卒業研究論文

タスク複製による大規模計算環境向け分散タスクスケジューリング

指導教員　　松尾 啓志 教授
津邑 公暁 准教授

名古屋工業大学 工学部 情報工学科
平成 17 年度入学　17115158 番

山城 穂高

平成 21 年 2 月 10 日

タスク複製による大規模計算環境向け分散タスクスケジューリング

山城 穂高

内容梗概

ネットワーク上に存在する多数の遊休計算資源の処理能力を集約し, 大規模分散計算に利用しようという研究が数多く行われている. 遊休計算資源を利用した大規模分散計算環境では, 計算資源が分散計算に提供する処理能力が動的に変動するため, 不適切な負荷の割り当てによる計算時間の増加が発生する. そのため, 適切な量の負荷を各計算資源に割り当てる動的負荷分散手法が求められている. 本論文では, 遊休計算資源がタスクを重複実行することによって計算時間を短縮する動的負荷分散手法を提案する. 提案手法は, 単一のサーバが集中的に負荷分散処理を行う従来手法に比べて高いスケーラビリティを実現する. 本論文では, 多数の計算資源を利用した大規模分散計算環境をシミュレータ上で再現し, 提案手法と従来手法の動的負荷分散性能の比較評価を行った.

タスク複製による大規模計算環境向け分散タスクスケジューリング

目次

1 はじめに

近年, 計算機技術の発達により計算機の高性能化と, ネットワークの高速化が進んでいる. しかし, WEB の閲覧や, 文章の作成, メールの送受信などの通常の用途で計算機を利用する場合,CPU リソースの大部分が遊休状態になっているといわれている. そこで, ネットワーク上に存在する余剰 CPU リソースを集約し, 1 台の仮想的なスーパーコンピュータを形成し, 大規模分散計算を実現しようとする試みが行われている. 代表的な例として, 一般ユーザがプロジェクトに計算資源を提供するボランティアコンピューティングや, 組織内または組織間で計算資源を共有するグリッドコンピューティングなどがあげられる [10][11].

上記のような分散計算環境は, 多数の計算資源の余剰 CPU リソースを集約しているため, 計算資源の処理性能は不均一であり, 分散計算に提供できる処理能力は常に変動する. このような環境で, 並列アプリケーションを効率よく実行させるためには, 動的に変化する処理能力に応じた, 適切なタスク割り当てを行うことが必要となる. これはタスクスケジューリング問題と呼ばれ, これまでにタスクスケジューリング問題に関して様々な研究が行われてきた [8] [14] [19]. その研究の多くは, 比較的小規模な計算環境を想定したものであり, スケジューリングサーバが集中的にタスクスケジューリングを行うことで, 高い動的負荷分散性能を実現している. しかし, グリッドコンピューティングのような数千台の計算資源を利用する大規模分散計算環境において, スケジューリングサーバが集中的にタスクスケジューリングを行う手法では, スケジューリングサーバのスケジューリングコストが増大し, 動的負荷分散性能が低下することが予想される. そこで, 本論文では大規模分散計算環境において, 高い動的負荷分散性能を実現するタスクスケジューリング手法を提案する.

本論文の提案手法は, 分散計算に参加する計算資源が自律的にタスクを重複実行することで動的に負荷の分散を行う. この手法では, 各計算資源が自律的にタスクスケジューリングを行うことによって, 集中的にタスクスケジューリングを行うよりもスケーラビリティの高いタスクスケジューリングが可能となる.

以下本論文では,2 章で研究の背景,3 章でタスクスケジューリングの既存手法について説明し,4 章で提案手法について述べ,5 章で評価を行い, 最後に 6 章で本論文のまとめを行う.

2　研究の背景

IT 技術の進歩により, 計算機の高性能化とネットワークの高速化が進んでいる. しかし, 一般ユーザの多くは計算機を WEB の閲覧や文章作成, メールの送受信などにしか利用していない. そのため, 高性能な計算資源の処理能力は大部分が余剰能力となっている. 大規模分散計算では, ネットワーク上にある計算資源の余剰能力を統合し, 仮想的なスーパーコンピュータとして利用する形態をとっている.

大規模分散計算環境において, 計算資源の処理性能はそれぞれ異なり, 分散計算に提供できる処理能力が変動する. そのため, 並列アプリケーション実行時に, タスクを静的に割り当てると計算資源ごとの計算時間にばらつきが生じ, 全体の計算時間が最も計算時間の長い計算資源に依存してしまう. このような環境で, 並列アプリケーションを効率よく実行させるためには, 動的に負荷を分散させてすべての計算資源の計算時間を均一化することが必要となる. これはタスクスケジューリング問題と呼ばれ, これまでにタスクスケジューリング問題を解決するための様々なタスクスケジューリング手法が提案されてきた. そのタスクスケジューリング手法の多くは, 小規模の計算環境を想定したものであり, 専用のスケジューリングサーバが集中的に負荷を分散させる手法である. しかし, サーバが集中的に負荷分散させる手法は, 大規模分散計算環境ではスケジューリング処理に長い時間を必要とし, それによってアプリケーション全体の計算時間の増大が予想される.

3　タスクスケジューリングの既存手法

これまでに大規模分散計算環境において, 並列アプリケーションを効率よく実行させるために様々なタスクスケジューリング手法が提案されてきた. 本節では, これまでに提案されてきた, 既存のタスクスケジューリング手法について論ずる.

既存のタスクスケジューリング手法は大きく分けて２つに分類される. そのうちの１つはスケジューリングサーバがモニタリングによって計算資源のモニタリング情報を収集し, その情報をもとに各計算資源の処理能力に応じたタスクスケジューリングを行う適応型のスケジューリング手法である. この手法ではスケジューラが計算資源の動的負荷変動を検出して, 適切な負荷分散を行うことですべての計算資源の計算時間を均一化している. この手法の代表例として AppLeS[2] と Condor[3][4] がある.

もうひとつの手法は, 計算資源のモニタリング情報を利用せずに, 計算資源がスケジューリングサーバよりタスクをダウンロードしてタスクを処理する Work Queue である. この Work Queue モデルを採用した手法の代表例として,BOINC[5] や JNGI [6],P3[7] などがある.

以下, タスクスケジューリングの代表的既存手法である AppLeS を用いた適応型スケジューリングと Work Queue とその改良である WQR について説明し, それぞれの問題点を明らかにしていく.

3.1　AppLeS を用いた適応型スケジューリング

AppLeS(Application Level Scheduling) はタスクスケジューリングに必要な情報を集め, その情報を元に計算資源の動的負荷変動を考慮したタスクスケジューリングを実現することが可能なスケジューリング方式である. 文献 [8] では,AppLeS と Condor の分散計算環境構築機能を組合せた適応型手法を提案している. この手法ではスケジューリングサーバが一定間隔で計算資源の動的情報, タスクの処理状況などのスケジューリングに必要な情報を集める. そしてその情報を元に各計算資源の処理能力に応じたタスクスケジューリングを行う. 以降, 本論文ではこの手法のことを適応型手法と呼ぶ. 適応型手法では, サーバが一定時間ごとに各計算資源を監視し, すべての計算資源のモニタリング情報を集中的に管理することで最適なタスク割り当てを実現することができる. その一方で, 次のような適応型手法の問題点が指摘されている.

- 適応型手法の再スケジューリング間隔
  適応型手法のタスクスケジューリングがどの程度適切に負荷分散できるかは, 再スケジューリングの間隔に大きく依存する. 再スケジューリング間隔を短くすると計算資源の動的負荷変動に短時間で対応できるので並列アプリケーションの実行効率を高めることができる. しかし, この間隔を短くするほど, スケジューリングサーバには多大な負荷がかかる. さらに, ネットワーク上に計算資源のモニタリング情報収集のための無駄なトラフィックが大量に流れることになる. スケジューリングサーバへの負荷と動的負荷変動への即応性がトレードオフとなっており, 適切な再スケジューリング間隔の設定方法が大きな課題となっている.

