大規模計算環境における分散協調型負荷分散手法

村田善智[†1] 稲葉勉[†3]
滝沢寛之[†1] 小林広明[†2]

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，大規模計算に利用しようという研究が数多く行われている．これら遊休計算資源を利用した分散計算環境では，計算資源が提供する処理能力が動的に変動するため，不適切な負荷の割当てによる計算時間の増加が発生する．そのため，非常に多くの計算資源の負荷状況を監視し，適切な量の計算負荷を各計算資源に割り当てる動的負荷分散手法の確立が求められている．本研究では，このような動的負荷分散手法として，分散協調型のスケジューリング機構を用いた動的負荷分散手法を提案する．提案手法では，単一のサーバが行っていた負荷分散処理を並列化し，各計算資源で分散して処理する．分散計算に参加する計算資源が負荷分散の処理も同時に行うため，従来手法に比べ高いスケーラビリティを実現する．多数の計算資源を利用した分散計算環境をシミュレータを用いて実現し，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．その結果，従来手法では計算資源の台数増加により動的負荷分散が有効に機能しなくなったのに対し，提案手法では計算資源の台数に依存せずつねに動的負荷変動に対して有効な負荷分散を実現することが示された．

A Distributed and Cooperative Load Balancing Method for Large-scale Computing Environments

Yoshitomo Murata,[†1] Tsutomu Inaba,[†3] Hiroyuki Takizawa[†1] and Hiroaki Kobayashi[†2]

There are many studies that aggregate idle computing resources connected to networks and exploit their computing power for large-scale distributed computing. In a distributed computing environment with idle computing resources, the computing power devoted by each peer is likely to dynamically change during the computation, and causes load-imbalance that increases the execution time. Therefore, dynamic load balancing by monitoring the load of each peer and appropriately reassigning tasks is required. In this paper, we propose a distributed and cooperative dynamic load balancing method to achieve an appropriate dynamic load balancing among a huge number of peers. The proposed mechanism parallelizes and assigns load balancing processes to all available peers, though the conventional one usually assumes the processes are performed by a single concentrative server. Since each computing resource participating in the distributed computing simultaneously performs both computing and load balancing, the proposed mechanism can achieve higher scalability than the conventional ones. We simulate a distributed computing environment consisting of many peers, and evaluate the dynamic load balancing performance of the proposed mechanism in comparison with that of the conventional ones. The evaluation results indicate that the proposed mechanism can effectively perform dynamic load balancing irrespective of the number of peers, and can accelerate the distributed computing.

1. はじめに

計算機技術の発達により，一般家庭への高性能計算機と高速ネットワークの普及が進んでいる．しかし一般ユーザの多くは，計算機を文章作成ソフトやメールソフトの利用あるいは web の閲覧などにしか利用していない．そのため，高性能な計算機や高速ネットワークなどの資源は一部しか利用されておらず，大部分が遊休状態となっている．そこで，これらの遊休状態にある計算機の処理能力を利用して並列アプリケーションを実行し，大規模計算を実現しようという試みが行われている．このような形の大規模分散計算として，

†1 東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University
†2 東北大学情報シナジー機構
Information Synergy Center, Tohoku University
†3 NTT 東日本宮城支店
NTT East Corporation

一般ユーザが持つ計算資源を利用するボランティアコンピューティングや，組織内の遊休計算資源を利用するデスクトップグリッドなどが提案されており，新たなコンピューティング形態として期待されている[4]．実際，SETI@home[3] や distributed.net[11] などの代表的なボランティアコンピューティングプロジェクトでは，インターネットを介して数万台以上の遊休計算資源を集約し，数十 TFlops を超える処理性能を実現している．

上記のような大規模分散計算環境は，複数の異なるアーキテクチャの計算資源により構成され，多くの利用者により共有される．また，予測不可能な負荷変動の発生によって，計算資源が分散計算に提供できる処理能力は大きく変動する．このような環境で分散計算を効率良く実行するためには，計算機やネットワークの負荷変動を予測し，それらを考慮して個々のタスクや並列アプリケーションを適切な計算機に割り付けるスケジューリングが必要になる．このスケジューリングに関する問題についての様々な研究が行われており[9),10),13)]，実際に大規模分散計算環境を構築するためのスケジューラの実装もいくつか行われている[2),6),12),14)–16),18)]．

