hadoop および関連研究について述べる．3 節では，提案手法でタスクの実行時間推定を行うために必要となる Hadoop ジョブの実行時間解析の結果を示し，4 節では提案手法について述べる．5 節では本稿のまとめと今後の課題について述べる．

2. 関連研究

本節では，本稿が対象とするソフトウェアフレームワークである Hadoop について解説するとともに，提案手法の関連研究について述べる．

2.1 Hadoop

Hadoop を用いたジョブの実行は，Map フェーズ，Shuffle フェーズおよび Reduce フェーズから構成される．Map フェーズでは，Map タスクと呼ばれるタスクが各計算ノード上でユーザの指定した関数を実行して入力データを処理し，処理結果を key と value の形式で出力する．Hadoop 上でのタスク実行はマスタ・ワーカ方式に基づいており，1 台の計算ノード（マスタ）が全タスクの管理を行い，複数の計算ノード（スレーブ）がマスタからタスク受け取り，実行する．

Hadoop 上で扱われるデータは，HDFS と呼ばれる分散ファイルシステムに保存される．HDFS では，データが複数のデータ・ブロックに分けられ，複数の計算ノード上に分散してデータ・ブロックが保存される．Map タスクは，計算ノード間の通信を低減するためにデータのローカリティを考慮し，可能な限りネットワーク的に近い計算ノード上のデータ・ブロックを入力して処理を実行する．

†1 東京工業大学
Tokyo Institute of Technology
†2 国立情報学研究所
National Institute of Informatics

Map タスクを実行する計算ノードと入力のデータ・ブロックが保存される計算ノードが同一の場合は，ノードローカルと呼ばれる．また，Map タスクを実行する計算ノードと入力のデータ･ブロックが保存される計算ノードが異なるが，同一ラック内に配置されている場合は，ラックローカルと呼ばれる．HDFS 上のデータ・ブロックは，主に耐故障性の向上を目的として，複数の計算ノード上に複製（レプリカ）が作成される．

Shuffle フェーズでは，同じ key のデータが同じ計算ノード（Reduce 担当ノード）に集約されるようにデータの交換が行われる．また，データの交換は各データ・ブロックの処理が完了次第，即ち，Map フェーズとほぼ同時進行で行われる． Reduce フェーズでは，各計算ノードが同一 key のデータを入力し，ユーザが指定した関数を実行して最終結果を出力する．

2.2 関連研究

Hadoop の実行環境（Hadoop クラスタ）上で複数のジョブを実行する場合のジョブスケジューリング手法として，FairScheduler[2]や CapacityScheduler[3]が提案されている．Fair Scheduler では，スケジューラが各ジョブ間で利用する計算ノードが公平になるように，計算ノードの割り当てを決定する．CapacityScheduler では，独立したユーザや組織が Hadoop クラスタ上でジョブを実行する場合に詳細な制御を可能にしている．

Hadoop 上でのジョブ実行では，データのローカリティを確保することが，ジョブの実行性能を向上させるために重要である．データローカリティの確保を目的として，ノードローカリティを確保できるタスクがない場合に一定時間スレーブへのタスク割り当てを行わず，ラックローカリティのタスクを割り当てる手法[4]が提案されている．また，タスクの実行時間を見積もり各ジョブの実行時間の制約を守りながら，複数のジョブ間で適切にリソースを共有する手法[5][6]や，実行時間の制約を守るために必要なリソース量を推定する手法[7]が提案されている．

これらの既存研究では，Hadoop クラスタをマルチユーザ環境で利用する場合の資源の利用効率を向上させることや，データのローカリティの確保やタスクの実行時間推定によりジョブの実行性能を向上させることを目的としている．しかし，本稿が提案する不要な計算ノードを離脱させることにより，計算資源全体の利用効率を向上させる技術については議論されていない．

3. Hadoop ジョブの実行時間解析

本節では，Hadoop ジョブ内の各タスクの実行時間をベンチマークプログラムにより評価した実験結果を示すとともに，本実験結果より Hadoop ジョブの実行時間推定を行う方法を議論する．本解析結果は，提案手法においてジョブの実行時間を予測するために用いられる．