- 計算資源の増加によるモニタリングコストの増大
  適応型手法では NWS(Network Weather Service)[9] を利用した計算資源の動的情報収集や, 負荷変動の統計的な予測を行っている. しかしモニタリング対象となる計算資源の数が多くなると, モニタリング情報をサーバへ集める時間が増大し, モニタリング情報収集時にかかる時間分だけ, その時刻における計算資源の能力とモニタリング情報とに相違が生じてしまい, スケジューリングの精度が低下する. このような精度の低いスケジューリングを行うと各計算資源の計算時間にばらつきが発生するので並列アプリケーション全体の処理時間増大につながる.

3.2 Work Queue

Work Queue は SETI@home[10] や distributed.net[11] などのボランティアコンピューティングプロジェクトで採用されている代表的なタスクスケジューリング手法である. Work Queue では, 全てのタスクをスケジューリングサーバが集中的に管理している. 遊休状態になった計算資源はスケジューリングサーバにタスクを要求し, サーバが計算資源にタスクを割り当てる. そして, 割り当てられたタスクの処理を終えた計算資源は処理結果をサーバに返すと同時にサーバに新たなタスクを要求する. この手法では処理速度の早い計算機ほど短時間でタスクの処理が完了するので, タスクのダウンロードと処理結果の返却を頻繁に繰り返すことになる. 結果的に高性能な計算資源ほどより多くのタスクが割り当てられることになり, 計算資源の処理能力に応じた適切な負荷分散を実現できる. しかし, この手法では 2 つの問題点がある.

- スケジューリングサーバへの負荷増大
  Work Queue では, 全ての計算資源からの要求をスケジューリングサーバが集中的に処理している．しかし, タスク１つ１つの計算量が少ない場合や, 計算資源台数の数が増えるとスケジューリングサーバに寄せられる単位時間あたりの要求数が増え, 計算資源からの要求に対するレスポンスタイムが長くなってしまい, 並列アプリケーションの実行性能低下につながる．

- 不適切なタスク割り当て
  Work Queue ではスケジューリングサーバが計算資源群に対し, 適切なタスク割り当てを行うとは限らない．たとえば, 性能の高い計算資源と性能の低い計算資源の 2 台で分散計算することを考える．１つのタスクをこの 2 台の計算資源のどちらかにタスクを割り当てることを考えた時, 通常はタスクの実行時間を短くするために, スケジューリングサーバが性能の高い計算資源へタスクを割り当てる．しかし,Work Queue によるタスクスケジューリングを行う場合, 2 台の計算資源の性能によらず, スケジューリングサーバへ先にタスクを要求した計算資源にタスクが割り当てられてしまう．そのため, 性能の低い計算資源の方が先にタスクを要求してしまうと, スケジューリングサーバは性能の低い計算資源にタスクを割り当ててしまい, 並列アプリケーションの実行性能が低下する．

3.3 WQR

Work Queue における不適切なタスク割り当ての問題点を緩和する手法として, WQR(Work Queue with Replication)[12] が提案されている．WQR は Work Queue のアルゴリズムにタスクの複製機能を追加したものである．初期段階およびスケジューリングサーバが全てのタスクを割り当て終わるまでは Work Queue のアルゴリズムと同じである．その後,Work Queue において割り当てられたタスクの処理を終えた計算資源は最後に処理を終える計算機を待っている状態（遊休状態）となっていた．WQR では計算機が実行中のタスクをスケジューリングサーバが複製し, 前述のような遊休状態となっている計算機が複製されたタスクを重複的に処理する．以降, 複製されたタスクのことをレプリカタスク, 複製の元となるタスクをオリジナルタスクとする．実行中のオリジナルタスクは大量に複製され続けることを避けるために, あらかじめ決められた最大複製数を越えない範囲内で複製され続ける．もし, レプリカタスクのほうが先

に終了した場合, オリジナルタスクと他のレプリカタスクの実行はキャンセルされる. オリジナルタスクが先に終了した場合も同様にレプリカタスクの実行はキャンセルされる. このように重複的にタスクを実行することで性能の高い計算機にタスクが割り当てられる確率が高くなり, 不適切なタスクの割り当てが起こる確率を抑えることができる. しかし, 新たな問題や解決できていない問題もある.

- スケジューリングサーバへの負荷増大
  WQR は,Work Queue と同様に全ての計算資源からの要求をスケジューリングサーバが集中的に処理しているため, タスク１つ１つの計算量が少ない場合や, 計算資源台数の数が増えるとスケジューリングサーバへの負担が増大する. さらに, タスクの複製処理もサーバが行うため, Work Queue よりもサーバへの負担が増大してしまう.

- トレードオフの存在
  WQR ではタスクの複製に最大複製数を設定しているが, この最大複製数が小さすぎると不適切なタスク割り当てが発生し, 計算時間が増大する確率が高くなる. 逆に, 最大複製数が大きすぎるとネットワーク上を流れるトラフィック数が増大する. その結果, ネットワークを流れるトラフィック数と適切なタスクの割り当てにトレードオフが存在している.

3.4 既存手法の問題点

大規模分散計算環境のように計算資源が提供する処理能力が動的に変動する環境において, 計算資源の提供する処理能力に応じて動的に負荷分散させることが有効であることは既に明らかにされている. しかし, 既存手法のようにスケジューリングサーバが集中的に動的負荷分散を行う方式では, 管理するべき計算資源が増えるほどタスクスケジューリングのコストが大きくなり, 動的負荷変動に対応した最適な負荷分散が行えなくなることが予想される. そこで, 大規模分散計算環境において有効に機能する, スケーラビリティの高いタスクスケジューリング手法が求められている.

4　提案手法

グリッドコンピューティングのような大規模分散環境において, スケジューリングサーバが集中的に動的負荷分散を行う手法では, 計算資源の管理が困難になり, スケジューリングコストが増大することが予想される. 本論文では, 各計算資源が自律的に重複実行の管理を行うことでタスクスケジューリング処理を分散化させる手法を提案する. 以下本章では, 最初に重複実行制御スケジューラの構成と提案手法のスケジューリングの流れについて説明し, 次に計算資源の状態遷移と重複実行の判定処理について説明していく.

4.1　重複実行制御スケジューラ

既存手法では, スケジューリングを行う専用のサーバと, 実際にタスクの処理を行う計算資源群が明確に区別されている. これに対し, 本論文の提案手法では専用のスケジューリングサーバが存在せず, 個々の計算資源が自律的に計算資源間のやりとりと重複実行の管理を行うことで, 分散タスクスケジューリングを実現している. 4.1 節では, 分散タスクスケジューリングを実現するための, スケジューリング機構について説明する.

本論文の提案手法では, 各計算資源に図 1 に示されるモジュール群を持たせる. 以降, このモジュール群のことを重複実行制御スケジューラとする. 各計算資源は OS が持つローカルスケジューラとは別に重複実行制御スケジューラを持つものとする. 重複実行制御スケジューラは, タスクディスパッチャ, タスクキュー, メッセージ管理ユーティリティ, ネットワークマネージャの４つのモジュール群から構成される. 以下では, この４つのモジュール群について説明する.

タスクディスパッチャ　タスクディスパッチャの役割はタスクキューの管理, タスクへの CPU 時間の割り当て,CPU の負荷状態のモニタリングである. タスクが完了すると, タスクディスパッチャがタスクキューから次の未完了タスクを取り出して,CPU 時間を割り当てる. すべてのタスク処理を終えた場合は, メッセージ管理ユーティリティに処理を終えた事を伝える. また, メッセージ管理ユーティリティから割り込み要求が起きた場合は処理中タスクを中断し, その命令に応じた処理を行う.

