Vol.2013-OS-126 No.11
2013/7/31

# システム状態の変化にロバストな異常タスク特定手法を備えたMapReduceタスクスケジューラ

浅原 理人[1,a)]　藤森 偉恭[1]　成田 和世[1]　劉 健全[1]

**概要**：MapReduce のような分散並列処理基盤では，実行速度が異常に遅いタスクが突発的に発生することでジョブの完了が遅れる現象が観測されている．これに対して，異常タスクを動的に特定して再実行することでジョブの完了時間を短縮する手法が提案されている．ところが従来手法は，投入されたジョブの種類や計算機環境といったシステム状態が一定であることを仮定しているため，システム状態の変化に弱いといえる．本論文ではシステム状態の変化にロバストな異常タスク特定手法を備えた MapReduce タスクスケジューラを提案する．提案手法はシステムの状態が変化する前の処理完了時間の影響を受けにくくするために，2 点の工夫を行う．ひとつは，判定基準となるサンプルを取得する範囲を規定するタイムウィンドウによる過去情報のフィルタリングである．もうひとつは，処理時間の確率分布モデルをポアソン分布を仮定して推定し，分布と矛盾するタスクを異常と判定する仕組みである．実験では，システム状態が変化する状況において，従来手法を模した異常タスク判定手法に対して提案手法の誤判定回数が 96.4%減少することを確認した．その結果，処理時間が 84.0%減少することを確認した．

## 1. はじめに

MapReduce [1] や Dryad [2] を例とする，大量のデータを対象とした分散並列処理基盤が提案されている．こうした基盤は多数の安価な計算機で構成された計算機クラスタの上で動作し，利用者に対して計算機クラスタの共有環境を提供する．利用者はこの共有環境上で，計算機クラスタに格納された巨大なデータセットに対して様々なバッチ処理を実行することができる．共有環境を構築することで，大量データの複製や移動といった負担の大きい作業が不要となるため，様々なデータセットに対する処理を効率よく実行できるようになっている．

一方，分散並列処理基盤において，実行速度が異常に遅いタスクが突発的に発生することでジョブの完了が遅れる現象が観測されている [3], [4]．一般に分散並列処理では，利用者が投入する処理の集合（ジョブ）は，集合を構成している処理の単位（タスク）に分割される．分散並列処理基盤は自身を構成する計算機に対してタスクの実行を指示し，タスクの出力結果を集約したものが最終的なジョブの出力として得られる．ここで，ジョブの完了とは全てのタスクが完了することである事実に注意する．つまり，何らかの異常により終了しないタスクが存在すると，その異常タスクによってジョブの完了が遅延することを意味している．大規模な分散並列処理基盤においては異常タスクによるジョブ完了時間の遅延の影響は大きい．例えば，Microsoft 社の Bing 検索サービス用計算機クラスタは 2009 年から 2010 年において，数万台の計算機で構成され，一日当たり数百のジョブが実行されそれらのジョブによって数ペタバイトのファイルが処理されていたが，この環境では異常なタスクによってジョブ完了時間の中央値が 34%増加したと報告されている [3].

異常タスクによるジョブ完了時間の遅延を短縮する手法が研究されている．一般に異常タスクは，利用計算機資源の競合，ネットワークの輻輳，オペレーティングシステムやミドルウェアのバグや設定ミス，ハードウェア環境の不具合といった様々な要因で発生するため，発生時期を特定することは難しい．異常タスクによる処理時間の増加を抑制する技術のひとつとして Mantri [3] がある．Mantri では，タスクの実行中にその完了時間を推測し，他と比較して十分長いタスクに対して，再実行や他の計算機上での複数実行を行う．これにより異常タスクの影響が緩和され，処理時間の増加が抑えられるとされている．

ところが従来手法は，実行中のジョブの種類や計算機環境といったシステム状態が一定であることを仮定しているため，システム状態の変化に弱いといえる．既存手法では，タスクの推定完了時間を，実行済みであるタスクの完了時

[1] NEC クラウドシステム研究所
Kawasaki, Kanagawa 211–8666, Japan
[a)] m-asahara@bu.jp.nec.com

間や他の実行中タスクの推定完了時間と比較し，十分長いものを異常タスクとしている．ここで，利用する計算機資源が競合する他のジョブが実行されると，実行中であるタスクの処理速度は低下する．これは正常な処理速度の低下であり，これによって推定完了時間が増大することは許容されるべきである．しかし，既存手法では実行済みであるタスクの完了時間と比較した際に，推定完了時間が増加しているため，異常タスクであると誤って判定してしまう．こうして，異常タスクの判定機構が誤作動してしまうことにより，既存手法ではジョブの完了時間が不必要に増加してしまうことがある．

本論文ではシステム状態の変化にロバストな異常タスク特定手法を備えた MapReduce タスクスケジューラを提案する．提案手法はシステムの状態が変化する前の処理完了時間の影響を受けにくくするために，2 点の工夫を行う．

ひとつは，判定基準となるサンプルを取得する範囲を規定するタイムウィンドウによる過去情報のフィルタリングである．設定したタイムウィンドウ内に完了した実行済みタスクの完了時間と現在実行中である他のタスクの推定完了時間から，タスクが異常であるか否かを判定する．これにより，システム状態が変化する前の影響を緩和する．

もうひとつは，処理時間の確率分布モデルを推定する仕組みである．提案手法は処理時間の分布をポアソン分布であると仮定し，完了タスクの処理時間と実行中タスクの推定処理時間から分布のパラメータを推定する．そして，推定したパラメータから計算される確率から，異常タスクであるか否かを判定する．以上 2 点の工夫により，異常タスクの判定基準が現在のタスクの処理時間に対して適応的に変化できるようになるため，タスクの処理時間が正常に増加した場合に対する異常タスク判定機構の耐性が高まる．