これらの大規模分散計算におけるスケジューリング問題には 2 つの側面がある[8)]．1 つ目はジョブスケジューリングと呼ばれる，計算資源の集約と確保に関する問題である．ジョブスケジューリングでは複数のユーザからの並列アプリケーションの実行要求を受け付ける．そして，ネットワーク上の様々な性能や特性を持つ多くの計算資源の中から，各ユーザの並列アプリケーションを実行するのに適した計算資源を検索し，計算資源の確保を行う[13)]．2 つ目はタスクスケジューリング問題，あるいはアプリケーションスケジューリング問題と呼ばれるものである．ここではジョブスケジューリングによって確保した複数の計算資源に対し，アプリケーションを適切に実行するようにタスク割当てを決定する[9),10)]．

これまでに，多数の計算資源を対象とした様々なジョブスケジューリング手法が提案され，その有効性が検証されている[2),12),14),16),18)]．一方，タスクスケジューリングに関しても様々な研究[6),9),10)] が行われているが，そのほとんどは少数の計算資源を利用する場合しか対象としていない．そのため従来のタスクスケジューリング手法では，多数の計算資源を対象とした環境では期待どおりに機能しない．そこで本研究では，多数の計算資源を利用する大規模分散計算環境において有効に機能するタスクスケジューリングの手法を提案する．

Casanova らは文献 6)，9)，10) において，分散計算環境におけるタスクスケジューリングについて述べている．その中で分散計算環境における問題点として，計算に利用する各計算資源が持つ処理能力の不均一性と，計算資源の遊休・非遊休状態や分散計算に提供可能な処理能力などがつねに変動することをあげた．そして，各計算資源の状態を監視し，発生した動的変動に応じてスケジューリングを再度行うことで負荷分散を行うことが有効であると述べた．

Casanova らの手法では，専用のスケジューリングサーバが計算資源の動的情報を一元的に取り扱う，マスタ–ワーカ型の集中管理モデルを用いている．しかし，本論文で想定する大規模分散計算のように，広域ネットワーク上での通信を必要とする多数の計算資源を対象としたスケジューリングには，長い処理時間を必要とする．また，従来用いられてきた集中型の手法では負荷の集中や単一故障点の問題が生じてしまい，適切なスケジューリングを行うことは困難である．すなわち，彼らの手法では台数の限られた比較的小規模な計算環境を想定しており，数千台以上の計算資源を利用する大規模計算環境については考慮されていない．

本論文では，多数の計算資源で構成される分散計算環境上での適切な動的負荷分散を実現するタスクスケジューリング手法を提案する．提案手法では，計算資源の動的情報の収集と動的負荷分散処理を並列分散化し，処理を各計算資源上で独立して実行する．また計算資源自身によって動的負荷変動の検出を行うことで，動的負荷変動を引き金とするイベント駆動型の動的負荷分散を実施する．そして，動的負荷分散に必要なタスクスケジューリング処理は分散計算に参加する計算資源自身によって行われるため，多くの計算資源を利用する環境での適切な動的負荷分散を実現することができる．

以下本論文では，大規分散計算環境における分散協調型のタスクスケジューリングについて次の順序で議論する．まず，2 章では本論文で対象とするアプリケーションを示し，大規模分散環境でアプリケーションを実行するためのスケジューリング問題について述べる．3 章で分散協調型の動的負荷分散について述べ，4 章で性能評価を行う．最後に，5 章で本論文のまとめを行う．

2. タスクスケジューリング

2.1 対象アプリケーション

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，きわめて並列度の高い（embarrassingly parallel）計算問題を用いる．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，より多くの計算資源を用いてタスクを同時に処理することで，並列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ウィープ型並列アプリケーションがあり，各種シミュレーション解析やデータマイニング，画像処理などの分野で数多く利用されている[1),3),17)]．

2.2 タスクスケジューリング問題

タスクを処理する計算資源の性能はそれぞれ異なるため，単位時間あたりに処理できるタスクの数は異なる．そのため，各計算資源に均一にタスクを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，全体の計算時間は最も計算時間が長い計算資源に依存してしまう．そこで各計算資源の計算時間が均一になるように，計算資源の処理能力に応じて割り当てるタスク数を変化させることで，アプリケーションの実行を効率化する．

本論文で扱うタスクスケジューリングの最終目標は，すべての計算資源の計算時間を均一化することで，アプリケーションの実行時間を最短にすることである．そのために，計算資源の計算能力が考慮されていない不適切な負荷割当て状態を検出し，各計算資源の負荷の均一化を実現する．