3.1 実験環境

本実験では，Amazon EC2 上[8]に Hadoop クラスタを構築し，Hadoop ジョブ実行時の Map タスクおよび Reduce タスクの実行時間やその変動を測定した．実行した Hadoop ジョブは，楽天社が提供している楽天市場という E コマースサービスのクチコミデータ[9]に対して，Hadoop 付属のプログラムであるソート，グレップ，ワードカウントをそれぞれ実行するものである．入力データは，タブで区切られたテキストファイルで，サイズが 70GB である．また，ソートでは，Reduce タスクを担当する計算ノード数を 2 台，その他では 1 台使用し，各プログラムについて 10 回の実行を行った．スレーブ 1 台あたりの最大割り当て可能 Map タスクは 1 つに設定した．なお、データ・ブロックサイズは 512MB に設定した。

表 1　**予備実験における Hadoop クラスタの構成**

OS	Hadoop	マスタ	スレーブ
Ubuntu 10.04	0.20.3	m1.small 1 台	m1.small 9 台

3.2 Reduce 担当ノード上でのタスク実行時間

表 2 は，Map タスクと Reduce タスクを実行する計算ノード（Reduce 担当ノード）と Map タスクのみを実行する計算ノード（Map のみ担当ノード）上での Map タスク実行時間の平均と変動係数（=標準偏差/平均値）を示す．

表 2　**Reduce 担当と Map のみ担当ノード上でのタスク実行時間の比較**

	Map のみ担当		Reduce 担当	
	平均(sec)	変動係数	平均(sec)	変動係数
ソート	56.600	0.125	226.800	0.477
グレップ	447.300	0.080	462.000	0.061
ワードカウント	139.900	0.190	298.700	0.290

表 2 より，ソートとワードカウントに関しては，Map のみ担当ノードが実行したタスクの実行時間のほうが，Reduce 担当ノードに比べてかなり短いことがわかる．これは，各 Map タスクの出力のコピー先が Reduce 担当ノードに集中するため，Reduce 担当ノードに負荷がかかるためだと考えられる．グレップでは Map 出力が小さいため，コピーするデータ量が 1.2GB ほどと少なく負荷が小さかったと考えられる．

また，各ノード上で最初に実行される Map タスクが完了するまでは，Map タスクの出力のコピーが起こらないため，Reduce 担当ノードにコピーの負荷がかからない．そのため，Reduce 担当ノード上でのタスクの実行時間は，ジョブの実行開始当初のものとそれ以降では差があると

考えられる．この現象を確認するために，Reduce 担当ノード上で実行されたタスクのうち，ジョブ実行開始当初に実行されたタスクを除いて平均実行時間を算出した結果を表 3 に示す．

表 3 の結果より，ジョブ実行開始から 2 回目のタスクの実行時間を除いて算出すると平均と変動係数は安定することがわかる．

表 3 ジョブ投入直後のタスクを取り除いた場合の検証（Reduce 担当ノード）

	ソート		グレップ		ワードカウント	
	平均 (sec)	変動係数	平均 (sec)	変動係数	平均 (sec)	変動係数
除去なし	226.848	0.477	462.033	0.061	298.747	0.293
1 回目までを除去	271.468	0.256	462.239	0.063	320.073	0.224
2 回目までを除去	297.576	0.197	460.811	0.063	333.000	0.204
3 回目までを除去	298.821	0.212	460.828	0.064	331.755	0.208

3.3 Map のみ担当ノード上でのタスク実行時間

Map のみ担当ノードについても，各計算ノード上の 1 つ目のタスクが完了しなければ，コピーの負荷がかからない状態で，次のタスクの実行が行われる．そのため，表 3 と同様の検証を Map のみ担当ノード上のタスク実行時間についても行った結果を表 4 に示す．

表 4 の結果より，全ての場合においてジョブ実行開始当初のタスク実行時間を除去しても除去しなくても，結果に変化がなかった．よって，Map のみ担当ノードのコピー負荷による性能低下は考慮する必要がないと考えられる．