提案手法の効果を測定するため，試作を Apache Hadoop [5] 2.0.3-alpha に実装し実験を行った．実験では，システム状態が変化する状況において，従来手法を模した異常タスク判定手法に対して提案手法の誤判定回数が 96.4%減少することを確認した．その結果，処理時間が 84.0%減少することを確認した．また，提案手法の効果により，異常タスクが 10%発生する状況において，オリジナルの Hadoop に対して処理時間が 41.5%削減されることを確認した．

本論文の構成は以下の通りである．2 章では，MapReduce 処理基盤において異常タスクが発生した状況の報告と，既存手法での問題点の考察を行う．3 章では提案手法について述べる．4 章では Hadoop に対して行った提案手法の実装について述べる．5 章では評価結果を報告する．6 章では関連研究について述べ，最後に 7 章で本論文をまとめる．

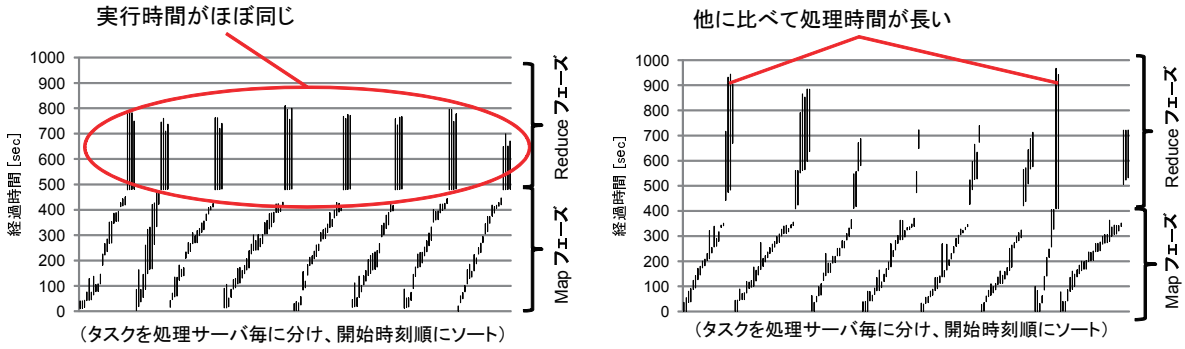
## 2. 背景

大規模分散並列処理基盤では，異常タスクの発生によってジョブの完了時間が増大するという事象が観測されている．一般に分散並列処理では，ジョブと呼ばれる処理の集合をタスクと呼ばれる並列実行可能な処理単位に分割し，それぞれのタスクを複数ある処理サーバが独立に実行する．そのため，全てのタスクが完了しなければジョブは完了しない．一方，分散並列処理ではタスクの実行速度が異常に低下する事象が観測されている [3], [4]．異常に速度が低下したタスクが発生すると，ジョブの完了が異常タスクによって遅延するため，ジョブの処理時間が増大する．

**図 1** は，MapReduce において異常タスクが発生しジョブの処理時間に影響を与えている様子を示す．図 1 (a) と (b) は同じ環境で同じジョブを実行した例であるが，図 1 (a) は正常時を，図 1 (b) は異常タスク発生時の観測結果を示している．縦軸が処理時間を秒単位で示しており，各直線はタスクの処理時間を表している．直線の始点は開始時刻を，終点は終了時刻をそれぞれ示す．それぞれのタスクは実行された処理サーバごとに分けられ，さらにタスクは開始時刻順にソートされている．この例では，特に Reduce フェーズにおいて差が大きい．図 1 (a) が示す正常な実行時では，Reduce フェーズのタスクの処理時間がほぼ一致していることがこの図より読み取れる．一方，図 1 (b) が示す異常タスク発生時では Reduce フェーズにおいて，処理時間が他のタスクに比べて長い異常タスクが発生している．また，異常タスクの処理時間が長いために，正常な実行時の例と比較してジョブの処理時間が長くなっている．図 1 (b) の例が示すような異常タスクの発生要因は，ネットワークの輻輳やディスクアクセスの競合といったハードウェアによるものや，ソフトウェアバグや不適切な設計といったものや，ユーザの設定ミスなど多岐にわたる．そのため，発生要因を解消することは難しい．

異常タスクによるジョブ完了時間の増大を抑制するための技術が提案されている．従来手法は，タスクの予測完了時間を計算し，これを同じ種類のタスクの処理完了時間や他のタスクの予測完了時間と比較することで，異常タスクを特定している．特定された異常タスクを再実行もしくはタスクを複製し複数の計算機で同時に処理を実行させることで，異常タスクによる影響を緩和する．

ところが，従来手法ではシステム状態が変化した場合に，正常なタスクを誤って異常と判定することがある．これは異常タスクの判定を，同じ種類のタスク全てと処理時間の分布の単純な比較を行っているためである．従来手法が誤判定を行う例を**図 2** に示す．この図は完了した T1 から T8 のタスクについて，記録時刻と 1 秒当たりの処理進捗率を示している．この例では，時刻 1:29:30 時点で新たな




(a) 正常な実行時の例　　(b) 異常タスク発生時の例

**図 1**　MapReduce における正常な実行時と異常タスク発生時の処理時間の比較．各直線はタスクの処理時間を表しており，始点が開始時刻，終点が終了時刻を示す．

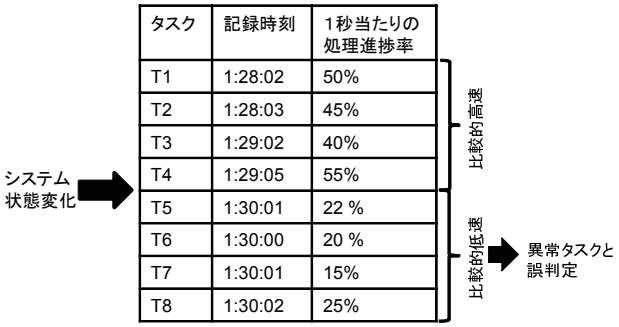



**図 2**　従来手法が正常タスクを異常タスクと誤判定する例

ジョブの実行が開始されたため，システムの状態が変化している．タスク T1 から T4 はシステム状態変化前に実行されたものであり，比較的高速に処理が行われている．一方で，タスク T5 から T8 はシステム状態変化後も処理が行われたため，処理速度は比較的低速である．従来手法ではタスクの処理速度は一定であると仮定しているため，この例ではタスク T5 から T8 を異常タスクと判定する．しかし，これらのタスクが低速であったのは新たなジョブの実行が開始されたためであるため，この速度低下は正常なものであると認識されなければならない．

