クラウドコンピューティング環境でのマルチコアプロセッサの停止故障を考慮したタスクスケジューリング

後藤田 祥平[†1] 柴田 直樹[†2]
山内 由紀子[†3] 伊藤 実[†1]

本稿では，データセンタの計算機の多くがマルチコアプロセッサを搭載している環境において，停止故障を考慮したタスクスケジューリング手法を提案する．提案手法では，ネットワークコンテンションを考慮し，停止故障発生時には回復処理を行った上でのタスク終了までの時間を目標実行終了時間内に抑えた上で，停止故障が発生しない場合のタスク処理時間を最小化する．本手法を実現するため，ネットワークコンテンションを考慮した既存のタスクスケジューリングを拡張した．提案手法をシミュレーションにより既存手法と比較し，本手法の有効性を示す．実験の結果，停止故障発生時のレイテンシを含む処理時間を既存手法の約半分に減少させることができ，停止故障非発生時の処理時間増加もわずかにとどまることが確認できた．

Multi-core aware task scheduling considering single fault of computing node in cloud computing

Shohei Gotoda,[†1] NAOKI SHIBATA,[†2]
YUKIKO YAMAUCHI[†3] and MINORU ITO[†1]

In this paper, we propose a fault tolerant task scheduling algorithm for a single fail-stop failure in data centers with multi-core processors. In the proposed method, network contention is taken into consideration and the task execution time is minimized when there is no failure, while guaranteeing smaller task execution time than the target time when there is a single failure. We compared the proposed method with an existing method to show the effectiveness of our method with simulations. We confirmed that our method reduced the execution time by half in case of failure, while there is almost no degradation of execution time in case of no failure.

1. はじめに

近年，クラウドコンピューティングが注目を集めており，データセンタをはじめとする多数の計算機が接続された環境で分散システムが利用されている．クラウドコンピューティングとは，社会インフラとして利用できる状態にある分散処理システムである．特に，Google Docs や Google App Engine，Microsoft Windows Azure，Amazon EC2，VPS のような計算機仮想化技術等として分散システムが実用化されてきており，クラウドコンピューティングは社会の重要なインフラとなっている．クラウドコンピューティングに信頼性を持たせ，また実用上十分な応答速度を実現するためには，ネットワーク上でのデータ転送を制御する仕組みや計算機の停止故障に備える仕組みが必要となる．一方で，近年高いパフォーマンスを持つプロセッサはいずれもマルチコアプロセッサとして設計されており，マルチコアプロセッサの利用を同時に考慮する必要がある．マルチコアプロセッサとは 1 プロセッサに複数コアを搭載しており，それぞれのコアで別処理を並列して行うことができるプロセッサである．また，コア間の通信はコア内で行われるため，通常のネットワークリンクと比較して非常に高速である．

計算機にタスクを割り当てる処理としてタスクスケジューリングという手法があるが，ネットワークでのデータ転送によるパケットの衝突や計算機の停止故障による計算の中断が発生し，一定時間内に処理結果を出力しなければならない条件下で，マルチコアプロセッサが多数を占める計算機環境を想定したタスクの処理時間を効率化する手法は提案されていない．また，既存のシングルコアプロセッサ向けのタスクスケジューリング手法ではマルチコアプロセッサのコア間の通信が非常に高速で，1 マルチコアプロセッサにタスクが集中してしまう問題がある．

本稿では，クラウドコンピューティング環境において，ネットワーク上での制約を考慮しつつプロセッサの停止故障が発生した場合でも，与えられた実行終了目標時間内で処理が完了するためのレイテンシ制約を満たし，停止故障が発生しない場合にタスク処理時間を最小化するようなタスクスケジューリング手法を提案する．

†1 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
†2 滋賀大学
Shiga University
†3 九州大学
Kyushu University

また，提案手法と既存手法をシミュレーションを用いて比較を行い，その効果を確かめる．タスクスケジューリングは手法により割り当て方法が様々であるが，ここでのタスクスケジューリングとは閉路を持たない有向グラフ，すなわち非循環有向グラフ (Directed Acyclic Graph, 略して DAG) を用いて，タスクを分類し，複数のコアを持つ各プロセッサに割り当てる手法である．タスクスケジューリングには Oliver[1] らのネットワーク上でのデータ転送によるパケットの衝突を考慮した手法を拡張して用いる．提案手法では停止故障が発生した場合，マルチコアプロセッサでのタスク割り当ての偏りを防ぐため，停止故障が発生した場合の影響が大きいタスクを，別プロセッサに割り当てる処理が含まれている．本稿では，停止故障が発生した場合の回復時間を含め，既存手法との比較を行った．その結果，与えるタスクの大きさを変化させる実験では，停止故障発生時の処理時間が約半分になった．また，停止故障非発生時の処理時間の増加もわずかにとどまることを確認できた．プロセッサ数を変化させる実験では，既存手法の処理時間に比べ提案手法は約 0.7 倍の時間で処理できることを確認した．