タスクキュー　新たに割り当てられたタスクや複製されたタスクなどの未完了タスク

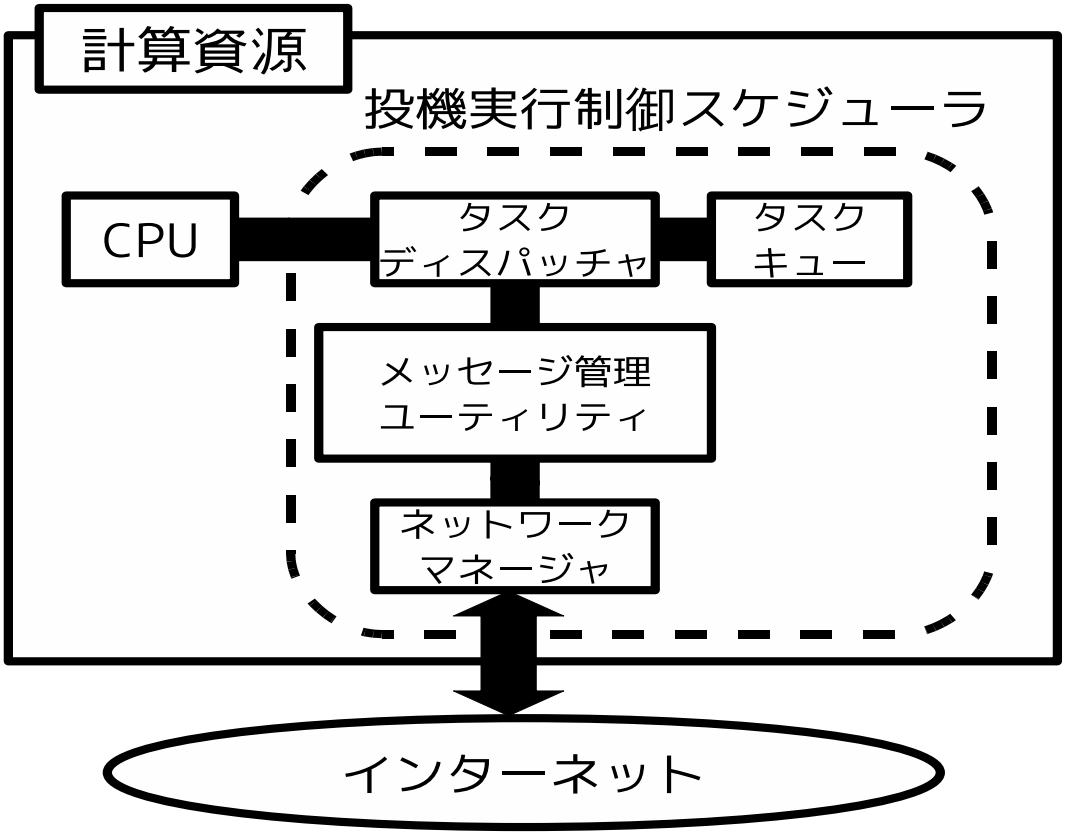

図 1: 計算資源上のスケジューリングモジュール群

はタスクキューに入れられる.

メッセージ管理ユーティリティ　メッセージ管理ユーティリティの役割は, ポーリングメッセージの処理, 重複実行の判定処理,Cancel メッセージの処理である. 計算資源間のやりとりは, メッセージ管理ユーティリティで行われる. また, メッセージ管理ユーティリティでは, 必要に応じてタスクディスパッチャに割り込み要求を送る.

ネットワークマネージャ　ネットワークマネージャの役割は計算資源間通信の仲介, ポーリングリストの保持である.

4.2 スケジューリングの流れ

既存手法ではスケジューリングサーバが集中的にタスクスケジューリングすることですべての計算資源の計算時間を均一化している. これに対し, 本論文の提案手法では遊休状態になった計算資源がタスクを重複実行し, すべての計算資源の計算時間を均一化する. 以下に計算資源間のやりとりを含めた, 提案手法のスケジューリングの流れを示す.

STEP 1 スケジューリングサーバに並列アプリケーションの実行要求が発生すると, サーバが各計算資源への初期割り当てタスク数を決定するための静的スケジューリングを行う. この静的スケジューリングでは, スケジューリングサーバが各計算資源のモニタリング情報を集め, その情報をもとに処理能力に応じてタスクを計算資源に割り当てる. 同時に各計算資源にポーリング先 IP アドレスのリストも送信する. 以降, 静的スケジューリングによって割り当てたタスクをオリジナルタスク, ポーリング先 IP アドレスのリストのことをポーリングリストと呼ぶ. そして, 静的スケジューリングにより割り当てられたオリジナルタスクの処理を終えた計算資源は, 複製したタスクの送信先に処理中タスクの中止をさせる Cancel メッセージを送信して STEP2 に移行する. （タスクの複製をしていない場合はそのまま STEP2 に移行する）

STEP 2 遊休状態となった計算資源は, ポーリングによって処理時間を短縮できそうな処理状態計算資源を探し出す.

遊休状態の計算資源は,STEP 1でスケジューリングサーバから受信したポーリングリスト内の計算資源にポーリングメッセージを送信する. この時, メッセージには送信者の処理能力情報も付加する. ポーリングメッセージを受け取った計算資源は重複実行の判定処理により Accept と Reject のどちらかを返信する. 返信が返ってきた計算資源は以下の処理を行う.

- Reject を受け取った場合,Accept が返るまで, もしくはポーリングリスト上の全ての計算資源にポーリングメッセージを送信するまでポーリングメッセージを送信し続ける
- Accept を受け取った場合は STEP3 に移行する
- すべての返信が Reject だった場合は, 停止状態となる

STEP 3　STEP2 で Accept を受け取った計算資源は, Accept を返信してきた計算資源からタスクの複製をダウンロードし，複製したタスクを重複実行する．重複実行状態になった計算資源は以下の処理を行う

- タスクの重複実行が完了した場合は STEP4 に移行する
- タスクの重複実行中に Cancel メッセージを受信した場合は, タスクの重複実行を中止して STEP2 の最初に戻る

STEP 4　タスクの重複実行を完了した計算資源は,STEP2 で Accept を返信してきた計算資源に Cancel メッセージを送信し,STEP2 の最初に戻る．オリジナルタスク実行中の計算資源が,Cancel メッセージを受信したら, 実行中のオリジナルタスクを中止する．

負荷の少ない計算資源は静的スケジューリングにより割り当てられたタスクを早い段階で処理完了する傾向があり, 逆に負荷の大きい計算資源は処理時間が長くなる傾向がある. つまり, 早い段階で遊休状態になった計算資源は負荷が少ない確率が高いと言える．提案手法では, 負荷の少ない遊休状態の計算資源がタスクを重複実行することで, 負荷の大きい計算資源の処理時間を短縮できる．

4.3　計算資源の状態遷移

提案手法では, 計算資源の状態はおおまかに４つの状態に分類される．

処理状態　静的スケジューリングにより割り当てられたオリジナルタスクを実行している状態

遊休状態　CPU リソースが余っている状態

重複実行状態　複製したタスクを重複実行し, 計算時間の均一化を試みている状態

停止状態　分散計算には参加していない状態

本節では 4.2 節のスケジューリングの流れを元に計算資源がどのように状態遷移するのかを説明する．図 2 に計算資源の状態遷移図を示す.