表 4 ジョブ投入直後のタスクを取り除いた場合の検証（Map のみ担当ノード）

	ソート		グレップ		ワードカウント	
	平均 (sec)	変動係数	平均 (sec)	変動係数	平均 (sec)	変動係数
除去なし	56.638	0.125	447.285	0.081	139.862	0.191
1 回目までを除去	57.139	0.120	448.198	0.063	140.514	0.193
2 回目までを除去	57.303	0.120	448.914	0.084	141.076	0.195
3 回目までを除去	57.256	0.122	449.274	0.087	141.180	0.200

3.4 計算ノード間の性能差

パブリッククラウドである Amazon EC2 では，計算ノードを各ユーザが占有することが保証されないため，同じインスタンス間でも性能にばらつきがある可能性がある．この現象を確認するため，各計算ノード（slave）が実行したタスクの実行時間の平均値と変動係数を計算して比較した．Map のみ担当ノード上でソート，グレップ，ワードカウントをそれぞれ実行した場合のタスクの実行時間を表 5 に示す．なお，3.3 節の結果からジョブ実行開始当初のタスクの実行時間の除去は行なっていない．

表 5 より，異なる計算ノード上でのタスクの実行性能には差があることがわかる．ソートでは，最も速いリソースと遅いリソースでは，平均で約 18%，変動係数で約 57%の差があった．グレップでは，平均で約 9%，変動係数で約 66%の差があった．ワードカウントでは，平均で約 26%，変動係数で約 52%の差があった．

本評価結果より，同一の Map のみ担当ノード上でのタスク実行時間については，おおむね変動係数も 10%未満であり，差が少ないことがわかる．従って，一部の実行済みタスクの実行時間から，同一の計算ノード上での未実行タスクを含めた全体の実行時間を推定することが可能であるといえる．一方，計算ノード間の実行時性能差は大きいことがわかり，この結果から，不要な計算ノードを離脱させる場合は，候補となる計算ノードの実行時性能を考慮する必要があるといえる．

表 5 各計算ノード上でのタスク実行時間

	ソート		グレップ		ワードカウント	
Ip	平均 (sec)	変動係数	平均 (sec)	変動係数	平均 (sec)	変動係数
slave0	58.511	0.064	442.675	0.062	130.746	0.100
slave1	52.330	0.098	452.414	0.117	131.898	0.209
slave2	63.653	0.080	426.178	0.040	122.707	0.128
slave3	58.506	0.064	467.162	0.061	166.148	0.195
slave4	58.646	0.085	446.664	0.094	130.213	0.182
slave5	59.683	0.159	457.322	0.057	162.007	0.140
slave6	53.890	0.071	444.497	0.090	137.704	0.118
slave7	58.183	0.055	461.321	0.058	162.167	0.145
slave8	52.167	0.188	441.774	0.074	137.707	0.129
全体	56.638	0.125	447.284	0.081	139.862	0.191

3.5 タスク実行時間の平均値算出

一部の実行済みタスクの実行時間から，同一の計算ノード上での未実行タスクを含めた全体の実行時間を推定するためには，推定に用いる実行済みタスク数を決定する必要がある．図 1，図 2 および図 3 は，各 Map のみ担当ノード（slave）上でジョブ実行開始直後に実行されたタス

クの平均実行時間を示している．例えば，図中の 0～2 は，最初（0 番目）から 2 番目までの 3 個のタスクの平均実行時間を意味する．これらの結果より，5 個以上の実行済みタスクから算出した平均値はほぼ一致しており，5 個の実行済みタスクの平均実行時間からジョブ全体の実行時間を推定可能と考えられる．