以降 2 章では，関連研究について述べ，提案手法の位置づけを明確にし，3 章ではタスクスケジューリングおよび諸定義について述べる．4 章では，問題を定式化した後，ネットワーク，マルチコアプロセッサ，停止故障を考慮したタスクスケジューリングでのタスク処理時間を最小化するためのタスクスケジュールを決定するアルゴリズムを述べる．5 章では，既存手法と提案手法の性能をシミュレータで評価し考察を行う．最後に 6 章でまとめを述べる．

2. 関 連 研 究

タスクをプロセッサに割り当てる手法はタスクスケジューリングと呼ばれ多くの既存研究が存在する．特に近年マルチコアプロセッサの普及が著しい他，ネットワーク帯域の圧迫などの問題もあり，様々な問題を含んだ環境でのタスクスケジューリング手法が提案されている[7)8)9)10)]．また，古典的なタスクスケジューリングではネットワークコンテンションを考慮されていないため，ネットワークの利用が遅延のない理想環境であるため，本稿での手法としては不適である．

Oliver らはネットワーク帯域やネットワークコンテンションが生じない理想環境でのリストスケジューリングを拡張し，ネットワーク状況を考慮したスケジューリング手法の提案した[1]．文献中の実験では，実環境を用いてタスクスケジューリングを実行しており，ネットワーク状況を考慮した提案手法により，理想環境を想定したタスクスケジューリング手法と比較して，スケジュールの正確性や処理効率の上昇が述べられている．しかし，計算機に故障が発生することは想定されていない．

Wolf らはタスクスケジューリングにおいてタスクの種類に合わせていくつかのテンプレートを用意し，入力されたタスクに合わせたグラフテンプレートを選択するタスクスケジューリングである[3]．システムが高負荷であるときの効率的なタスクスケジューリングを提供するものであり，データレートや CPU 使用率を考慮した．しかし，ネットワークコンテンションに関しては考慮されていない．

Gu らは High Availability システムでのストリームデータ処理環境で，計算機の停止故障が発生することを想定しており，ジョブの処理データを回復させる手段としてチェックポイントに関する手法を提案した[2]．停止故障とはプロセッサ自体の機能が停止することである．Gu らの手法では，特にチェックポイントの取得の手法について言及されており，プロセッサで独立したタイミングで取得する方法やシステム全体で同期して取得する方法などがある．チェックポイントとは計算機内のメモリのイメージのバックアップである．一定間隔でプライマリ計算機からセカンダリ計算機へタスクの処理結果や処理過程をコピーする．プロセッサの停止故障が発生したとき，親タスクノードを辿り停止故障していないプロセッサからチェックポイントで取得した結果を参照し，子タスクノードが再計算していくことによって，停止故障時の結果を回復していく．停止故障が発生していないとき，通信等によるオーバヘッドは生じないものとなっている．また，Active Standby，Passive Standby でのチェックポイント性能の比較，文献の提案手法におけるチェックポイントの動作間隔によるチェックポイントの生成によるコスト比較，リカバリに要する時間の比較などがなされている．Active Standby，Passive Standby は High Availability システムでの構成の一つである．Active Standby とは，複数台の計算機上で同じタスクを同時並行で処理を行いデータをバックアップするシステムである．プライマリ計算機の停止故障が発生した場合，停止故障が発生していないセカンダリ計算機の処理データを利用して回復する．Passive Standby とは，プライマリ計算機でタスク処理を行い，チェックポイントを取得しプライマリ計算機にバックアップするシステムである．プライマリ計算機の停止故障が発生した場合，停止故障の発生していないセカンダリ計算機でチェックポイントを参照して処理データを回復するシステムである．文献ではマルチコアプロセッサの停止故障を考慮されていないほか，ネットワークトポロジは考慮したが，ネットワークコンテンションは考慮されていない．

以上のように，ネットワーク上でのデータ転送のタイミングを制御しパケットの衝突を防ぎつつ，計算機への処理割り当てを決定するモデルやマルチコアプロセッサを利用したとき

効率的な処理割り当てを決定するモデルなど，それぞれの問題に特化した研究がこれまでなされてきている．しかし，マルチコアプロセッサを考慮したもののネットワークは遅延のない理想的な環境でのモデルや，ネットワーク上での制約を考慮したが，マルチコアプロセッサを考慮されていないため，マルチコアプロセッサを含む環境では効率的な割り当てが困難になるという問題がある．特に，シングルコアプロセッサ向けに想定されたスケジューリング手法ではマルチコアプロセッサのコア間での通信がネットワークでの通信に比べて非常に高速であり，タスク処理時間を短縮する方向に向けてスケジューリングを行うと，一つのマルチコアプロセッサにタスクが集中しがちである．この場合，プロセッサの停止故障が発生したとき，1 マルチコアプロセッサ内に含まれるコア上のデータを全て損失してしまうことになる．その後の回復処理を行うとき，すべてのタスクをやり直さなければならないという問題が生じる．その上，各コアごとにチェックポイントを保有していた場合，それらは同時に失ってしまうため，チェックポイントを活用することができないという問題も生じる．また，停止故障に関してもシングルコアプロセッサを考慮したモデルが多いほか，ネットワーク上における制約とマルチコアプロセッサの停止故障を同時に考慮したモデルは著者らの知る限りでは存在しない．よって，本稿では，ネットワークにおけるコンテンションおよびマルチコアプロセッサを主した計算機環境でマルチコアプロセッサが停止故障することを想定したタスクスケジューリング問題を定式化し，その問題を解くタスクスケジューリングアルゴリズムを提案する．