以下, 状態が遷移するための条件 (1),(2),(3),(4) について述べる．

(1)　処理状態から遊休状態に遷移する条件

- 静的スケジューリングにより割り当てられたオリジナルタスクが処理完了した

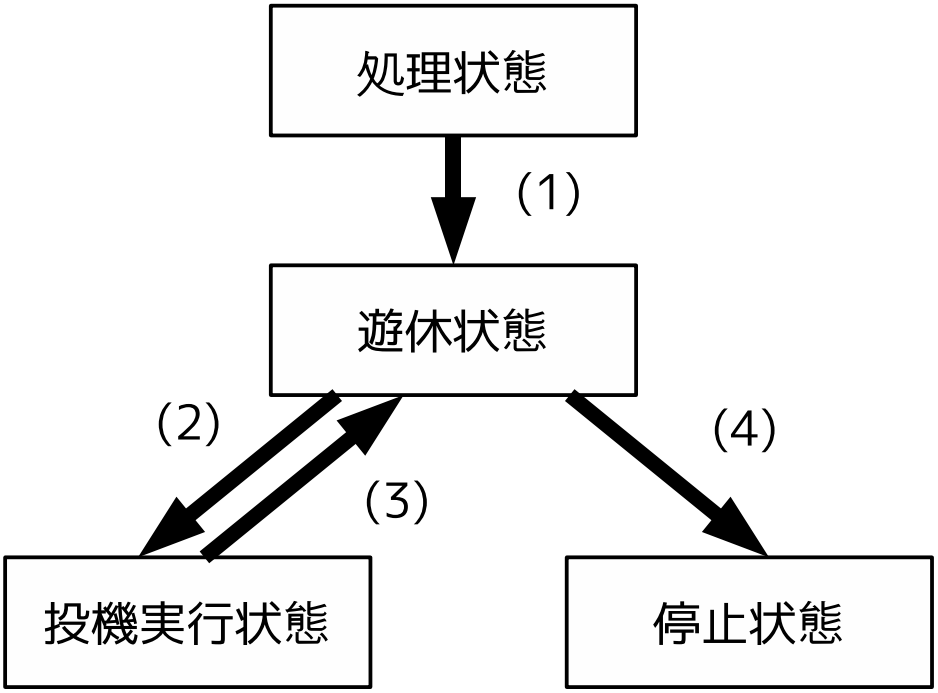

図 2: 計算資源の状態遷移図

- タスクキューが空の状態で, 重複実行を完了した計算資源から Cancel メッセージを受信した

(2) 遊休状態から重複実行状態に遷移するための条件

- STEP2 で Accept を受け取り, 複製されたタスクをダウンロードしてきた

(3) 重複実行状態から遊休状態に遷移するための条件

- 複製されたタスクの重複実行が完了した
- STEP1 でオリジナルタスクを完了した計算資源から Cancel メッセージを受信した

(4) 遊休状態から停止状態に遷移するための条件

- ポーリングリスト内の全ての計算資源にポーリングを終えた

提案手法において全てのタスクが完了することは, すべての計算資源が処理状態ではなくなることを意味する.

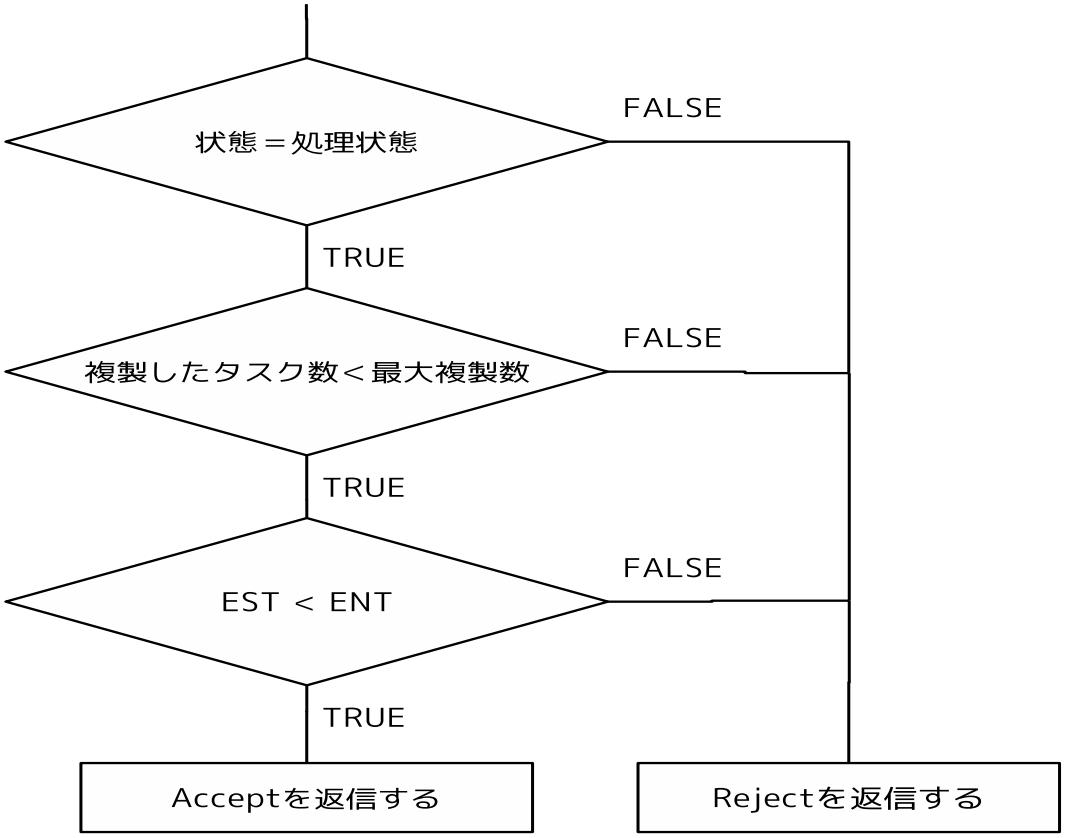

図 3: 重複実行の判定処理の流れ図

4.4 重複実行の判定処理

4.2 節で説明した通りポーリングメッセージを受け取った計算資源は重複実行の判定処理により Accept と Reject のどちらかを返信する. 提案手法では, 重複実行の判定処理を行うことにより, 同じオリジナルタスクの複製回数制限と計算資源の処理能力を考慮した重複実行の制御を行っているといえる. 以降, この制限されたタスクの複製回数の上限値を最大複製回数とする.

本説では, 提案手法における, 重複実行の判定処理の流れと計算資源の処理能力を考慮した重複実行について説明する.

重複実行の判定処理の流れを図 3 に示す. 図 3 から分かるように, ポーリングメッセージを受け取った計算資源は, 以下の 3 つの条件にあてはまる場合のみ Accept を返す.

条件 1 処理状態である

条件 2 既に複製したタスクが最大複製数より小さい

条件 3　EST(Estimated Speculative execution Time) と ENT(Estimated Normal execution Time) を計算し, EST が ENT よりも小さい

条件 2 では, 実行中のオリジナルタスクが大量に複製され続けることを避けるため, 同じオリジナルタスクの複製回数を制限している.

以下, 条件 3 について説明する. ENT(Estimated Normal execution Time) とは, 処理状態の計算資源があとどのくらいで処理中タスクを完了するかを表す予測値である. 処理状態の計算資源は自身の処理能力と処理中タスク残り処理量から ENT を計算する. EST(Estimated Speculative execution Time) とは, ポーリングメッセージの送信元計算資源が処理中タスクを重複実行した場合, あとどのくらいで処理を完了するかを表す予測値である. ポーリングメッセージを受け取った計算資源は, メッセージと共に送られてきた処理能力情報から EST を求める. もし EST<ENT の場合, 重複実行を行うことによりタスクの処理時間が短縮できる可能性が高いと判断する.

提案手法では, 計算資源の処理能力を考慮することによって, 処理時間を短縮できる可能性の高い重複実行を行う. これは, 無駄に終わる可能性の高い重複実行は行わないほうが, 他タスクの重複実行に CPU リソースを提供できるためである. そうすることにより, 負荷の大きな計算資源の処理時間を短縮させることが期待できる.

5 評価

第 5 章では, 異なる処理性能の計算資源が混在し, 分散計算に提供できる処理能力が常に変動する大規模分散計算環境をシミュレートすることにより, 大規模分散環境における提案手法の有効性を評価した.

はじめにシミュレーションモデルについて説明する. 次に, 大規模分散計算環境における並列アプリケーションの実行時間を比較し各タスクスケジューリング手法の動的負荷分散性能を評価する. 今回, 比較対象のタスクスケジューリング手法として, 代表的な集中型手法である適応型手法 [8] と WQR[12] を選んだ. 最後に, 提案手法が計算資源の処理能力を考慮した投機実行を行えているのかを確認する.

5.1 シミュレーションモデル

5.1.1 計算資源のモデル

大規模分散計算を構成する計算資源群には, 様々な処理能力を持つ計算資源が混在する. 計算資源群の不均一性を再現するにあたって, 各計算資源の最大処理能力を Paranhos[12] らのモデルを用いる. Paranhos らのモデルに従えば, 最も性能の低い計算資源の最大処理能力を 1 としたとき, 計算資源群の中には 1,2,4,8,16 の最大処理能力を持つ計算資源が混在することになる.