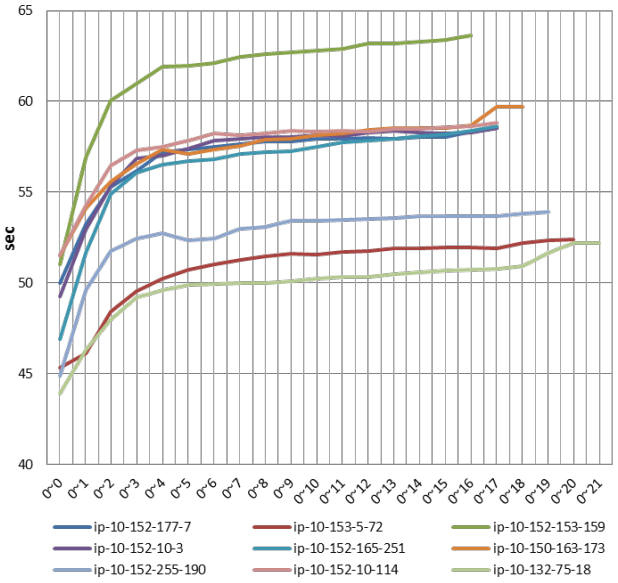

図 1　ソートの平均値の安定化に関する検証

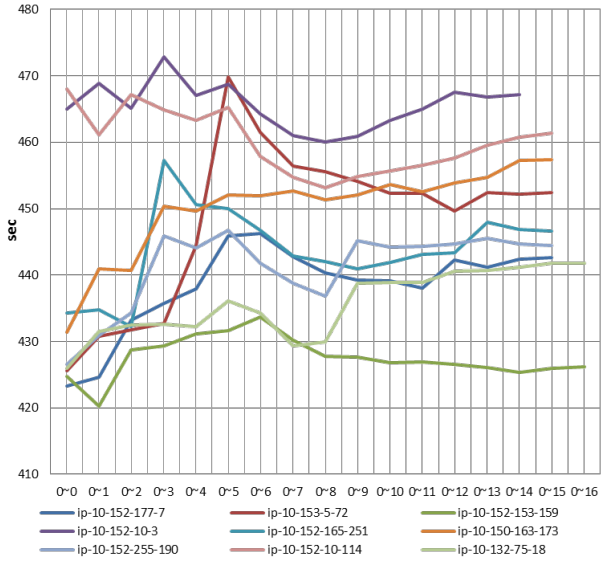

図 2　グレップの平均値の安定化に関する検証

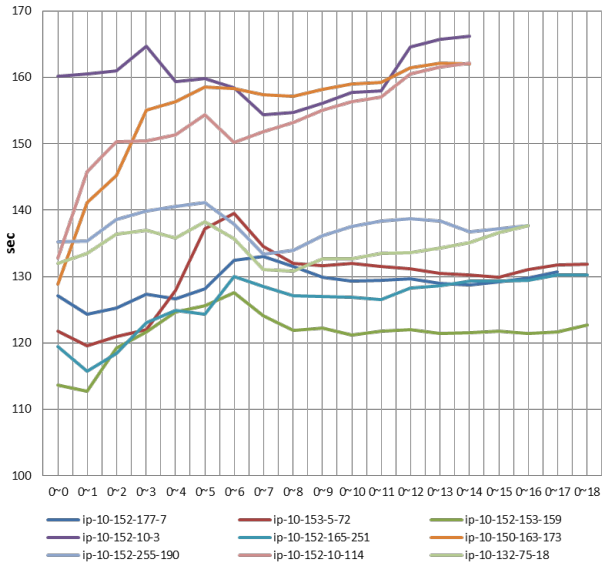

図 3　ワードカウントの平均値の安定化に関する検証

4. 提案手法

本節では提案手法について述べる．提案手法では，利用者が予め定めた Map フェーズの実行時間（締切）を守りながら，アイドル状態にある不要な計算ノードを離脱させる．これを実現するため，計算ノード（スレーブ）上のタスク実行が終了する度に，スケジューラが残りタスクの実行終了時間の推定を行い，当該計算ノードを離脱させても締切までに実行終了が可能と判断される場合に当該計算ノードを離脱させる．以後，図 4 に従って，提案手法のアルゴリズムを述べる．

4.1 離脱可能条件の判定

Hadoop におけるタスク割り当てでは，各計算ノード（スレーブ）がタスクの実行が終了すると HeartBeat をマスタに送信し，マスタは，スケジューラを呼び出して次に割り当てるタスクを決定し，当該計算ノードにそのタスクなどの情報を HeartBeatResponse として送信する．スケジューラは，マスタが計算ノード（スレーブ）から HeartBeat を受け取ると，ます，当該計算ノードを離脱させても以後の処理の継続が可能かどうかを判定する．具体的にはまず，当該計算ノードが Reduce 担当ノードである場合は，Reduce 担当ノードは離脱させると以後のジョブの実行に支障があるため，離脱不可と判定し，当該計算ノードに新たなタスクを割り当てる．