3. 問題定義

3.1 概　　要

タスクスケジューリングの入力は後述するタスクグラフ，プロセッサグラフとする．また，出力はタスクノードにプロセッサノードを割り当てたスケジュールとする．目的関数は，ジョブの処理時間，停止故障の影響を最小化することである．

3.2 諸　定　義

本節では，提案手法を必要になる用語について解説を行う．また，以降で使用する記号を表 1 にまとめる．

タスクグラフ

タスクグラフ G とは並列実行可能なひとまとまりのジョブを表現したものである[4]．タスクグラフは DAG で表すことができ，頂点をタスクノードと呼び，ジョブの一部を一つのプロセッサで実行するためのデータとして表現している．タスクノード v の処理に必要な時間を $C_{comp}(v)$ と表す．辺をタスクリンクと呼び，タスクノード間で必要となる通信を表している (図 1 参照)．その通信に要する時間は $C_{comm}(e)$ と表すことができる．タスクノード及びタスクリンク全ての集合をそれぞれ，$\mathbf{V}$，$\mathbf{E}$ とするとき，タスクグラフは $G = (\mathbf{V}, \mathbf{E}, C_{comp}, C_{comm}, L)$ と表す．

表 1　記号表
Table 1　Notations

記号	意味
G	タスクグラフ
$\mathbf{V}$	タスクノードのすべての集合
$\mathbf{E}$	タスクリンクのすべての集合
$C_{comp}(v)$	タスクノード $v \in V$ の計算コスト
$C_{comm}(e)$	タスクリンク $e \in E$ の転送コスト
H	プロセッサグラフ
$\mathbf{P}$	プロセッサノードのすべての集合
$\mathbf{R}$	プロセッサリンクのすべての集合
$\mathbf{M}$	マルチコアであるプロセッサノードのすべての集合 ($M \subseteq P$)
$T_{classic}$	既存手法による停止故障による回復時間を含めたタスク処理時間
$T_{latency}$	タスク実行終了目標時間（レイテンシ）
n_i	i 番目のタスク
c_{ij}	i 番目のタスクから j 番目のタスクへの通信
$w(n_i)$	タスク n_i の計算時間
$c(e_{ij})$	タスク n_i からタスク n_j への通信時間
$proc(n)$	タスクノード $n \in V$ が割り当てられているプロセッサ
$succ(n_i)$	タスク n_i の親ノード
$pred(n_i)$	タスク n_i の子ノード
$tl(n_i)$	タスク n_i の全ての子孫ノードを辿った中での最大の処理時間
$bl(n_i)$	タスク n_i の全ての祖先ノードを辿った中での最大の処理時間

本稿で扱うタスクグラフは Oliver[1] らのモデルをベースとしており，制約条件に関しては論文の Condition 1〜3 に該当し，それぞれ，同じプロセッサに割り当てられたタスクノードの処理はどちらかが終了するまで実行できないということ，親タスクノードの処理が終了し，そのデータが転送が子タスクノードに全て転送されるまで実行できないこと (本稿ではこれを先行制約とする)，タスクノードの処理をスケジュールする際，その処理がプロセッサの空き時間内に終了することを制約としている．i 番目のタスクノードのタスクを n_i と表し，その親ノードを $pred(n_i)$，子ノードを $succ(n_i)$ と表す．

また，それぞれのタスクノードで処理を実行するには，親タスクノードによる処理と通信

がすべて終了していなければならいという依存関係をもつ．

プロセッサグラフ

プロセッサグラフ H とはプロセッサとネットワークを表現しており，これはすなわち計算機とネットワークの接続関係を表したものであり，ネットワークトポロジとしても見ることができる[4]．プロセッサグラフは図 2 のように表す．頂点をプロセッサノードと呼ぶ．リンクを 2 つ以上持つものはネットワーク上のスイッチまたは，マルチコアプロセッサが搭載されている計算機のネットワークインターフェイスとする．リンクが 1 つであるものは計算機内の CPU を表しており，辺はプロセッサリンクと呼び，プロセッサ間の通信路を表している．プロセッサノード及びプロセッサリンク全ての集合をそれぞれ $\mathbf{P}$，$\mathbf{R}$ と表す．マルチコアプロセッサ全ての集合を $\mathbf{M}$ としたとき，プロセッサグラフは $H = (\mathbf{P}, \mathbf{R}, \mathbf{M})$ と表すことができる．