そして, 分散計算に提供できる処理能力が常に変動する環境を, 文献 [13] に書かれているランダムウォークモデルを用いて再現する. 分散計算に提供できる処理能力の時間変動は次式によって表される.

$$x_n = x_{n-1} + \upsilon \qquad (0 \leq x_n \leq 100) \tag{1}$$

$$y_n = x_n + \omega \qquad (0 \leq y_n \leq 100) \tag{2}$$

y_n は 0 から 100 までの値をとり, ある時刻 n における計算資源 m が提供できる処理能力の割合（%）を表している. ここでの υ と ω は, それぞれランダムウォークモデルにおけるシステムノイズ, 観測ノイズとする. システムノイズ υ は平均が 0, 分散が υ_m の正規分布乱数とする. υ_m は計算資源 m が提供できる処理能力の動的変動の大きさを表しており,υ_m が大きな値ほど提供できる処理能力が短時間で大きく変動することを示している. 同様に観測ノイズ ω は平均が 0, 分散が ω_m の正規分布乱数とする. 以降, この提供できる処理能力が変動することを動的負荷変動と呼ぶことにする.

5.1.2 アプリケーションモデル

本論文では, タスクスケジューリングの対象アプリケーションとして極めて並列度の高い (embarrassingly parallel) アプリケーションを用いる．これは,1 つの並列アプリケーションが非常に多数の独立したタスクによって構成される並列計算問題である. またタスク間に依存関係がないため各タスクを完全に独立して処理することが可能であり, 分散計算性能を引出しやすく, より多くの計算資源を利用することでアプリケーションの実行時間を短縮することができるという特徴がある.

このような特徴を持つ代表的なアプリケーションとしてパラメータスィープ型並列アプリケーションがある．[1] パラメータスィープ型並列アプリケーションは生物情報学, オペレーションズリサーチ, データマイニング, ビジネスモデルシミュレーション, 電気 CAD, フラクタル計算, 画像処理等数多くの分野で利用されている

5.1.3 スケジューラのモデル

スケジューラと計算資源間で行われる 1 回の応答を 1 つのイベントとし, スケジューリング処理に必要なイベント数で, スケジューリング処理に要する処理負荷を定義する. また, スケジューラの処理能力は, 単位時間あたりに処理可能なイベント数で定義する. スケジューラに要求されるイベント数が, スケジューラの処理能力を越えるとそのスケジューラは過負荷状態となり, スケジューリング処理に遅延が生じるとする. スケジューラの過負荷状態で発生する, 1 イベントあたりの処理遅延は, スケジューラの処理能力の逆数とする.

各スケジューリング手法に必要なイベント数を以下のように定める．提案手法では 1 回のポーリングメッセージのやりとりに 1 イベント, 複製タスクの割り当て処理に 1 イベント, 複製したタスクの処理結果返却に 1 イベント必要であるとする．適応型手法は, 計算資源 1 台につき動的情報の収集と再スケジューリング処理にそれぞれ 1 イベント必要とし, 計算資源台数に応じて全イベント数は線形に増加する.WQR では, 計算資源の処理結果返却と新たなタスクの割り当てのための処理にそれぞれ 1 イベント必要であるとする.

提案手法と適応型手法では, 最初にスケジューラが静的スケジューリングによって各計算資源にタスクを割り当てる．この静的スケジューリングでは, スケジューラが各計算資源の処理能力を調査し, その処理能力情報に基づいて各計算資源に割り当てるタスクの数を決定する. 本実験では, 静的スケジューリングに必要とされるイベント数を, 適応型手法が動的負荷分散を 1 回行うために必要なイベント数と同じとした. また, 提案

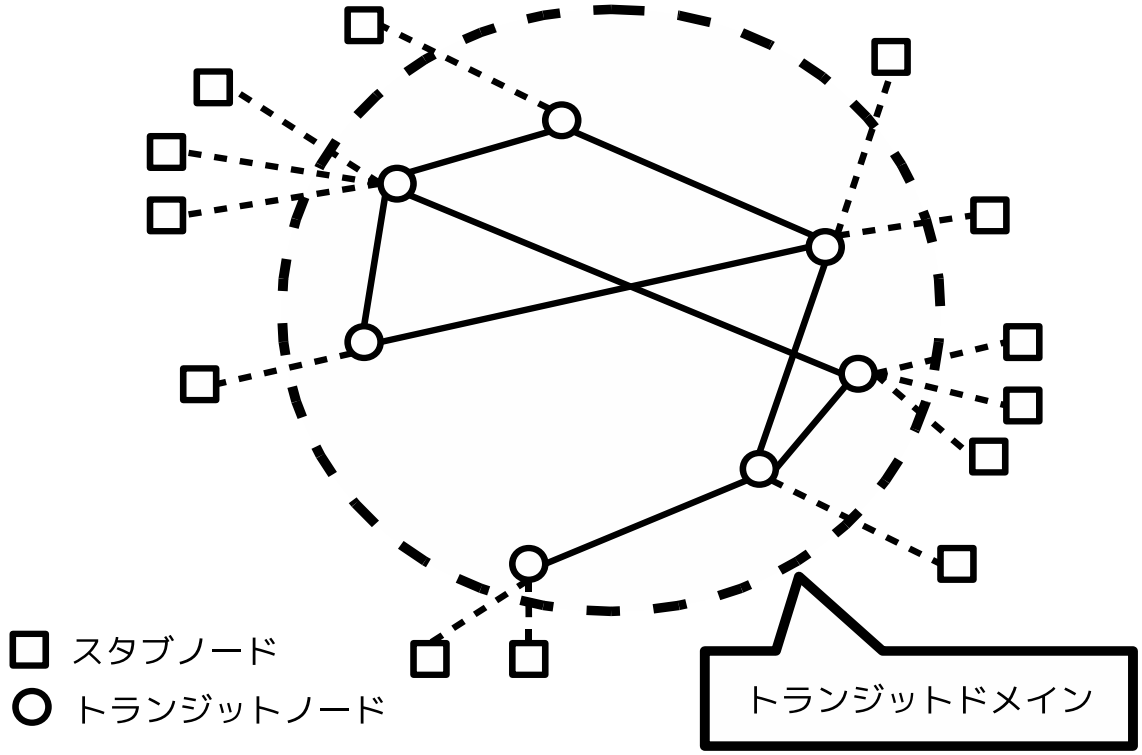

図 4: トランジットスタブ型のネットワークトポロジ

手法と適応型手法は最初にタスクが割り当てられてから, タスクの処理を開始するものとする.

5.1.4 ネットワークモデル

評価実験で用いるネットワークトポロジとして図 4 のようなトランジットスタブ型のネットワークトポロジを想定した. スタブノードから送信されたパケットはトランジットノードを中継して, 目的のスタブノードへと運ばれる.

評価実験では, 同じトランジットドメイン内に 50 のスタブノードが接続されているとする. トランジットノードとスタブノード間リンクを支線 LAN, トランジットノードとトランジットノード間リンクを幹線 LAN とし, 支線 LAN と幹線 LAN では利用可能な帯域幅が異なるものとした. また, 異なるトランジットドメイン間の通信をエリア外通信とし, 同じトランジットドメイン内で通信を行う場合と比べて余分に往復レイテンシがかかるものとした.

パラメータ名	値
計算資源台数	100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600
計算資源の最大処理能力	100, 200, 400, 800, 1600[MFLOPS]
動的負荷変動の発生間隔	10[s]
動的負荷変動 υ_m	π^2, $\pi^2/2$, $\pi^2/3$, $\pi^2/4$, $\pi^2/5$, $\pi^2/6$
動的変動 ω_m	1
利用可能な帯域幅 (幹線 LAN)	1[Gbps]
利用可能な帯域幅 (支線 LAN)	100[Mbps]
エリア外通信の往復レイテンシ増加	10[ms]
アプケーション 1 つあたりのタスク数	10000[tasks]
タスク 1 つあたりの浮動小数点演算命令数	10^9,10^{10},10^{11}[命令/task]
タスクのデータサイズ	100[KBytes]
スケジューラの処理能力	50[event/s]
ポーリングメッセージのデータサイズ	16[Bytes]
適応型手法の再スケジューリング間隔	100[s]

表 1: シミュレーションパラメータ

5.2 評価実験

実験に用いたシミュレーションパラメータを表 1 に示す. これまでの研究の多くは, 計算資源台数が多くても 100 台程度の比較的に小規模な分散計算環境を想定し, 性能評価を行っていた.[8][14][19] 本実験では,1000 台以上の分散計算環境をシミュレートすることで, これまで十分に評価されていなかった大規模分散計算環境における各スケジューリング手法の性能評価を行う. 本実験では, 単位時間あたりに処理可能な浮動小数点演算命令数を処理能力の指標とした. Intel Core2Duo 2.66Ghz の実行性能が 1600MFLOPS であることから [15], 最も高性能な計算資源の処理性能を 1600MFLOPS と設定した. 他の計算資源についても 5.1.1 節で説明した Parahons らのモデルに従い, 計算資源の処理性能を 5 段階に設定した.