特に実行されるジョブの種類が変化したり，利用状況に変動がある計算機クラスタでは，以上に示した従来手法の問題を解決する必要性が高いといえる．次章では，従来手法の問題を解決する異常タスクの判定手法について述べる．

## 3. 提案手法

提案手法は，直近に実行されたタスクにおける処理完了時間からタスク完了時間の確率分布を推定し，この確率分布を元に異常タスクの判定を行う．直近に実行されたタスクの特定は，タイムウィンドウを用いて過去情報をフィルタリングすることで実現する．タスク完了時間の確率分布の推定は，タスク完了時間のヒストグラムを計算し，これを用いてポアソン分布を仮定したパラメータ推定を行うことで実現する．

以下，提案手法の詳細について述べる．提案手法は，実行中のタスクについて定期的に異常タスクの判定を行い，異常タスクと判定されたタスクを再実行もしくは複製の実行を指示する．異常タスクの判定では，まず過去情報のフィルタリングを行い，次に確率分布の推定を行う．確率分布の推定後，異常判定を行うタスクの推定完了時間に対して，推定した確率分布から発生確率を計算する．算出した発生確率が一定以下の場合，提案手法は対象のタスクを異常であると判定し，再実行もしくは複製の実行を分散並列処理基盤に対して指示する．

過去情報のフィルタリングは，サンプリング対象範囲を限定するタイムウィンドウを用いて行う．タイムウィンドウは現在を起点にして過去方向に設定される．完了済みのタスクのうち，終了時刻がタイムウィンドウの範囲に含まれないタスクの記録は適宜消去される．一方，タイムウィンドウに含まれるタスクはサンプリング対象として採用し，確率分布の推定に用いる．また，現在実行中のタスクについてもサンプリング対象とする．

提案手法における過去情報のフィルタリングの動作を**図 3** を用いて説明する．あるジョブを構成する 6 つのタスク T1 から T6 が存在したとする．ジョブ開始を 0 秒とし，T1 は 8 秒時点で，T2 は 11 秒時点で，T3 は 32 秒時点で，T4 は 36 秒時点で完了したとする．一方，T5 と T6 は現在実行中であるとする．またタイムウィンドウの幅は 30 秒であったとし，現在時刻は 45 秒経過時点であるとする．この条件下で提案手法は，現在時刻である 45 秒時点から過去 30 秒の時点の範囲でタイムウィンドウを設定する．この例では，15 秒時点から 45 秒時点がタイムウィンドウの範囲となる．T1 と T2 はタイムウィンドウに含まれないため，サンプリング対象から除外される．一方でタイムウィンドウに含まれる T3 と T4 はサンプリング対象とする．また，T5 と T6 は実行中であるため，これらもサンプリング対象とする．

タイムウィンドウの幅は提案手法のシステム状態変化へのロバスト性と異常タスク判定の精度に影響する．タイム


| タスク | 記録時刻 | 1秒当たりの処理進捗率 |
|---|---|---|
| T1 | 8 秒 | 50 % |
| T2 | 11 秒 | 45 % |
| T3 | 32 秒 | 20 % |
| T4 | 36 秒 | 15 % |
| T5 | 45 秒（実行中） | 5 % |
| T6 | 45 秒（実行中） | 20 % |

**図 3** 過去情報のフィルタリング例

ウィンドウの幅を小さくするとシステム状態の変化に対するロバスト性は向上するが，異常タスクの判定の精度は悪化する．これは，サンプルにおける異常タスクの割合が大きくなることが原因である．例えば現在実行中のタスクの多くが異常であった場合，サンプルにおける異常タスクの割合が上昇する．すると，異常タスク同士の比較となってしまうため，異常タスクを特徴付ける指標の差が認められなくなってしまう．一方で，タイムウィンドウの幅を大きくすると，サンプルにおける正常タスクの割合が大きくなるため異常タスクを判定しやすくなるが，過去の情報を長く記憶するためロバスト性は低下する．システム状態変化へのロバスト性にタイムウィンドウの幅が与える影響は 5 章で評価する．タイムウィンドウの幅をシステム状態変化に応じて動的に変更するという方法も考えられるが，これに関しては今後の課題とする．

フィルタリングの実施後，提案手法はフィルタリングされたタスク完了時間情報と，現在実行中であるタスクの推定完了時間から，タスク完了時間の確率分布を推定する．提案手法はタスク完了時間の分布をポアソン分布と仮定して，母集団の分布を推定する．

ポアソン分布は，時間または空間における発生間隔が指数分布となる事象を説明する分布モデルである．これを，ある単位時間に実行されたタスクの処理時間の分布を説明するために用いる．パラメータ $\lambda > 0$ のポアソン分布の確率は次式

$$P(\lambda, X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{1}$$

で与えられる．ただし，$X$ は確率変数である．また，$\lambda$ は予め与えられているとする．$X$ をある単位時間に実行されたタスクの処理時間で与えると，その処理時間となるタスクの発生確率 $P$ が得られる．

最後に，提案手法は推定した確率分布 $P(\lambda, X = k)$ を用いて，実行中のタスクの推定完了時間に対する発生確率を算出する．発生確率が一定以下のタスクに対して異常であると判定し，再実行もしくは複製の実行を行う．

以下，提案手法が実行する異常タスクの判定アルゴリズムを詳細に説明する．提案手法が実行する異常タスクの判定アルゴリズムを Algorithm 1 に示す．まず提案手法は，フィルタリングされたタスク管理時間情報から，タスク速度のヒストグラムを作成する．ヒストグラムは各ビンにおけるデータの値の範囲を均一にする，いわゆる EQUI-WIDTH の方法で生成する．次に提案手法は，生成したヒストグラムから最も頻度の高いビン番号 $k\_mode$ を得る．