また，本研究では，スイッチ自体の処理時間はタスク処理に比べ非常に短いため無視できるものとする．

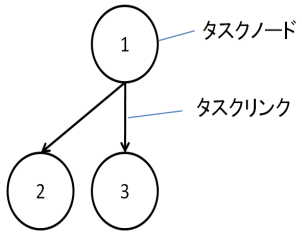

図 1　タスクグラフ

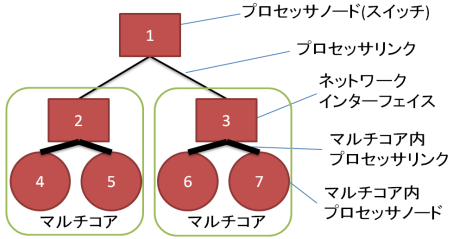

図 2　プロセッサグラフ

スケジュール

本稿でのスケジュールとはプロセッサの空き時間にタスクを割り当て，ネットワークの空き時間にタスク間のデータ転送を割り当てることとする．そのため，本稿のスケジュールでは，タスクグラフとプロセッサグラフを与え，ネットワークコンテンションを考慮したプロセッサに対してタスクを割り当てた結果となる．

3.3　問題の仮定

本稿では，タスク処理は与えられたレイテンシ制約を満たして終了しなければならないとする．レイテンシ制約は，停止故障が一回発生した場合においても，与えられた時間 $T_{latency}$ 内に全タスク処理を終了しなければならない制約である．これを定式化すると式 1 で表すことができる．

また，ネットワークリソースに関しても有限であるため，ネットワークコンテンションの発生は免れない．そのためネットワークに関する制約も設定する必要がある．ネットワークコンテンションとはタスクノードが次のタスクノードへのデータを転送する際に，他のタスクノード間の転送によってネットワークが利用されているとき，パケットが衝突し同時に転送できなくなることである．図 2 のプロセッサ 2 からプロセッサ 3 へデータを転送する際，同時に逆方向へのデータ転送が存在した場合，パケットの衝突が発生する．ここでも，ネットワークコンテンションを回避するため，Oliver[1] らのモデルをベースとする．文献中の Condition 4〜6 で述べられている制約条件を本研究でも適用し，タスクノード間のデータ転送は同時に行えないこと，データ転送の開始時間が先行のデータ転送の開始時間より小さくできないこと，転送時間のスケジュールを行うとき，利用されるネットワークが使用されていない空き時間内にデータ転送終わることが挙げられており，これを元に拡張を行うものとする．また，データ転送の上下問わず同一リンク上で複数の転送は同時に行えないものとし，リンクの帯域は全て一定とする．

計算機環境としてはデータセンタを想定しており，1 ホップを 1 つのネットワークインターフェースやスイッチを経由するものとしたとき，全計算機のプロセッサに 3 ホップで到達できるものとなっている．また，4 ホップで到達できる範囲に Active Standby の計算機が必ず搭載されているものとする．本稿での停止故障の発生を想定しており，停止故障の検出は一定間隔でハートビートと呼ばれる検出信号を各計算機に送信し，一定時間内でレスポンスがなければ，停止故障が発生しているものとして判断する．ハートビートはネットワーク上で消失することやデータ転送が優先され遅延が起こるといったことは発生しないものとする．また，ハートビートによるレスポンス取得回数による検出に関しては，文献[5] にしたがって 1 度の検出でも信用できるため，これを適用する．

本稿でのマルチコアプロセッサはコア間での転送時間はネットワークでの転送時間と比較すると無視できるほど小さいものとして 0 とする．マルチコアプロセッサのネットワークインターフェイスはコア間で共有するものとし，プロセッサの停止故障が発生した場合，そのプロセッサ上にあるタスクのデータはすべて失われるものとする．

チェックポイントの取得はプロセッサノードがそれぞれの各タスクノードの処理を終了した時点で確保されるものとし，また，チェックポイントへの書き出し，チェックポイントの読み込みの処理時間は 0 とし，チェックポイントによるリソースの消費はないものとする．

3.4 問題の定式化

本問題の入力としてタスクグラフ $G = (\mathbf{V}, \mathbf{E}, C_{comp}, C_{comm}, L)$，プロセッサグラフ $H = (\mathbf{P}, \mathbf{R}, \mathbf{M})$ を与える．出力は各タスクノード，タスクリンクをプロセッサノード，プロセッサリンクへの割り当てたスケジュールである．

$w(n_i)$ は i 番目のタスク n_i の処理時間，$c(e_{ij})$ は i 番目のタスク n_i から j 番目のタスク n_j への通信に要する時間，$tl(n_i)$ はタスク n_i の全ての子孫ノードを辿った中での最大の処理時間，$bl(n_i)$ はタスク n_i の全ての祖先ノードを辿った中での最大の処理時間をそれぞれ表しているものとし，レイテンシ制約が $T_{lantecy}$ の場合，タスクグラフの各タスク n_i では以下の条件が成り立たなければならない．

$$tl(n_i) + bl(n_i) - w(n_i) < T_{latency} \tag{1}$$

$$bl(n_i) = w(n_i) + max_{n_j \in succ(n_i)} \{c(e_{ij}) + bl(n_j)\}$$
$$tl(n_i) = w(n_i) + max_{n_h \in pred(n_i)} \{c(e_{hi}) + tl(n_h)\}$$

Oliver らの手法による停止故障発生時のタスク回復時間を含めたタスク処理時間を $T_{classic}$ としたとき，式 2 を満たす場合は Oliver らのタスクスケジューリングを行う．

$$tl(n_i) + bl(n_i) - w(n_i) > T_{classic} \tag{2}$$

本問題の目的関数は以下の式を与える．

$$\text{minimaize } (\mathrm{bl}(\mathrm{n_{start}})) \tag{3}$$
$$\text{subject to constraints (1)}$$