タスクひとつあたりの計算量には複数の値を設定し, 粒度の異なる並列アプリケー

ションを実行したときの性能変化を調べる. 粗粒度並列アプリケーションのタスク 1 つあたりの浮動小数点演算命令数として 10^{11} 命令を設定する. 同様に中粒度, 細粒度アプリケーションのタスク 1 つあたりの浮動小数点演算命令数に 10^{10},10^{9} 命令を設定する. また, アプリケーション 1 つあたりのタスク数は 10000 タスクに固定する.

また文献 [16] で利用されているサーバマシンの処理性能を元に, 本評価実験に用いるスケジューラの処理能力を設定した. この文献では,CPU を 2 基搭載するサーバマシンを用いて, 毎秒 100 台の計算資源からの要求を処理している. 評価実験では,CPU を 1 基搭載したスケジューラを想定し, 毎秒 50 回の要求を処理可能であると設定した. 提案手法では, 処理中タスクの最大複製数を 1 とし, 計算資源群の中からランダムに 10 台選んでポーリングリストを作成した. ポーリングメッセージのデータサイズについては文献 [17] を元に設定した. また,WQR の最大複製数は 4 とした.

評価実験ではまず, 各計算資源の最大処理能力と動的負荷変動のパターンをランダムに作成する. 計算資源 m には 6 通りのシステムノイズ υ_m をランダムに設定し, 負荷の変動しやすい計算資源と変動しにくい計算資源の変動パターンを作成する. そして上記のように再現した計算資源のモデルを利用して, 大規模分散環境における提案手法の有効性を評価した.