次に提案手法は，ポアソン分布 $P(\lambda, X = k')$ の分布情報を出力する．ただし，

$$k' = \max(k - k\_mode, 0) \tag{2}$$

とする．これは，処理時間が著しく増加しているタスクのみを異常タスクとみなすようにするための変換である．最頻値をポアソン分布関数の凸部分と対応させるようにポアソン分布のパラメータ $\lambda, k$ を決定すると，処理時間が短いタスクについても確率が小さくなるため判定アルゴリズムは異常とみなすよう動作する．しかし，処理時間が短いタスクについては，処理が早く進んでいるためシステムとしては好ましい状態である．従って，提案手法では処理時間が短いタスクは正常とみなすべきである．上記に示したポアソン分布の分布関数の変換は，処理時間が短いタスクを正常とみなすための処理となる．

次に提案手法は，判定対象となるタスク $T$ に対して，判定処理を実施する．まずタスク $T$ の推定処理完了時間 $t'$ を求める．推定処理完了時間 $t'$ は，タスク $T$ の現在までの実行時間 $t$ とタスクの進捗率 $r$ から次式

$$t' = \frac{t}{r} \tag{3}$$

を用いて求める．提案手法はタスクの進捗率 $r$ の算出方法に依存しない．例えば，ユーザプログラムによって定義された進捗率でも良いし，タスクの入力データ量に対する処理完了データ量の比で表しても良い．ただし，提案手法は進捗率 $r$ の算出精度には依存するため，進捗率 $r$ の算出方法の精度は高いことが望ましい．

次に提案手法は，タスクの推定完了時間 $t'$ からビン番号 $k$ を求める．ビン番号 $k$ の算出は，ヒストグラムを生成する際に使用したビンの幅の決定方法を用いて行う．続いて，変換後のポアソン分布関数 $P(\lambda, X = k')$ に対応するように，提案手法はビン番号 $k$ の $k'$ への変換を式 2 を用いて行う．そして，提案手法は $P(\lambda, X = k')$ を求め，$P$ が予め与えられた閾値 $P'$ より小さい場合，タスク $T$ を異常タスクであると判定する．

異常タスク判定アルゴリズムの動作例を**図 4** を用いて説明する．フィルタリングされた後のタスク完了時間情報に含まれるタスクがタスク T1〜T6 であったとする．タスク T1 と T2 は完了済みであり，タスク T3〜T6 は実行中であるとする．タスク T1〜T6 の完了時間と推定完了時間はそれぞれ，5 秒，4 秒，20 秒，5 秒，7 秒，4 秒であったとす

**Algorithm 1** 異常タスク判定アルゴリズム

1: フィルタリングされたタスク完了時間情報から，タスク速度のヒストグラムを生成する
2: ヒストグラムから最頻値のビン番号 $k\_mode$ を求める
3: ポアソン分布 $P(\lambda, X = k')$（ただし $k' = \max(k - k\_mode, 0)$）の分布情報を出力する
4: **for all** 判定するタスク $T$ について **do**
5: 　タスク $T$ の推定処理完了時間 $t'$ を求める
6: 　タスク推定完了時間 $t'$ からビン番号 $k$ を求める
7: 　$k' = \max(k - k\_mode, 0)$ を求める
8: 　**if** $P(\lambda, X = k') < P'$ **then**
9: 　　タスク $T$ を異常タスクとする
10: 　**end if**
11: **end for**

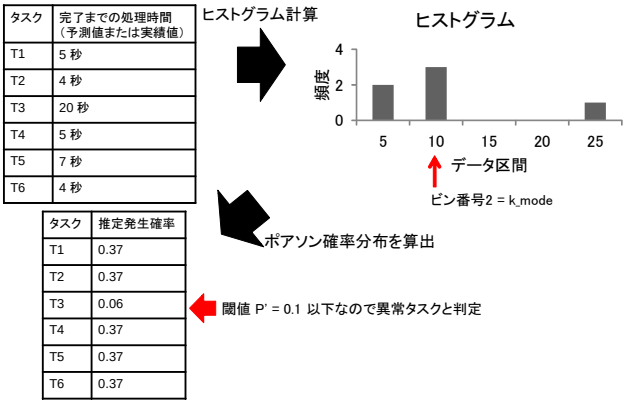



**図 4**　異常タスク判定アルゴリズムの動作例

る．また，ポアソン分布の算出に用いるヒストグラムのビンの幅を 5 秒，パラメタ $\lambda$ を 1，異常タスクと判定する閾値の確率 $P'$ を 0.1 とする．

提案手法は，タスク T1〜T6 を標本とし，まずヒストグラムを作成する．タスク完了時間の範囲が $[0, 5)$ のビンをビン番号 1 とし，タスク T2 と T6 を入れる．すると，番号 1 のビンの頻度は 2 となる．同様に，範囲が $[5, 10)$ のビンはビン番号 2 となり，番号 2 のビンはタスク T1，T4，T5 を入れるため頻度は 3 になり，範囲が $[20, 25)$ のビンはビン番号 5 となり，番号 5 のビンはタスク T3 を入れるため頻度は 1 になる．生成したヒストグラムにおいて，最も頻度の高いビンの番号は 2 であるため，$k\_mode = 2$ とする．

次に提案手法は，ポアソン分布の関数式 $P(\lambda = 1, k')$ を求める．ただし，$k' = \max(k - 2, 0)$ とする．提案手法はこのポアソン分布の関数式から，実行中のタスク T3〜T6 について異常タスクの判定を行う．タスク T3 の予測完了時間は 20 秒であるため，ビン番号 $k$ は 5 である．よって，$k' = \max(5 - 2, 0) = 3$ である．したがって，タスク T3 の予測完了時間の発生確率は $P(1, 3) \simeq 0.06$ となる．$P < P' = 0.1$ であるため，提案手法はタスク T3 を異常であると判定する．一方，タスク T2〜T4 の予測完了時間の発生確率はいずれも約 0.37 であり，いずれも $P'$ より大きい．よって，タスク T2〜T4 は正常であるとみなす．