4. 提 案 手 法

本章では，3 章で定義した問題を解くための基本方針を示し，ネットワークコンテンション，マルチコアプロセッサ，停止故障をを考慮したタスクスケジューリングアルゴリズムを示す．

4.1 提案手法の概要

3 章で述べたとおり本稿で扱う問題は組合せ最適化問題であり，最適解を短時間で算出することは難しい．そこで，本問題を効率的に解くためにリストスケジューリングを拡張した近似アルゴリズムを提案する．古典的なリストスケジューリングではネットワークコンテンションが考慮されていないため,，ネットワークコンテンションを考慮した Oliver[1] らのリストスケジューリングを元にした拡張を行う．これは，文献中の Algorithm1，2 に相当する．

本手法の基本方針は Oliver らのタスクスケジューリングによってタスクをプロセッサに割り当てを行なった後，全タスクに対して 1 タスクずつ停止故障が発生したとき，タスク回復後の処理時間を含めタスク終了まで要する時間をリスト化する．その後，生成したスケジュールが停止故障発生時にもレイテンシを満たしていれば，生成したスケジュールを適用する．レイテンシを満たしていなければ，リストから停止故障による影響が最大となるタスクと依存関係にある 1 つ前のタスクで割り当てるプロセッサを変更し，停止故障発生時に一つ前のプロセッサを辿り途中の処理結果を利用して回復できるように分離する．この時点でも，レイテンシを満たしていない場合，リストから次に影響が大きくなるタスクを求め，前述と同様に別プロセッサに割り当てるように分離を行う．これをレイテンシを満たすまで繰り返すことによって，停止故障時の回復時間を短縮する．

4.2 アルゴリズム

本アルゴリズムは，マルチコアプロセッサが主となる計算機環境において，ネットワークコンテンションを考慮し，プロセッサの停止故障発生時のレイテンシを制約内に収め，停止故障非発生時のタスク処理時間を最小化するためにスケジュールを決定する．アルゴリズムの擬似コードを Algorithm1 に示す．

本アルゴリズムは，タスクグラフの先行制約とクリティカルパスによる処理すべきタスクノードの優先度順にソートしたリストを生成することから始める．次に生成されたリストを順に参照し，タスクノードの計算が最も早く終了するプロセッサを探索する．

このとき，参照したタスクノードが停止故障発生時に最も回復に時間を要するタスクであった場合，できるだけ親タスクノードが割り当てられているプロセッサを避けて，別プロセッサに割り当てを行う．

参照したタスクノードの親タスクノードを取得して，12 行目で見つけたプロセッサと同じプロセッサに割り当てられているか計算する．異なるプロセッサに割り当てられている場合，親タスクノードからの使用するプロセッサリンクを決定して，プロセッサリンク上に通

信をスケジュールする．最後に，タスクノードをプロセッサノードにスケジュールする．

以上の計算を行うことで，停止故障非発生時の回復時間を含めた処理時間をレイテンシ制約内に収め，かつ停止故障非発生時の処理時間をヒューリスティックに最小化するスケジュールを算出できる．

Algorithm 1 マルチコアプロセッサの停止故障発生時のリカバリ時間短縮を考慮したアルゴリズム

全てのタスクノード n を先行制約を満たした状態で $bl(n)$ の最も大きくなる順でソートを行いリスト L に格納する．
while true **do**
  if L が空 **then**
    ループから抜ける．
  end if
  リスト L から先頭の要素を取り出し n_j に代入する．
  n_j を p に割り当てたとき，終了時間が最も早くなる $p \in \mathbf{P}$ を見つける．
  if 停止故障発生時にタスク全体の処理時間が最も長くなるタスク $= n_i$ **then**
    $proc(pred(n_j))$ と異なる割り当て済みタスク数が最も少ないプロセッサに n_j 割り当てる．
  end if
  for each $n_i \in pred(n_j)$ **do**
    if $proc(n_i) \neq p$ **then**
      $proc(n_i)$ と p を結ぶリンクに e_{ij} を割り当てる．
    end if
  end for
  p に n_j を割り当てる．
end while

4.3 トポロジ制約での回復処理

上記のアルゴリズムによってタスクスケジューリングを行ったにもかかわらず，タスクグラフの形状や規模によっては停止故障非発生時の回復時間を含めた処理時間をレイテンシ制約内に収めことができないことも考えられる．その場合，データ転送で 1 ホップ余分に経由することになるが，4 ホップ内に存在する Active Standby の計算機を利用し，タスク処理時間の短縮を図る．停止故障発生時の影響の大きいタスクを並列処理を行い，レイテンシ制約を満たすようにスケジュールするものとする．

5. 実験及び考察

5.1 実 験 概 要

提案手法におけるタスクスケジューリングによってプロセッサの停止故障は発生時，非発生時のタスク処理時間に対する効果を評価した．実験ではタスクグラフの直列数，プロセッサ数の変更を行い，停止故障発生時の影響について処理時間の変化より，提案手法の性能を評価した．実験において用いた詳細なパラメータは次の節で示す．