2.3 関 連 研 究

多数の計算資源を利用した分散計算を実現するためのソフトウェアの実装がいくつか行われている．これらのソフトウェアで用いられているタスクスケジューリング手法は，大きく2つに分類することができる．1つ目は，スーパコンピュータなどの専用並列計算機で培われてきたタスクスケジューリング技術を応用し，大規模環境下でのタスクスケジューリングを行う方法である．この方式では，モニタリングによって計算資源が持つ動的情報を収集し，タスクスケジューリングを実施する．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計算資源の利用効率を高め，高い処理能力を実現する．一方で，スケジューラが各計算資源の状態を監視しなければならないため，管理できる計算資源の台数に制限がある．このように実装されたソフトウェアの代表例として，AppLeS[6)]があげられる．

2つ目は，計算資源の動的情報を利用せずにタスクスケジューリングを行う方法である．この方法では，動的情報を利用する方式に比べ計算資源の利用効率は低下するが，利用する計算資源の台数を増やすことで高い処理能力を実現する．この代表例として，BOINC[2)]やJNGI[18)]，P3[16)]などがあげられる．

以下，それぞれの代表例であるAppLeSとBOINCについて説明する．

2.3.1 AppLeS

AppLeS（Application Level Scheduling）は代表的なタスクスケジューラであり，計算資源の動的負荷情報を考慮したタスクスケジューリングを実現する．このAppLeSとCondor[15)]などの資源集約機構を組み合わせることで，十数台から100台程度の遊休計算資源を利用して分散計算を行うシステムを構築することができる．

AppLeSではアプリケーションの処理を一定の時間間隔で分割し，それぞれの間隔ごとでのタスクのスケジューリングを行う．このようにAppLeSは，直前のタスクの処理状況や計算資源の負荷変動などの情報を利用して定期的に再スケジューリングを行い，計算資源の処理能力に応じた最適なタスク割当て状態を実現する．この方式では，サーバ一定時間ごとに計算資源を監視し，すべての計算資源の情報を一元的に管理することで，タスクスケジューリングを効率的に行うことができる．

その一方，文献6)ではAppLeSのタスクスケジューリングについての問題点がいくつか指摘されている．1つ目の問題点は，AppLeSの再スケジューリングの間隔に関するものである．AppLeSでは一定時間ごとに各計算資源のモニタリング情報を収集し，計算資源間で負荷に偏りがないか調べる．したがって，計算資源の動的負荷変動による負荷の偏りの発生に対し，AppLeSのタスクスケジューリングがどの程度適切に再スケジューリングできるかは，この再スケジューリングの間隔に大きく依存する．計算資源どうしの負荷の比較を短時間で繰り返し行うことで，計算資源の動的負荷変動をすばやく検出し，適切なアプリケーション実行状態を維持することができる．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大きな処理負荷が生じる．さらに，計算資源での負荷変動の有無にかかわらず，つねに計算資源の動的情報を収集しているため，ネットワーク上に大量の無駄なトラフィックを発生させることになる．また，計算資源の台数に応じたモニタリングコストが必要となるため，実現可能な再スケジューリング間隔の最短時間はモニ

タリングコストによって制約される．スケジューリングサーバにおける処理負荷と負荷変動への即応性とのトレードオフから，AppLeS では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．

2 つ目の問題に，計算資源の増加によりモニタリングに要するコストの増大が発生することがあげられる．AppLeS では NWS [19] を利用した計算資源のモニタリング情報の収集や，負荷変動の統計的な予測を行っている．しかし対象となる計算資源が多くなれば，そのモニタリング情報をサーバへ集約するための時間も増大する．モニタリング情報を集約するのに要する時間分だけ，その時刻における計算資源の状態とモニタリング情報に相違が生じるため，その計算資源の動的情報の精度は低下する．このような間違った情報に基づいてスケジューリングを実施すると，逆に計算資源間での負荷の偏りが発生しまうことになり，処理時間の増大につながる．