タスクの進捗速度はタスクの実行段階によって変動するため，提案手法では進捗速度の揺らぎに対する工夫を行っている．例えば，タスクの実行開始直後は初期化を行うため，進捗速度は極めて小さいことがある．また，処理負荷が大きくなるデータを処理する期間は一時的にタスクの進捗度が低下する．提案手法はこのような進捗速度の正常な変化に対応する工夫を 2 点行っている．まず，提案手法はタスクの完了時間を推定する際に，過去数個の推定値から最低値を用いる．また提案手法は，連続判定回数が閾値を超えた場合のみ，異常タスクと判定したタスクについて実際に再実行もしくは複製の実行を行う．これらの工夫により，処理速度の揺らぎに対する誤判定の確率を下げている．

## 4. 実装

提案手法を導入した MapReduce タスクスケジューラを Apache Hadoop 2.0.3-alpha に実装した．以下に実装の詳細を述べる．

提案手法は Hadoop MapReduce フレームワークとは独立したライブラリとして実装し，MapReduce フレームワークから適宜呼び出される形とした．MapReduce の ApplicationMaster を構成する `MRAppMaster` クラスと `RMContainerAllocator` クラスを改変し，クラスタマシンから送信されるハートビートを受信するタイミングで異常タスクの判定ルーチンを呼び出して異常タスクの判定を行うようにした．提案手法はハートビートで送信されたタスクの進捗情報を使用して完了時間を推定するよう実装した．

試作ではタスクの再実行機構のみを実装した．タスクの再実行機構は，異常と判定されたタスクに対して `TaskAttemptEventType.TA_TIMED_OUT` イベントを送信することで，タイムアウトイベントを故意に発生させ，タスクを異常終了させる．この仕組みによって MapReduce フレームワークの再実行動作を呼び出すことで，異常タスクの再実行を実現する．タスクの再実行によって `fail` タスクが増加することで MapReduce フレームワーク側が誤作動しないように，ジョブを失敗とみなす `fail` タスク数の閾値を十分大きくした．実験では 9999 としている．

## 5. 評価

提案手法の評価を行うために実験を実施した．評価の目的は 3 点である．1 点目は，システム状態に対する提案手法のロバスト性を確認することである．2 点目は，異常タスク発生時における提案手法の処理時間短縮効果を確認することである．3 点目は，提案手法のオーバヘッドの計測である．

### 5.1 実験環境

評価には GridMix [6] のバージョン 2（以下，GridMix2）を，Hadoop 2.0.3-alpha で動作するように改

変して使用した．ワークロードとして GridMix2 の `GridmixMonsterQuery.large` ジョブを，Map と Reduce 共に出力データを非圧縮とする設定で実行した．入力データは GridMix2 の入力データ生成ツールで生成し，そのサイズを 407.9 GB とした．Reduce の数は 370 とした．

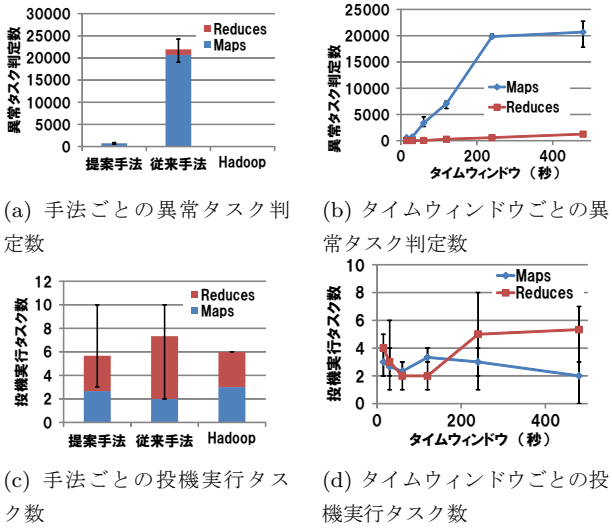
実験に使用した Hadoop クラスタは以下の通りである．Hadoop のバージョンは 2.0.3-alpha で，1 台のマスタサーバと 30 台のスレーブサーバで構成されている．マスタサーバとスレーブサーバは共に NEC Express5800/R120a-1 である．構成は，CPU が 4 コアを備えた Intel Xeon E5504 2.0 GHz，メモリが 12 GB，HDD が RAID-1 構成の 2.5 インチ 10k rpm SAS 300GB である．各サーバは NEC IP8800/S3640-48T2XW Gigabit Ethernet スイッチで接続されている．OS は Ubuntu 12.10 (64bit Linux 3.5.0) を使用した．

Hadoop の設定は以下の通りである．HDFS のチャンクサイズは 256 MB とした．MapReduce の最大タスク失敗回数と再実行回数は共に 9999 とした．Reduce タスクの割り当て開始は全ての Map タスクが完了した後とした．投機実行は有効とした．`shuffle` 時の同時コピー数を 50，`sort` 時に使用するメモリサイズを 512 MB，ディスクにマージするセグメント数を 100 とした．YARN のリソーススケジューラは `FairScheduler` とした．各タスクに割り当てるメモリ量は，Map タスクと Reduce タスクでそれぞれ 1.5 GB，3 GB とし，JVM 実行時のヒープサイズはそれぞれ 1 GB，2.5 GB とした．

提案手法の設定は，特段の記載がない場合は以下の通りである．サンプルのフィルタリングに用いるタイムウィンドウのサイズは 30 秒間とした．作成するヒストグラムのビン幅は 15 秒とした．異常タスクと判定する確率の閾値は 0.05 とした．予測完了時間の計算では，過去 5 回の計算結果から最小値を用いることとした．また，タスク開始から 60 秒を経過しても進捗率が 0 であるタスクは異常タスクとすることとした．また以下では，タイムウィンドウの幅を 480 秒としたものを，システムの状態変化に関わらず異常タスクの判定を行う従来手法と位置づけて議論を行う．