5.2.1 実験 1：動的負荷分散の性能評価

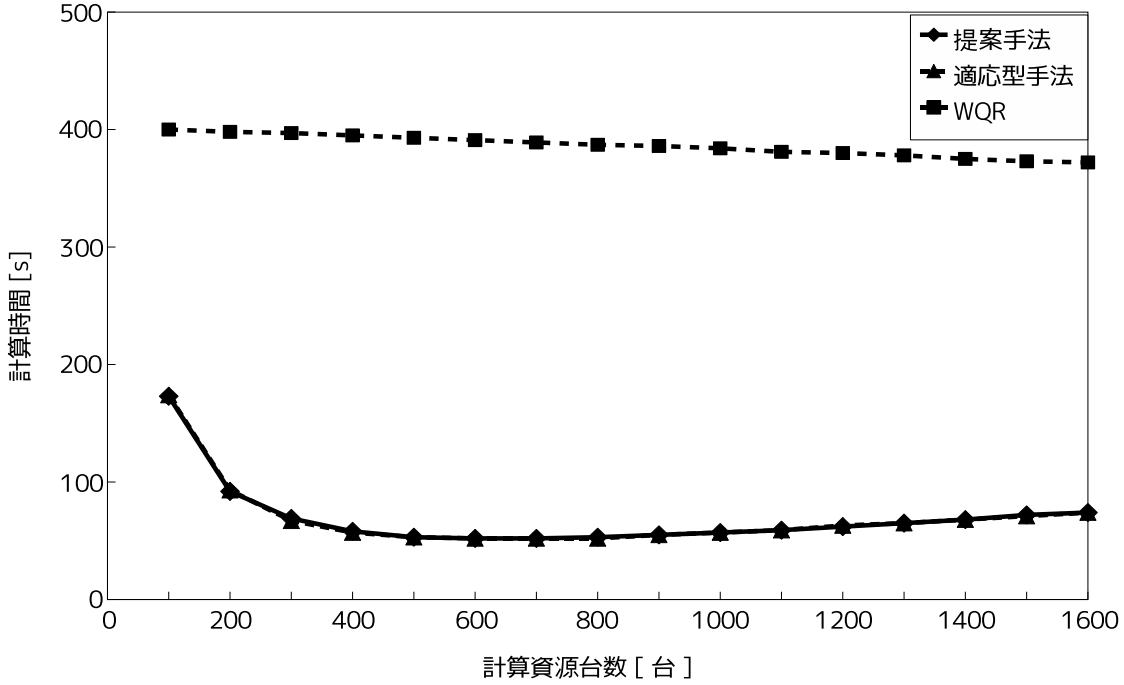

図 5: 動的負荷分散の評価（タスクの計算量：10^9 浮動小数点演算命令）

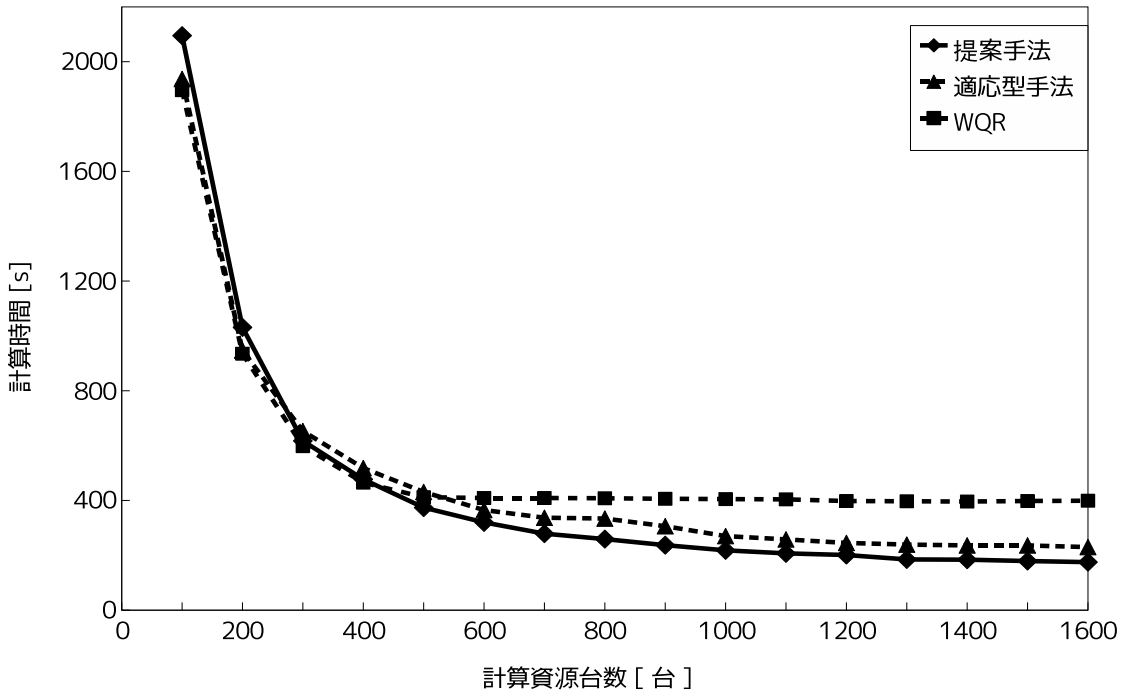

図 6: 動的負荷分散の評価（タスクの計算量：10^{10} 浮動小数点演算命令）

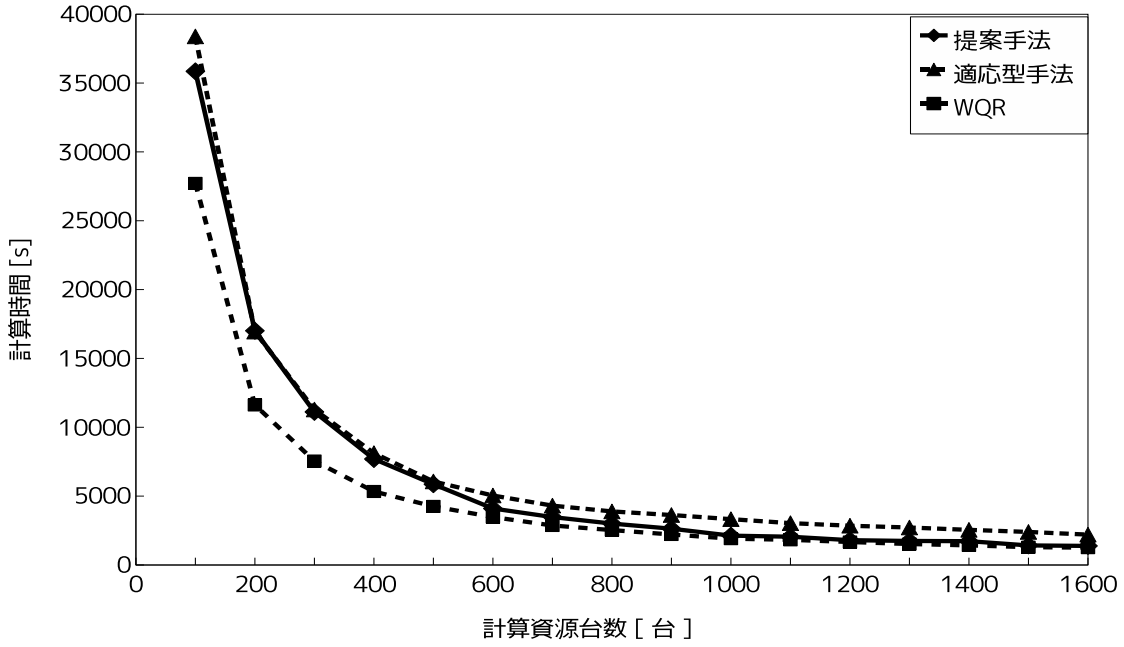

図 7: 動的負荷分散の評価（タスクの計算量：10^{11} 浮動小数点演算命令）

実験 1 では, 提案手法,WQR, 適応型手法の各スケジューリング手法の大規模分散計算環境における動的負荷分散の性能を評価する. 実験 1 では, シミュレーションを 20 回実行し, 並列アプリケーションの平均計算時間をもって, 動的負荷分散の性能を評価した.

評価実験では計算資源群の中からランダムに 10 個選んで, ポーリングリストを作成した. また,WQR の最大複製数を 4 とし, 適応型手法の再スケジューリングの間隔は 100 秒に設定した.

並列アプリケーションの計算時間は, ユーザによって並列アプリケーションの実行が要求されてから, すべてのタスクの処理が完了するまでの時間とする. また, 本評価実験ではジョブスケジューリングによる計算資源の収集と確保に要する時間は無視できるものとし, ユーザからの実行要求が発生するとただちに並列アプリケーションの処理が開始されるものとする. なお, 提案手法と適応型手法の計算時間には, 静的スケジューリングに必要な処理時間も含めるものとする.

図 5, 図 6, 図 7 に各スケジューリング手法を用いたときの並列アプリケーションの計算時間を示す. 図 5, 図 6 図 7 における, タスク 1 つ当たりの浮動小数点演算数は, それぞれ 10^9, 10^{10},10^{11} とする. 以下, 3 つの手法についての評価をしていく.

- WQR の評価
  WQR は細粒度並列アプリケーションの計算時間を示す図 5 では, すべての計算資源台数において他 2 手法よりも大幅に計算時間が長くなっている. これは, タスクひとつあたりの計算量が少ないため, 各計算資源がスケジューリングサーバへタスクを要求する頻度が高くなり, スケジューリングサーバが過負荷状態となっているためである. 過負荷状態では, スケジューリングサーバによる各計算資源へのタスク分配処理に遅延が生じ, 並列アプリケーションの計算時間が増大する. 図 6 の計算資源台数が 600 台以上においても, 同様のことがいえる. 計算資源台数が増加することによって, スケジューリングサーバへの単位時間あたりの要求回数が多くなり, 他 2 手法よりも計算時間が長くなっている. 一方,WQR は粗粒度並列アプリケーションの計算時間を示す図 7 では, すべての計算資源台数において 3 手法の中で計算時間が最短となっている. これはタスク 1 つあたりの処理時間が長い粗粒度並列アプリケーションの場合, 各計算資源がサーバへタスクを要求する頻度が低いため, 計算資源台数が増加してもスケジューリングサーバが過負荷状態となりにくいためである.

- 適応型手法の評価
  適応型手法は図 5 のすべての計算資源台数において, 提案手法と共に最も計算時間が短くなっている. また, 図 6 の 100 台から 300 台の計算資源台数においても, 高い動的負荷分散性能を実現している. しかし, 図 6 の 400 台以上の計算資源台数において, 適応型手法は提案手法に比べて計算時間が長くなっている. これは計算資源台数が増加し, モニタリングコストが増大することによってタスクスケジューリングの精度が低下しているためである. 図 7 において, 他 2 手法よりも適応型手法の計算時間が長くなっているのも, 同様の理由である. 適応型手法は, 細粒度・中粒度並列アプリケーションよりも再スケジューリングの回数が増え, 計算資源台数が増えたことによりモニタリングコストが増大し, タスクスケジューリングの精度が低下したためである.

- 提案手法の評価
  提案手法は, 図 6 の計算資源台数が 200 台以下の場合において, 他 2 手法よりも計算時間が長くなっている. この計算時間の差は, 既存手法が計算資源全体で負荷分散するのに対して, 提案手法が遊休状態になった計算資源だけがタスクの投機実行

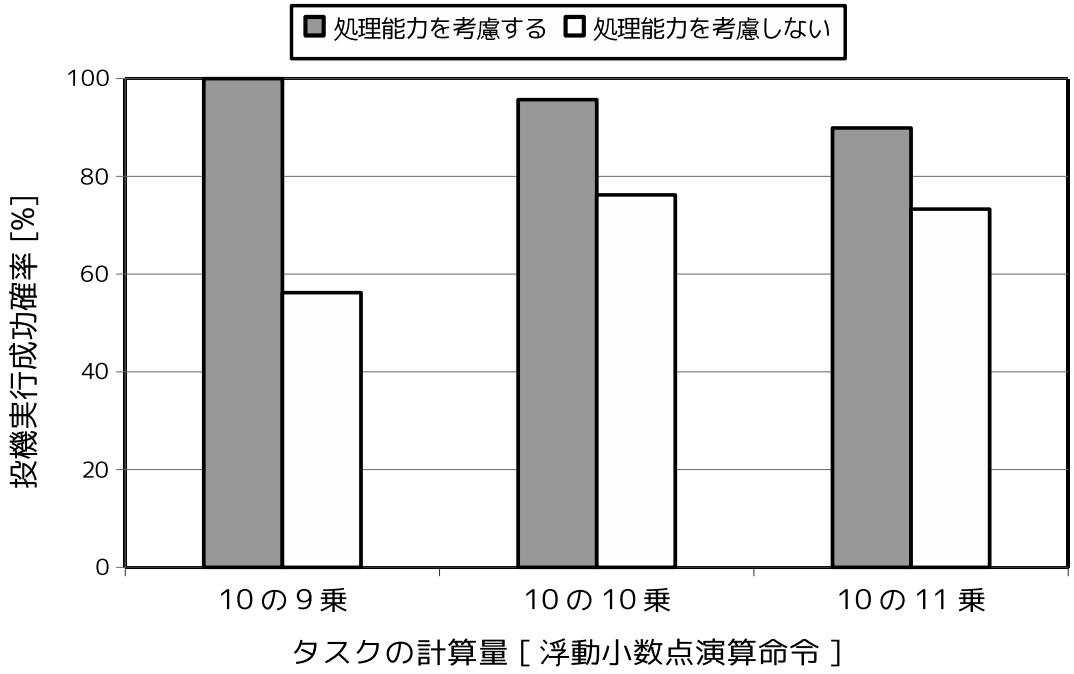

図 8: 投機実行成功確率の評価

によって負荷を分散させているためである. しかし, 提案手法は図 5, 図 6 図 7 からわかるように, 計算資源台数が 1000 台以上の大規模計算環境において, タスクの粒度とは関係なく, 他 2 手法よりも計算時間が短くなっている. これは既存手法がスケジューリングサーバで集中的にタスクの割り当てと実行を管理しているのに対し, 各計算資源が自律的にタスクの割り当てと管理を行うことによってスケジューリング処理を分散化させているためである.