次に，提案手法では実行済みタスクの平均実行時間から残りのジョブの実行時間を推定するため，実行時間推定に必要な数のタスクの実行が終了しているかどうかを判定する．具体的には，当該計算ノードが Reduce 担当ノードの場合は実行済みタスク数が 3 個，Map のみ担当ノードの場合は実行済みタスク数が 5 個に満たない場合は，離脱不可と判定し，当該計算ノードに新たなタスクを割り当て

る．

Hadoop の処理では，耐故障性向上のため，全データ・ブロックのレプリカが保存されている．スケジューラは，当該計算ノード上に保存されている全データ・ブロックのレプリカ数が２以上あるか判定する．レプリカ数が１のデータ・ブロックが当該計算ノード上に保存されている場合，当該計算ノードを離脱させてしまうとジョブの入力データに欠損がでてしまうため，離脱不可として当該計算ノードに新たなタスクを割り当てる．

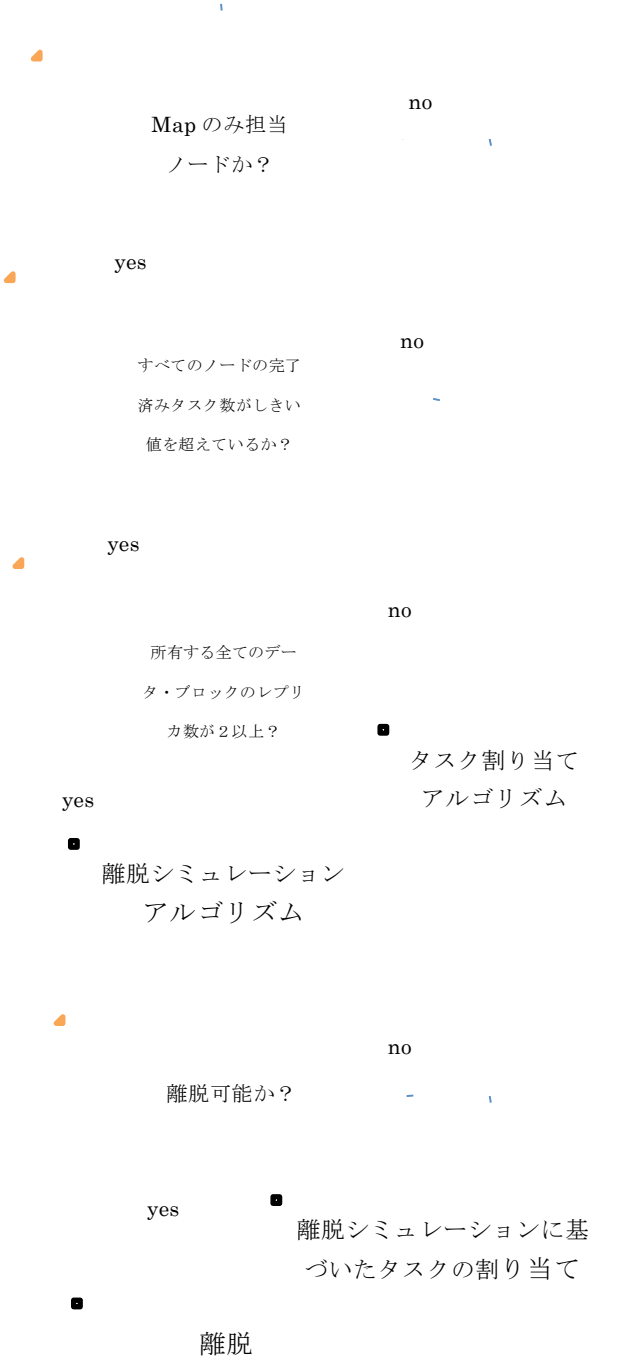

図 4　アルゴリズムの全体像

4.2　タスク割り当て

4.1 節の判定において，マスタに HeartBeat を送信した計算ノード（スレーブ）が離脱不可と判定された場合，スケジューラは，以下の手順でデータローカリティを考慮したタスク割り当てを行う．まずスケジューラは，当該計算ノード上に保存されているデータ・ブロックのうち，未実行の Map タスクの入力となるデータ・ブロック（未処理データ・ブロック）を抽出する．次に，未処理データ・ブロックのうち，レプリカ数が最小のデータ・ブロックを選択し，選択されたデータ・ブロックを入力とするタスクを当該計算ノードに割り当てる．割り当て対象のタスクが複数存在する場合は，タスク id が最小のタスクが割り当てられる．