### 5.2 システム状態の変化に対するロバスト性

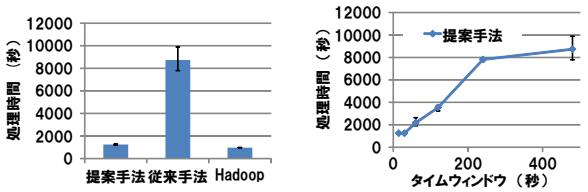
システム状態の変化に対する提案手法のロバスト性を確認するために，システム状態が変化する環境を用意し，その上で提案手法のタイムウィンドウを変化させる実験を実施した．システム状態の変化として，ジョブの実行中に全スレーブサーバの性能が一律に低下した状況を再現した．全スレーブサーバの性能を一律に低下させたため，全てのタスクにおいて処理速度が低下する．この処理速度の低下は正常であるとみなされるべきである．システム状態の変化による性能低下は，Linux control groups (cgroups) の cpuset サブシステムを使用して，ジョブ開始時点から 90 秒後に使用可能 CPU コア数を 4 から 1 に変更することで実現した．




(a) 手法ごとの異常タスク判定数　(b) タイムウィンドウごとの異常タスク判定数

(c) 手法ごとの投機実行タスク数　(d) タイムウィンドウごとの投機実行タスク数

**図 5** システム状態変化時における異常タスク判定数と投機実行タスク数




(a) 手法ごとの処理時間　(b) タイムウィンドウごとの処理時間

**図 6** 性能低下時における，タイムウィンドウの幅と処理時間の関係

実験では，タイムウィンドウの幅を 30 秒，60 秒，120 秒，240 秒，480 秒とし，それぞれについて異常タスクと判定したタスク数と，ジョブの処理時間を 3 回計測し，それぞれの平均値と最小値，最大値を算出した．タイムウィンドウが 480 秒の場合は従来手法としても扱う．また投機実行が行われたタスク数についても同様に計測した．さらに，比較のためにオリジナルの Hadoop についても計測した．

**図 5** に，システム状態変化時における異常タスク判定数と投機実行タスク数を示す．実験結果は，提案手法が従来手法に対して異常タスクの判定数が 96.4%削減したことを示している．この結果から，従来手法はシステム状態の変化によるタスク処理速度の正常な低下を異常とみなすが，提案手法はこの誤判定の回数が削減されたことが示された．

また実験結果は，タイムウィンドウの幅の増加に従って異常タスク判定数が増加することを示している．特に Map フェーズにおいて，タイムウィンドウの幅が 480 秒であった時の Map タスク以外については，タイムウィンドウの幅に対して異常と判定された Map タスク数がほぼ線型に増加した．例えば，タイムウィンドウの幅が 30 秒の場合 Map タスクの異常判定数が平均 716 であったのに対して，240 秒の場合は平均 19,825 であった．これは，正常なタス

クを異常と誤判定する時間がタイムウィンドウの幅に対して線型に増加するためである．一方，タイムウィンドウの幅が 480 秒の場合は，240 秒の場合に対して異常タスク判定数の増加比が減少した．タイムウィンドウの幅が 480 秒において異常と判定された Map タスク数は平均 20,683 であり，240 秒の場合の約 1.04 倍であった．この原因については今後詳細に検証する．

また，Reduce タスクについても，タイムウィンドウの幅に対して異常タスク判定数が線型に増加した．例えば，タイムウィンドウの幅が 30 秒の場合 Reduce タスクの異常判定数が平均 9 であったのに対して，480 秒の場合は平均 1,258 であった．Reduce タスクが動作する期間ではシステム性能は変化していないため，`GridmixMonsterQuery.large` ジョブにおける Reduce フェーズ特有の影響があったと考えられる．仮説として，Reduce タスクは各キーに対応付けられた値の集約処理を行うため，Reduce タスクごとの処理時間の差が大きいことが原因と考えている．タイムウィンドウが大きい場合，完了タスクの処理時間を長く記憶してしまうため，処理時間が短いタスクを確率分布を計算するサンプルとして多く含んでしまうことになる．これが原因で Reduce タスクの異常判定数が増加したと考えている．この現象については今後詳細に調査する予定である．

また，投機実行タスク数については，Hadoop，従来手法，提案手法のどのパターンでも 5〜8 程度で変わらなかった．これは，どのサーバも一律に性能が低下したために，実行が遅れたタスクの数にあまり変化がなかったためであると考えられる．

**図 6** に，システム状態変化時における処理時間を示す．実験結果によると，提案手法が従来手法と比較して処理時間を 84.0%削減したことが示された．また実験結果は，タイムウィンドウの幅の増加に従って処理時間が増加することを示している．特に，タイムウィンドウの幅が 240 秒まで増加する間は，タイムウィンドウの幅に対して処理時間が線型に増加した．例えば，タイムウィンドウの幅が 30 秒の場合の処理時間が平均 1,248 秒であったのに対して，240 秒の場合は平均 7,823 秒であった．これは，Map フェーズにおける異常タスク判定数の増加と傾向が一致するため，異常と誤判定された Map タスクの影響であると考えられる．

一方，タイムウィンドウの幅が 480 秒の場合は，240 秒以下に対して増加比が減少した．480 秒の場合の平均処理時間は 8,741 秒であり，240 秒の場合に対する増加比は約 1.12 倍であった．処理時間の分散を調べると，他の設定と比較して 2 倍から 6 倍に増加していることが確認できた．このことから，480 秒設定時は処理時間の揺らぎが大きくなり，比較的短時間で処理が完了している場合が発生していると考えられる．この現象については今後詳細に調査する予定である．