本実験を行った計算機の環境は，Intel Core i7 920 (2.67GHz)，メモリ 6.0GB，Windows 7(64bit)，Java(TM) SE Runtime Environment (build 1.6.0_21-b07) である．

5.2 実　　験

本節では，提案手法のシミュレーションを行うにあたって，比較手法および，実験パラメータについて説明する．

5.2.1 比較手法およびパラメータ

本実験では，プロセッサの停止故障発生時と停止故障非発生時を比較し，停止故障発生時，タスクの処理時間がレイテンシを満たすことを確認し，停止故障非発生時のタスク処理時間の最小化が実現できているかを調べる．

タスクグラフの先行制約における関係上，最下段のタスクでプロセッサの停止故障が発生したとき処理時間への影響が最も大きく，そのため，プロテッサの停止故障は上記の時点で発生するものとする．

比較手法は，Oliver らによる，ネットワークコンテンションを考慮したタスクスケジューリング手法を採用し，マルチコアプロセッサ，停止故障が発生する環境で提案手法と，停止故障非発生時と発生時におけるタスクの処理時間を比較する[1)]．

実験での入力として図 3 のようなタスクグラフ，図 4 のようなプロセッサグラフを与え出力としてタスクスケジューリングを行った後，処理の実行にかかる時間を得た．

本実験では，表 2 のパラメータを用いた．

5.3 実験の結果

本実験における結果を図 6，図 7，図 8，図 9，図 10，図 11 に示す．図 6 は停止故障発生時のタスクグラフ直列数に対するタスク処理時間について，プロセッサ数を 2，タスクグ

表 2 実験パラメータ
Table 2 Experimental Parameter

記号	意味
タスク数	6〜66
1 タスクの計算時間	10
1 タスクの通信時間	1
タスクグラフ並列数	4，64
タスクグラフ直列数	1〜7
プロセッサコア数	4
プロセッサ数	1〜8

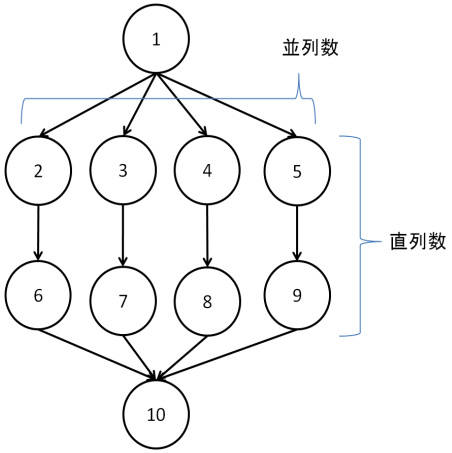

図 3 タスクグラフ

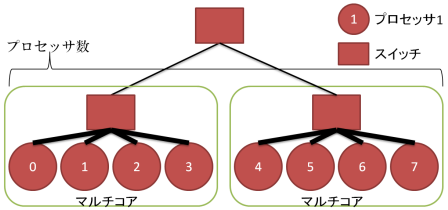

図 4 プロセッサグラフ

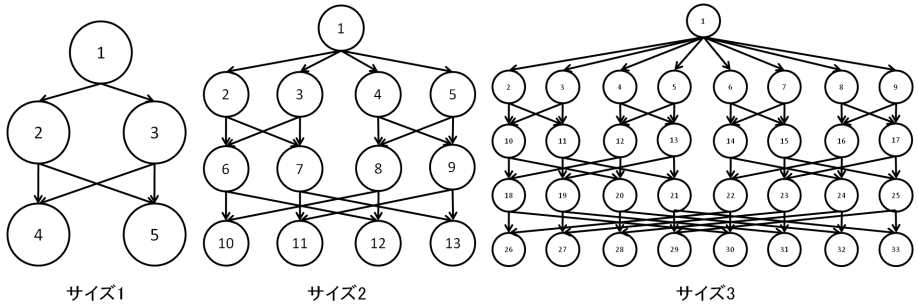

図 5 データ転送の多いタスクグラフ

ラフ並列数を 4 に固定して既存手法と提案手法を比較したものである．プロセッサの停止故障発生時は既存手法が約 2 倍の処理時間になっていることがわかる．

一方で図 7 は停止故障非発生時の結果で，提案手法は若干，既存手法より処理時間が増加しているが，既存手法との間隔はほぼ一定で大きく差が開くことはない．

また，図 8 は停止故障発生時のプロセッサ数に対するタスク処理時間について，タスクグラフ直列数を 1，タスクグラフ並列数を 64 に固定して既存手法と提案手法を比較したものである．プロセッサ数が 2〜4 のとき，既存手法に比べ提案手法はでは，処理時間が短くなっていっているが，それ以降は一定の差である．

図 9 は停止故障非発生時の結果であるが，既存手法と比較しても，最大 7%増加の処理時間となっている．

また，図 10 ではデータ転送の多いタスクグラフ (図 5 参照) を与え，そのサイズを 1〜3 で変更してタスク処理時間について既存手法と提案手法を比較したものである．なおプロセッサ数は 3 で行った．このときの提案手法のタスク処理時間は既存手法に比べ約 0.7 倍に短くなっている．