4.3　離脱シミュレーション

4.1 節の判定において，HeartBeat を送信した計算ノードの離脱が可，即ち当該ノードを離脱させても以後の処理を継続可能と判定した場合には，以下の手順で当該計算ノードを離脱させた場合のシミュレーションを実施する．シミュレーションの結果，締切時刻までに実行終了可能と判定された場合には，スケジューラが当該計算ノードを離脱させる．

具体的にはまず，各計算ノード上の実行済みタスクの平均実行時間から，Map フェーズの締切までに実行終了可能なタスク数（実行可能タスク数）を算出する．なお 3.2 節の結果から，Reduce 担当ノードについては，最初の 2 個のタスクを除いて平均実行時間を算出する．

次に，計算ノード上へのタスク割り当てシミュレーションを実施し，離脱可能な計算ノードを求める．Reduce 担当ノードへのタスク割り当てシミュレーションでは，レプリカが最小の未処理データ・ブロックを入力とするタスクから順番に，実行可能タスク数のタスクを Reduce 担当ノードに割り当てる．割り当て候補のタスクが複数存在する場合は，タスク id が最小のタスクが割り当てられる．次に，Map のみ担当ノードへのタスク割り当てシミュレーションでは，実行可能タスク数が大きい Map のみ担当ノードから順番に Reduce 担当ノードと同じ方法でタスクを割り当てる．ただし，ノードローカリティを確保するため，Map のみ担当ノード上の未処理データ・ブロック数が実行可能タスク数未満の場合は，当該計算ノードに割り当てるタスク数を未処理データ・ブロック数と同数とする．

本シミュレーションの結果，タスクを割り当てられなかった計算ノード群は不要と判断され，その中に HeartBeat を送信した計算ノードが含まれていれば，当該ノードを離脱可能と判定する．離脱可能な計算ノードがない場合は，スケジューラが当該ノードに新たにタスクを割り当てる．

5.　まとめと今後の課題

本稿では，Hadoop ジョブの実行性能の解析結果を示す

とともに, 本結果を用いて設計した不要な計算ノードの解放手法を提案した．現在，提案手法の実装を行っており，今後, 性能評価を行って提案手法の有効性を検証する予定である．

謝辞　本研究の一部は，科研費（24240006）の助成を受けたものである．

参考文献

1) Welcome to Apache™ Hadoop . http://hadoop.apache.org/
2) Fair Scheduler Guide - Apache™ Hadoop. http://hadoop.apache.org/docs/r0.20.2/fair_scheduler.html
3) Capacity Scheduler Guide - Apache™ Hadoop. http://hadoop.apache.org/docs/r0.20.2/capacity_scheduler.html
4) Zhang, X., Zhong, Z., Feng, S., Tu, B., & Fan, J. Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments. 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications, 2, 120–126. doi:10.1109/ISPA.2011.14 (2011).
5) Polo, J., Carrera, D., & Becerra, Y.. Performance-driven task co-scheduling for mapreduce environments. 2010 IEEE. (2010)
6) Polo, J., & Nadal, D. De.. Adaptive task scheduling for multijob mapreduce environments. … multijob-mapreduce (2009)
7) Kc, K., & Anyanwu, K.. Scheduling Hadoop Jobs to Meet Deadlines. 2010 IEEE Second International Conference on Cloud Computing Technology and Science, 388–392. doi:10.1109/CloudCom.2010.97 (2010)
8) Amazon EC2. http://aws.amazon.com/jp/ec2/
9) 楽天データ公開 http://rit.rakuten.co.jp/rdr/