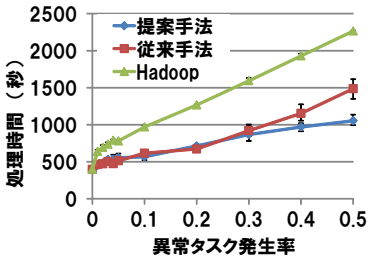
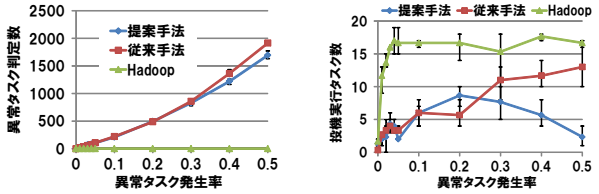



**図 7** 異常タスクの発生率と処理時間の関係




(a) 異常タスク判定数　(b) 投機実行タスク数

**図 8** 異常タスクの発生率と異常判定数，投機実行タスク数の関係

### 5.3 異常タスク発生時における処理時間短縮効果

異常タスク発生時における提案手法の処理時間削減効果を確認するために，一定の比率で異常タスクが発生する環境を用意し，処理時間，異常タスク判定数，投機実行タスク数を計測した．実験では，オリジナルの Hadoop と提案手法のそれぞれで 3 回計測し，それぞれの平均値と最小値，最大値を算出した．異常タスクの発生環境は Hadoop を改変することで実現した．改変版 Hadoop では，Map タスクと Reduce タスクのそれぞれにおいて指定した確率でタスクを選択し，その処理を遅延させた．具体的には，タスクにおけるキーと値のペアについて `map/reduce` メソッドを呼び出すループを改変し，異常タスクについてはループ 1 回毎に 1 ミリ秒遅延させるようにした．

**図 7** に，異常タスクの発生率と処理時間の関係を示す．提案手法の処理時間はオリジナルの Hadoop に対して短かった．異常タスクの発生比率が 0.1 の場合，オリジナルの Hadoop に対して処理時間が 41.5%削減された．また，異常タスクの発生比率が 0.5 の場合では，オリジナルの Hadoop に対して処理時間が 53.5%削減された．これは，提案手法が異常タスクを正しく検出しているために，処理時間が削減されたことを示している．

また，異常タスクの発生率の増加に対する処理時間の増加が，オリジナルの Hadoop と比較して提案手法では抑えられた．異常タスクの発生比率が 0.1 から 0.5 に増加した際，オリジナルの Hadoop では処理時間が 2.3 倍に増加したのに対して，提案手法では 1.9 倍に抑えられた．これは，異常タスクの発生率が増加しても，提案手法が正しく異常タスクを検出し，処理時間の増加を抑制していることを示している．

**図 8** に，異常タスクの発生率と異常タスク判定数，投機実行タスク数の関係を示す．この実験結果は，提案手法

は異常タスクの発生率の増加に応じて異常タスク判定数を増加させていることを表している．例えば，異常タスクの発生率が 0.1 の異常タスク判定数が平均 169 であったのに対して，発生率 0.5 の異常タスク判定数は 1602 と，約 9.5 倍に増加した．異常タスク判定数と発生率の増加比率が異なっているが，これは再実行したタスクからも異常タスクが発生することが原因である．

一方，オリジナルの Hadoop では，投機実行のタスク数が提案手法の場合と比較して増加している．実験では，異常タスクの発生率 0.1 から 0.5 で，平均 14 から 16 程度のタスクが投機実行されていた．これは，ジョブ終了直前まで残っていた異常タスクについて Hadoop が投機実行を行うためである．異常タスクの発生率に対して投機実行タスク数の変化が小さいのは，投機実行がジョブ終了直前の空き資源発生時のみでしか機能しないためである．また，提案手法と比較して処理時間の削減に寄与していない理由も同じである．投機実行は空き資源が発生し，余剰にタスク割り当てが可能である条件のみでしか機能しない．ジョブ実行の大半では割り当て可能なタスクは十分残っているため，空き資源は生じない．よって，投機実行は異常タスク発生時にはあまり機能しなかったと考えられる．これに対して，提案手法は積極的に異常タスクを検出し再実行するため，処理時間の削減に貢献したと考えられる．

また本実験結果は，提案手法は異常タスクの発生率が増加すると従来手法よりも処理時間が短縮されることを示している．これは，従来手法が過去の誤判定の履歴を長く記録することで，提案手法よりもさらに誤判定の頻度が上昇したためと考えられる．実際，図 8 (a) が示すように，異常タスクの発生率の上昇とともに，従来手法の異常タスク判定数は提案手法よりも増加している．

### 5.4 オーバヘッド

提案手法のオーバヘッドを確認するため，オリジナルの Hadoop との処理時間の比較を行った．本実験では，システム状態の変化がある場合とない場合との 2 つの条件において，オリジナルの Hadoop と提案手法を導入した Hadoop のそれぞれで処理時間を計測した．システム状態の変化は 5.2 節と同じ条件で発生させた．提案手法のタイムウィンドウの幅は 30 秒である．なお，異常タスクは発生させていない．

**表 1** に，上記のパターンにおける処理時間の平均値を示す．システム状態が変化した場合のオーバヘッドは 28.3%であった．これは，5.2 節での実験結果が示したように，提案手法が平均 716 の誤判定を行ったためである．5.2 節の実験結果が示すように，従来手法に対して提案手法は誤判定の回数を削減している．しかし，より一層のオーバヘッド削減を行うため，さらなる誤判定回数の削減を今後の課題とする．

**表 1** オーバヘッドの比較

| | Hadoop | 提案手法 | 増加率 |
|---|---|---|---|
| システム状態変化あり | 973 秒 | 1248 秒 | 28.3% |
| システム状態変化なし | 398 秒 | 394 秒 | −0.01% |

一方，システム状態が変化しない場合での提案手法のオーバヘッドは −0.01%であった．この結果は，システムの状態が変化していない条件では提案手法が正確に異常タスクを判定できていることを示している．