2.3.2 BOINC

BOINC は，SETI@home [3] や distributed.net [11] などで用いられている分散計算プラットフォームである．BOINC ではタスクスケジューリングに WorkQueue モデルを用いている．すべてのタスクはサーバによって一元的に管理されており，遊休状態にある計算資源は，サーバよりタスクをダウンロードし，余剰計算能力を用いてタスクを処理する．そして，計算資源は処理結果をサーバへとアップロードするとともに，必要に応じて新たなタスクをダウンロードする．この方式では，処理速度の速い計算資源ほど短時間でタスクの処理が完了するため，タスクのダウンロードと処理および処理結果の返却を頻繁に行うことになる．その結果，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，計算資源の性能に応じた適切な負荷分散を実現することができる．また，サーバはタスクの分配と回収のみを行えばよいため，サーバにかかる処理負荷を低く抑えることができる．

しかし WorkQueue モデルによるタスク割当て方式では，スケジューラがタスクを適切な計算資源に割り当てることを保障することができない．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 台を利用して分散計算をすることを考える．1 つのタスクをこれら 2 台の計算資源のどちらかに割り当てることを考えたとき，通常は性能の高い計算資源へタスクを割り当てる．しかし WorkQueue モデルによるタスクの割当てを行う場合，2 台の計算資源の性能によらず，サーバへ先にタスクを要求した計算資源に対し，処理が割り振られることになる．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，スケジューラは性能の低い計算資源へタスクを割り当ててしまい，アプリケーションの実行性能は低くなる．また，タスク割当てを実施するための引き金は計算資源が持っており，サーバが特定の計算資源に意図的にタスクを割り当てることができないという問題がある．そのため，たとえネットワーク上に遊休計算資源が存在していたとしても，その計算資源自らがサーバからタスクをダウンロードしない限り余剰計算能力を利用することができない．

2.4 既存手法の問題点

計算資源に動的負荷変動が生じる環境上での並列・分散計算において，計算資源の性能に応じて動的に負荷を移動させることが有効であることはすでに明らかにされている．しかし，管理するべき計算資源が多くなるほど，各計算資源の性能の調査および負荷の比較には多大なコストが必要となり，動的負荷変動を考慮した負荷分散の実施が現実的ではなくなる．その結果，既存のタスクスケジューリングでは，動的負荷分散による計算資源の利用効率と，同時に利用できる計算資源の台数とのトレードオフが存在している．この問題に対し，本論文では動的負荷分散を実現するための計算資源の動的負荷変動の検出処理，およびタスクの再割当て処理を分散化することで，計算資源の高い利用効率を実現しながら，1 度に数多くの計算資源を利用した分散計算を実現する．

3. 分散協調型スケジューリング

多数の計算資源が存在する分散計算環境では，1 台の管理サーバによってすべての計算資源を一元的に管理することは現実的ではない．そこで提案する動的負荷分散手法では，分散計算環境上の計算資源群をいくつかの部分集合に分割する．そして，それぞれの部分集合に属する計算資源間で計算負荷の比較とタスクの移動を行い，局所的な負荷の均一化を実現する．また，部分集合どうしでも負荷の比較とタスクの移動を行う手段を提供し，最終的に分散計算環境全体での負荷の均一化を実現する．ここで，各部分集合内における負荷の均一化処理において，計算資源間での負荷の比較および適切な移動タスク数の決定には，従来のタスクスケジューリングアルゴリズムをそのまま用いることが可能であるという特徴がある．

以下本章では，分散協調型スケジューリング機構による動的負荷分散手法について説明する．まず分散協調型スケジューラの基本構成と，タスクの管理方法を示す．次いで，分散協調型スケジューラ上での動的負

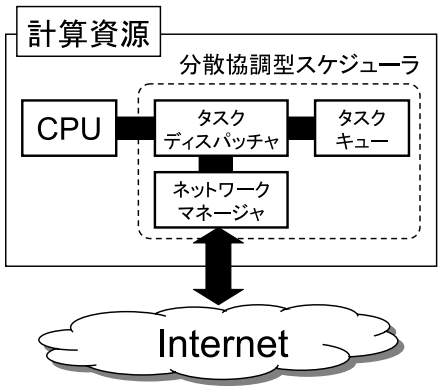

図 1 計算資源上のスケジューリングモジュールの構成
Fig. 1 Scheduling modules in a resource.