図 11 は停止故障非発生時の結果であるが，既存手法と比較しても，最大 6%増加の処理時間となっている．

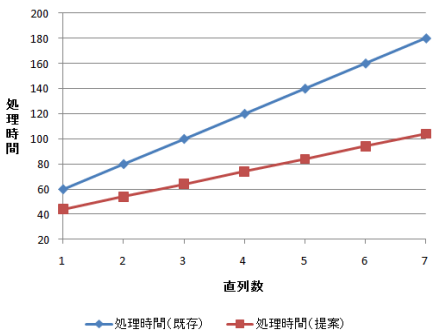

図 6 タスクグラフ直列数に対する停止故障発生時の比較

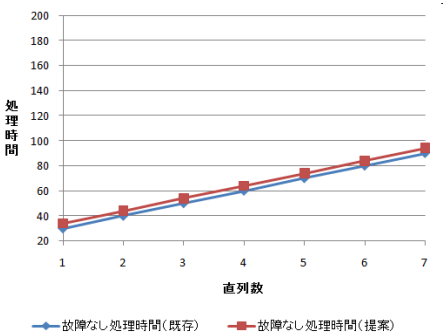

図 7 タスクグラフ直列数に対する停止故障非発生時の比較

5.4 考　　察

タスクグラフ直列数の変化による効果を調べる実験においては，タスクグラフ直列数が増

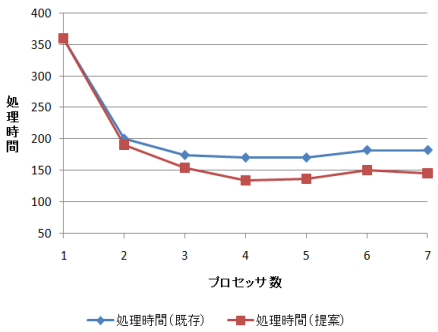

図 8 プロセッサ数に対する
停止故障発生時の比較

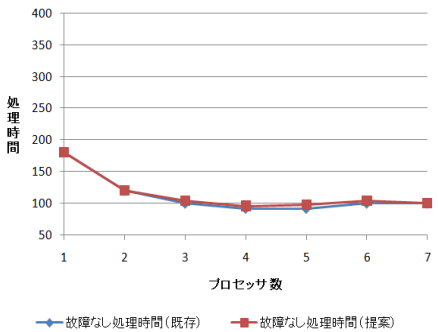

図 9 プロセッサ数に対する
停止故障非発生時の比較

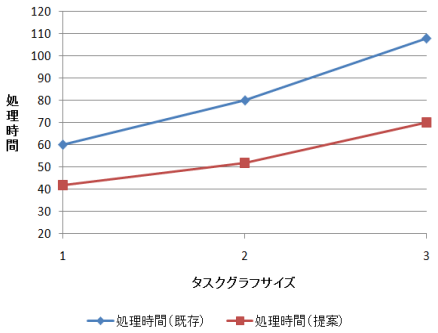

図 10 タスクグラフサイズに対する
停止故障発生時の比較

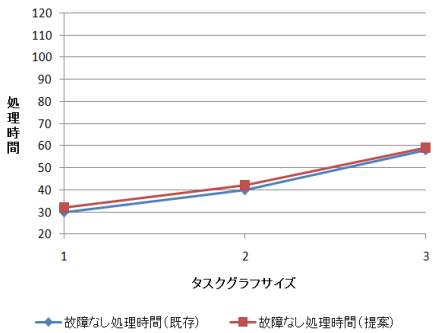

図 11 タスクグラフサイズに対する
停止故障非発生時の比較

加するにつれて既存手法との差が開いた．原因としては，提案手法では最も影響力が大きいタスクを別プロセッサに割り当てたため停止故障が発生しても，以前に処理したデータを再処理せずチェックポイントから回復できるためである．一方で，既存手法では停止故障発生時に全タスクのやり直しが必要であったため，約 2 倍の時間が必要になった．

一方，停止故障非発生時に提案手法の処理時間が増加した原因に関しては，停止故障時の影響の少ないプロセッサへタスクデータを転送する必要があるため，その転送による時間と，ネットワークコンテンションを考慮したネットワーク転送のスケジューリングによる時間制御で増加した．

また，プロセッサ数の変化による効果を調べる実験においては，プロセッサ数が少ないとき，十分に停止故障発生時に備えたタスクの割り当て移動を行なえず，既存手法と同程度の処理時間となった．しかし，プロセッサ数が増加するにつれて，一定数までは処理時間が短くなる．これは，停止故障の影響を小さくするため，タスクノードの割り当ての少ないプロセッサを選び，停止故障の影響が大きいタスクノードの割り当てを行えたからである．それ以降，処理時間が既存手法と一定の差で保たれる要因は，プロセッサ数が並列数に対して十分に用意されているため，全てのプロセッサに割り当てる必要がなくなることによる影響である．