## 6. 関連研究

異常タスクによる処理時間の増加を抑制する手法の研究がなされている [1], [2], [3], [4], [7]．オリジナルの MapReduce 処理基盤 [1] は投機実行と呼ばれる仕組みを備えている．投機実行とは，ジョブ実行の終盤などで処理サーバに割り当てるタスクが少なくなった際に，未終了であるタスクについて，その複製タスクを別の処理サーバで追加実行する仕組みである．これにより，何らかの異常により実行が遅れているタスクの終了を早めることができる．しかし，投機実行は異常タスクを積極的に特定することはしないため，処理時間の改善効果はジョブ実行の終盤といった，処理サーバへのタスク割り当てに空きがある状況のみでしか機能しない．

Mantri [3] は，実行中のタスクの処理時間を推定し，推定処理時間が他の実行中タスクや実行済みのタスクよりも十分長い場合，そのタスクを異常とみなして再実行もしくは複製の実行を行う．Mantri では異常タスクの判定基準として，MapReduce ジョブにおける Map フェーズや Reduce フェーズといった実行フェーズを考慮し，同じ実行フェーズであるタスクの処理時間の分布を用いる．したがって，同じ実行フェーズにおいて同時実行するジョブの種類やシステム構成など，システムの状態が変化した場合は正常なタスクを異常と誤って判定する可能性がある．提案手法は同じ実行フェーズにおいても過去の処理時間情報を除外するため，システム状態が変化する以前の影響が緩和され，判定の精度が向上する．

Dolly [4] は，応答性が要求される比較的小規模なジョブに対してジョブの複製を行い，オリジナルのジョブと同時に複製の実行を行う．これにより，異常タスクによるジョブの処理時間の増加を，タスクの再実行や投機実行を行うことなく抑えることができる．Dolly は大規模なジョブに対しては，ジョブの複製によるリソース使用量が増加するため，適用が難しい．これに対し，提案手法は大規模なジョブに対しても，リソース使用量の増加を抑えながら異常タスクによる処理時間の増加を抑制する．また，提案手法を大規模なジョブに対して適用し，Dolly を小規模なジョブに対して適用するといった，補完する形での運用も選択肢として存在する．

異常タスクは実行時における動的な要因で発生する以外に，処理内容と入力データの特性によって発生する場合がある．Ibrahim ら [8] と Gufler ら [9] は MapReduce の Reduce フェーズにおける処理データの分割の偏りによる処理時間の増加を抑制する手法を提案している．これらの手法ではコストモデルを生成し，そのモデルにおいてコストを最小にするように Reduce キーの割り当てを Reduce タスクに行う．SkewTune [10] は処理時間が長いタスクを動的に検出し，そのタスクを再分割し処理サーバに割り当て直すことで処理時間を短縮する．RoPE [11] はプログラムの解析を行い，割り当てデータの偏りが発生する箇所を修正することで処理時間を短縮する．これらの手法はデータやプログラムの構成といった，静的な要因で発生する処理時間の長いタスクへの対処方法である．したがって，提案手法とは補完する関係にあり，併用可能であるといえる．

## 7. おわりに

本論文ではシステム状態の変化にロバストな異常タスク特定手法を備えた MapReduce タスクスケジューラを提案した．提案手法はシステムの状態が変化する前の処理完了時間の影響を受けにくくするために，2 点の工夫を行った．ひとつは，判定基準となるサンプルを取得する範囲を規定するタイムウィンドウによる過去情報のフィルタリングである．もうひとつは，処理時間の確率分布モデルをポアソン分布を仮定して推定し，分布と矛盾するタスクを異常と判定する仕組みである．実験では，システム状態が変化する状況において，従来手法を模した異常タスク判定手法に対して提案手法の誤判定回数が 96.4%減少することを確認した．その結果，処理時間が 84.0%減少することを確認した．今後は，提案手法のオーバヘッドを削減するために，異常タスクの判定精度を向上させる工夫を行う予定である．また，実行中のジョブの種類が変わるといった，より実践的な条件での評価を実施する予定である．

### 参考文献

[1] Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *Proc. of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 137–149 (2004).
[2] Isard, M., Budiu, M., Yu, Y., Birrell, A. and Fetterly, D.: Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks, *Proc. of ACM European Conf. on Computer Systems (EuroSys)*, pp. 59 – 72 (2007).
[3] Ananthanarayanan, G., Kandula, S., Greenberg, A., Stoica, I., Lu, Y., Saha, B. and Harris, E.: Reining in the Outliers in Map-Reduce Clusters using Mantri, *Proc. of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 265–278 (2010).
[4] Ananthanarayanan, G., Ghodsi, A., Shenker, S. and Stoica, I.: Effective Straggler Mitigation: Attack of the Clones, *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 185–198 (2013).
[5] Apache: Apache Hadoop, `http://hadoop.apache.org/`.
[6] Apache: GridMix, `http://hadoop.apache.org/docs/stable/gridmix.html`.
[7] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. and Stoica, I.: Improving MapReduce Performance in Heterogeneous Environments, *Proc. of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 29–42 (2008).
[8] Ibrahim, S., Jin, H., Lu, L., Wu, S., He, B. and Qi, L.: LEEN: Locality/Fairness-Aware Key Partitioning for MapReduce in the Cloud, *Proc. of IEEE Int'l Conf. on Cloud Computing Technology and Science (CloudCom)*, pp. 17–24 (2010).
[9] Gufler, B., Augsten, N., Reiser, A. and Kemper, A.: Handling Data Skew in MapReduce, *Proc. of Int'l Conf. on Cloud Computing and Services Science (CLOSER)*, pp. 574–583 (2011).
[10] Kwon, Y., Balazinska, M., Howe, B. and Rolia, J.: SkewTune: Mitigating Skew in MapReduce Applications, *Proc. of ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD)*, pp. 25–36 (2012).
[11] Agarwal, S., Kandula, S., Bruno, N., Wu, M.-c., Stoica, I. and Zhou, J.: Re-optimizing Data-Parallel Computing, *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2012).