荷分散を実現する仕組みを示したのち，多数の計算資源が存在する環境下で負荷分散を実現する仕組みについて述べる．なお以下の議論では，計算資源の発見や集約などのジョブスケジューリングは Condor などの資源管理機構によって行われていることを前提としている．

3.1 分散協調型スケジューラの構成

従来の集中型のスケジューリング手法では，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多数の計算資源が，明確に区別されている．そして処理したいタスクの保持や，計算資源へのタスク割当てなどは専用のスケジューリングサーバによってのみ行われている．これに対し，提案する分散協調型スケジューラでは専用のスケジューリングサーバは存在せず，個々の計算資源がタスクの保持とタスクの処理開始タイミングの管理を行う．本節では，これを実現する分散協調型のスケジューリング機構について説明する．

分散協調型のスケジューリング機構を構成するために，各計算資源内に**図 1** に示すモジュール群を持たせる．各計算資源は OS が持つローカルスケジューラとは別に，分散協調型スケジューリングを実現するための分散スケジューラモジュールを持つ．さらに，分散スケジューラモジュールには並列アプリケーションのタスクを保持するためのタスクキューが存在する．分散スケジューラモジュールのタスクディスパッチャの役割は，タスクキューの管理およびタスクへの CPU 時間の割当てである．分散スケジューラモジュールは，タスクキュー内に未処理のタスクを保持するとともに，計算資源の負荷状態を監視する．タスクディスパッチャは計算資源が遊休状態にあることを検出すると，タスクキューから未処理のタスクを取り出して計算資源が持つ余剰 CPU 時間を割り当てる．

また，分散スケジューラモジュールは計算資源間での通信を実現する．ネットワークマネージャは，計算資源間の通信を仲介し，アプリケーションを実行するための計算資源群のコミュニティを構成する．このコミュニティは，計算資源どうしを論理リンクで接続した P2P ネットワークによって構成することで，高いスケーラビリティを実現する．分散スケジューラモジュール間での通信はネットワークマネージャを介して行われ，計算資源が持つ論理リンクの番号を使って通信相手を指定する．このため，分散スケジューラモジュールが直接通信することができるのは，論理リンクで接続された隣接計算資源までである．

次に，実際の並列アプリケーションの実行の流れを示す．まず，ユーザは解きたい問題をいくつかのタスクへ分割し，それらを各計算資源のタスクキューへと割り当てる．またタスク自体のデータだけではなく，タスクの実行順序や依存情報などのタスクに付随するスケジューリング情報も同時に転送する．計算が開始されると，計算資源はまずタスクキューから未処理のタスクを 1 つ取り出し，自身の CPU を用いて処理を開始する．タスクの処理が完了すると，そのタスクを処理済みのタスクとして保存した後，タスクキューから再び未処理のタスクを取り出し，処理を開始する．また必要に応じて，CPU の負荷情報やタスクの処理状況，タスク 1 つを処理するのに必要な時間などの統計情報の収集と記録を行う．

3.2 局所的なサブネットワークの形成

多数の計算資源が存在する環境での動的負荷分散を実現するために，全計算資源をいくつかの部分集合に分割する．各部分集合内では，1 台の計算資源をマスタとしたマスタ–ワーカ型の構造を持つサブネットワークを形成する．以下，**図 2** に示す 4 台の計算資源から構成されるオーバレイネットワークを例に，サブネットワークの形成方法について示す．計算資源 A，および計算資源 A と論理リンクで接続されている計算資源 B，D によって構成される集合を，計算資源 A を中心とする計算資源の部分集合とする．この計算資源 A を中心とする部分集合は，**図 3** (a) に示すように計算資源 A をマスタとし，計算資源 B，D をワーカとしたサブネットワークを形成する．同様に計算資源 B，C，D を中心とする計算資源の部分集合はそれぞれ，図 3 (b)，(c)，(d) に表されるサブネットワークを形成する．

このとき，各計算資源には必ず自身を中心としたサブネットワークが存在するとともに，ワーカとして他

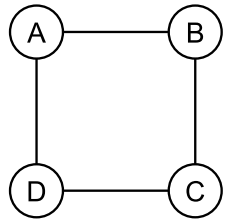

図 2 P2P オーバレイネットワーク例
Fig. 2 An example of P2P overlay network.

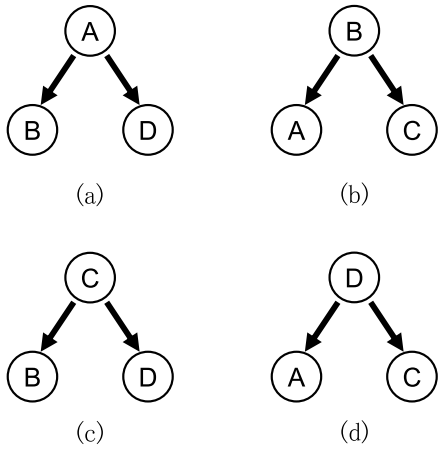

図 3 サブネットワーク群
Fig. 3 Sub-overlay networks.