データ転送の多いタスクグラフを与えサイズの変化による効果を調べる実験においては，停止故障発生時，サイズ 1〜3 に関しては既存手法に比べ提案手法は処理時間を約 0.7 倍に短縮できた．原因としては，このときもプロセッサ数が並列処理より多く，影響の大きいタスクを別プロセッサに割り当てることができ，再処理が必要なタスクを大幅に減らすことができたためである．また，停止故障非発生時に関しては，提案手法による処理時間が若干既存手法より増加する結果となったが，これは上記でも述べたように，ネットワークコンテンションを考慮したスケジューリングを行ったためである．

以上より，タスクグラフ並列数に対して，十分にプロセッサの数が割り当てられるとき，提案手法は有用であると言える．

6. ま と め

本稿では，ネットワークコンテンションが発生し，マルチコアプロセッサを搭載した計算機が主となるデータセンタを想定した環境でプロセッサの停止故障が発生するタスクスケジューリング問題で，マルチコアプロセッサの停止故障発生時のタスク処理時間がレイテンシ制約を満たしつつ，通常時のスケジュール時間を最小化する問題を定式化した．その問題

を解くために，まず既存手法のアイデアを用いたネットワークコンテンションを考慮したリストスケジューリング法を述べ，それに加えマルチコアプロセッサでのタスク割り当ての偏りを抑制したタスクスケジューリング手法を提案した．停止故障発生時もレイテンシ制約を満たしつつ，通常時のスケジュールを最小化するという特色がある．

提案手法のタスクスケジューリング性能を評価するために既存手法および提案手法での比較実験をシミュレーションにより行った．その結果，タスクグラフの直列数に対する実験では，プロセッサの停止故障発生時の処理時間が既存手法の約半分まで短縮することが確認できた．さらに，タスクグラフの直列数の増加に伴い，タスク処理時間は増加するが，既存手法に比べ提案手法では増加量を小さく抑えることが出来ていることを確認した．また，プロセッサ数に対する実験では，プロセッサ数が 3 以上の場合，常に処理時間が既存手法より小さくなることが確認できた．停止故障非発生時の処理時間に関しては，既存手法に比べ最大 7%増加する結果となった．データ転送の多いタスクグラフのサイズに対する実験でも，停止故障発生時に関しては，提案手法は既存手法に比べ約 0.7 倍のタスク処理時間に短縮でき，停止故障非発生時に関しては，タスク処理時間の増加をわずかに抑えられていることを確認した．停止故障非発生時の処理時間に関しては，既存手法に比べ最大 6%増加する結果となったが，停止故障発生時のレイテンシ制約を満たすためへのトレードオフと考えると，提案手法は有用であること言える．

今後の課題として，より現実に近い複雑なタスクグラフ，複雑なプロセッサグラフでの処理時間を短縮できるアルゴリズムへの性能向上，停止故障発生時に影響の大きいタスクの割り当て方の考慮が挙げられる．

参 考 文 献

1) O., Sinnen and L., A., Sousa.: “Communication Contention in Task Scheduling,” *IEEE Trans. Parallel and Distributed Systems, vol. 16, no. 6, pp. 503-515, 2005.*
2) Y., Gu, Z., Zhang, F., Ye, H., Yang, M., Kim, H., Lei and Z., Liu.: “An empirical study of high availability in stream processing systems,” *Middleware(2009), pp. 23:1-23:9, 2009.*
3) Wolf, J.L., Bansal, N., Hildrum, K., Parekh, S., Rajan, D., Wagle, R., Wu, K., Fleischer, L.: “SODA: An Optimizing Scheduler for Large-Scale Stream-Based Distributed Computer Systems,” *Middleware(2008), pp. 306-325, 2008.*
4) 吉永 一美，小出 洋: “ストリーミングデータ処理のためのタスクスケジューリング” 情報処理学会研究報告. ハイパフォーマンスコンピューティング，*2006(87), pp. 139-144, 2006.*
5) Zhe, Zhang., Yu, Gu., Fan, Ye., Hao, Yang., Minkyong, Kim., Hui, Lei., Zhen, Liu.: “A Hybrid Approach to High Availability in Stream Processing Systems,” *International Conference on Distributed Computing Systems, (ICDCS 2010), pp. 138-148, 2010.*
6) 須田 礼仁: “ヘテロ並列計算環境のためのタスクスケジューリング手法のサーベイ” 情報処理学会論文誌. コンピューティングシステム，*Vol. 47, pp. 92-114, 2006.*
7) N.M. Amato, P. An, “Task Scheduling and Parallel Mesh-Sweeps in Transport Computations,” *Technical Report, TR00-009, Department of Computer Science, Texas A&M University, 2000.*
8) 尾高 輝，甲斐 宗徳：“通信遅延を考慮したタスクスケジューリングのためのタスク粒度解析” 成蹊大学理工学研究報告 *44(1), 17-23, 2007.*
9) O., Sinnen and L., A., Sousa.: “Contention-Aware Scheduling with Task Duplication,” *Workshop Scheduling and Resource Management for Cluster Computing, pp. 382-387, 2001.*
10) O., Sinnen, L., Sousa, and F., E., Sandnes.: “Toward a realistic task scheduling model,” *IEEE Transactions on Parallel and Distributed Systems, pp. 263-275, 2006.*