本評価実験の結果より, 提案手法が, 大規模分散計算環境において高い動的負荷分散性能を実現していることが明らかになった.

5.2.2 実験 2：計算資源の処理能力を考慮した投機実行の確認

提案手法は遊休状態になった計算資源がポーリングメッセージを他の計算資源に送信し, ポーリングメッセージを受け取った計算資源は投機実行の判定処理によって Accept もしくは Reject を返すことにより, 投機実行を制御している. この投機実行の判定処理により, 提案手法は計算資源の処理能力を考慮した投機実行を行っている.

提案手法が計算資源の処理能力を考慮した投機実行を行う理由は, 4.4 節で説明したように, 遊休状態計算機がタスクの処理時間を短縮できる可能性の高い投機実行を行う

ためである. ここで, 提案手法において投投機実行によって処理状態計算機のタスク処理時間が短縮されることを投機実行成功とし, 投機実行してもタスク処理時間が短縮できなかったことを投機実行失敗とする.

実験 2 では, 提案手法が処理能力を考慮した投機実行を行うことにより, 投機実行成功確率の高い投機実行を行えているのかを確認した. 本評価実験では, 提案手法の比較対象として,4.4 節の条件 3 を提案手法から外したものを比較対象とする. 以降, 提案手法のことを”処理能力を考慮する” , 条件 3 を提案手法からはずしたものを”処理能力を考慮しない”と呼ぶ. 実験 1 と同じ環境で, 計算資源台数を 1000 台に固定し, タスク 1 つあたりの計算量を変えて, 投機実行成功確率を測定した. 実験 2 では, 投機実行成功確率を２０回測定し, その平均をとった値で評価した. その投機実行成功確率を示したグラフが図 8 である. 以下, 処理能力を考慮しない場合と考慮する場合の投機実行成功確率について評価する.

- 処理能力を考慮しない場合
  処理能力を考慮しない場合は, タスクの 1 つあたりの計算量が変わっても, 処理能力を考慮する場合より投機実行成功確率が低くなっている. つまり, 遊休状態計算機の CPU リソースは無駄になる可能性の高い投機実行に使われていることになる. また, 図 8 のタスク計算量が 10^9 において, 投機実行成功確率が低くなっているのがわかる. これは, タスクの処理時間に対し, 投機実行を行うためのスケジューラのオーバヘッドが大きくなっているためである.

- 処理能力を考慮する場合
  図 8 から, タスク 1 つあたりの計算量が変わっても, 処理能力を考慮しない場合よりも処理能力を考慮したほうが投機実行成功確率が高いことがわかる. しかし, 処理能力を考慮する場合はタスクの 1 つあたりの計算量が増えるにつれて, 投機実行成功確率が下がっていることがわかる. これは, タスク 1 つあたりの計算量が増加するにつれて, 計算資源が投機実行の判定処理を行ってからタスクの投機実行を完了するまでの間隔が長くなっているためである. この間隔が長いほど, 計算資源の提供できる処理能力が大きく変動する可能性があるので,4.4 節の EST,ENT と実際の値にずれが生じる可能性が高くなる. このずれによって提案手法の投機実行成功確率が下がっている.

本評価実験の結果より, 提案手法が処理能力を考慮した投機実行を行うことにより, 投機実行成功確率の高い投機実行を行っていることを確認できた.

6 おわりに

本論文では, スケジューリングサーバが集中的に動的負荷分散を行う従来の方式における問題点を明らかにし, 各計算資源が自律的に投機実行の管理を行うことでタスクスケジューリング処理を分散化させる手法を提案した. 評価実験では, 最大 1600 台の計算資源を利用する大規模分散計算環境をシミュレートし, スケジューリングサーバが集中的に動的負荷分散を行う代表的な既存手法と提案手法を比較評価することにより, 大規模分散計算環境における提案手法の有効性を評価した. その結果, スケジューリングサーバが集中的に負荷分散を行う既存手法では計算資源台数が増えるにつれ, 適切な動的負荷分散が行えなくなっていることを確認した. それに対し, 提案手法では計算資源が 1000 台以上の分散計算環境において, 高い動的負荷分散性能を実現していることが明らかになった. また本論文では, 提案手法が計算資源の処理能力を考慮することで, タスクの処理時間を短縮できる可能性の高い投機実行を行っていることを確認できた. このことから, 提案手法が大規模分散計算環境における動的負荷分散を実現する有効な手法であるといえる.

本論文では, タスクスケジューリングの対象アプリケーションとして多数の独立したタスクで構成される並列度の高いアプリケーションに限定して, 提案手法の評価を行った. 今後は, タスク間に依存関係のある並列アプリケーションについても提案手法の評価を行いたい.

謝辞

本研究に際して, 様々なご指導を頂きました松尾啓志教授, 齊藤彰一准教授, 津邑公暁准教授, 松井俊浩助教に深く感謝いたします. また, 日常の議論を通じて多くの知識や示唆を頂いた松尾・津邑研究室, 齊藤研究室の皆様に感謝します.

参考文献

[1] D. Abramson, J. Giddy, L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid", Parallel and Distributed Processing Symposium, pp520-528(2000)

[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, G. Shao, " Application-level scheduling on distributed heterogeneous networks" , Proc. Supercomputing 1996(1996)

[3] Condor Project Homepage, http://www.cs.wisc.edu/condor/

[4] M. Litzkow, M. Livny, M. Mutka, "Condor-a hunter of idle workstations",Proc. 8th International Conference of Distributed Computing Systems, pp. 104-111(1988)

[5] D. P. Anderson, "BOINC:A System for Public-Resource Computing and Storage", A System for Public-Resource Computing and Storage, 5th IEEE/ACM International workshop on Grid Computing, pp. 4-10(2004)

[6] J. Verbeke, N. Nadgir, G. Ruetsch, I. Sharapov, " Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment", Proc. 3rd International Workshop on Grid Computing(GRID 2002), pp. 1-12(2002)

[7] K. Shudo, Y. Tanaka, S. Sekiguchi, " P3: P2P-based middleware enabling transfer and aggregation of computational resources", IEEE International Symposium on CCGrid 2005, Vol. 1, pp. 259-266 (2005).

[8] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes,G. Obertelli, J. Schopf,G. Shao, S. Smallen,N. Spring,A. Su,D. Zagorodnov" Adaptive computing on the Grid using AppLeS", IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 4, pp. 369-382(2003)

[9] R. Wolski, N. Spring, "Implementing a performance forecasting system for metacomputing: the Network Weather Service", Proc. 1997 ACM/IEEE conference on Supercomputing, pp. 1-19(1997)

[10] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, "SETI@home: An Experiment in Public-Resource Computing", Communications of the ACM, Vol. 45, Number 11, pp. 56-61 (2002)

[11] distributed.net, http://www.distributed.net/

[12] D. Paranhos, D. Silva, W. Cirne, F. V. Brasileiro, C. Grande, "Trading Cycles for

Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids", Applications on Computational Grids , in Proc of Euro-Par 2003, vol. 2790, pp. 169-180(2003)

[13] 北川源四郎, "FORTRAN 77 時系列解析プログラミング", 岩波コンピュータサイエンス (2003)

[14] H. Casanova, A. Legrand, D. Zagorodnov, F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments",in Proc of the 9th Heterogeneous Computing Workshop, Mexico, pp. 349-363(2000)

[15] 姫野ベンチマーク, http://w3cic.riken.go.jp/ HPC/HimenoBMT/

[16] D. P. Anderson, E. Korpela, R. Walton, "High-Performance Task Distribution for Volunteer Computing", 1st IEEE international Conference on e-Science and Grid Technologies, pp. 196-203(2005)

[17] N. G. Shivaratri, P. Krueger, M. Singhal, "Load distributing for locally distributed systems",IEEE Trans. Comp., pp. 33-44(1992)

[18] GIMPS, http://www.mersenne.org

[19] 竹房あつ子, 松岡聡, "Grid 計算環境におけるデッドラインスケジューリング手法の性能", 情報処理学会並列シンポジウム JSPP2001 論文集, pp. 263-270(2006)