のサブネットワークへも所属することになる．このように，各計算資源は複数の部分集合に所属している．このような計算資源の仲介により，計算資源の部分集合間での負荷分散を実現する．

3.3 負荷不均衡状態の検出

計算資源は自分を中心とする計算資源の部分集合について，計算資源間で計算負荷に偏りがないか調べることで，動的負荷分散を実施する必要があるかどうかを判断する．従来手法では，タスクスケジューラという第 3 者によって計算資源に動的負荷変動が生じているかどうかが評価されていた．それに対し提案手法では，計算資源自身によって動的負荷変動が生じたかどうかの判断が行われる．これにより，動的負荷変動の発生に応じて動的負荷分散を実施するという，イベント駆動型の動的負荷分散を実現する．

計算資源間での負荷の偏りを評価する基準としては，アプリケーションによって様々な基準を用いることができるが，ここでは一般的な基準である makespan（タスクの処理を開始してからすべてのタスクが終了するまでの時間）を用いて議論する[21)]．ある計算資源について着目したとき，その計算資源に割り当てられているタスクがすべて処理されるまでの時間は，処理するべきタスクが持つ総計算量と，計算資源が単位時間あたりで処理可能な計算量によって決定される．そこで，各計算資源は自分自身の処理能力の情報や割り当てられているタスク数などのスケジューリングに必要な情報を，論理リンクを介して隣接する計算資源に対しあらかじめ転送しておく．そして隣接計算資源からの情報を受け取った計算資源は，その情報を自身のメモリ上にキャッシュしておく．

図 2 の計算資源 A を例に，負荷不均衡状態の検出法を示す．まず，計算資源 A〜D が単位時間あたりに処理可能な計算量は，動的負荷変動によってつねに変化している．しかし，計算資源 A 上の分散スケジューラモジュールは処理中のタスクを監視することで，つねに最新の計算資源 A の処理能力の情報を取得することができる．この最新の計算資源 A の処理能力情報と未処理のタスク数の情報から，計算資源 A でのタスク処理がすべて完了するまでの時間を見積もることができる．また，計算資源 A にはあらかじめ隣接計算資源 B，D の処理能力，およびタスク数の情報がキャッシュされている．計算資源 A 上の分散スケジューラモジュールはそれらのキャッシュされた情報を基に，計算資源 B，D ですべてのタスク処理が完了するまでの時間も同様に見積もることができる．3 台の計算資源間での負荷が均一であれば，予想処理終了時間はほぼ同じになる．ここで，計算資源 A のみに動的負荷変動が生じた場合を考えると，計算資源 A の予想処理終了時間と計算資源 B，D の予想処理終了時間には差が生じる．もし，計算資源間での予想処理終了時間の差があらかじめ決められた閾値を超えるような場合や，タスクを計算資源間で移動させた方が全体の計算時間が短縮できるような場合には，計算資源 A を中心とするサブネットワーク内では負荷の不均衡が生じていると判断する．サブネットワーク内で負荷の不均衡が生じていると判断すると，次節で示す方法で計算資源 A はタスクの移動を行うとともに，計算資源 A の最新の処理能力の情報を計算資源 B，D に転送する．

3.4 負荷の移動

計算資源が自分を中心としたサブネットワーク内で負荷の不均衡が生じていると判断すると，その計算資源は動的負荷分散の処理を開始する．ここで，サブネットワークは動的負荷分散を実施しようとする計算資源をマスタとしたマスタ–ワーカ型の構造を持っているため，従来の集中型の動的負荷分散手法と同一の手順によって，計算資源間でタスクの再分配を行うことができる．これによってサブネットワーク内での負荷の均一化が実現される．

次に，サブネットワーク間での負荷分散の仕組みについて述べる．いま，図 3 (a) のサブネットワーク内において，計算資源 A の負荷変動に起因した動的負荷分散が行われたとする．この動的負荷分散処理によって，計算資源 A だけではなく，計算資源 B，D に割り当てられているタスクの数もそれぞれ変化している．このとき計算資源 B を中心とするサブネットワーク（図 3 (b)）では，計算資源 A と B の予想処理完了時間が変化しているが，計算資源 C の処理時間は変化していない．その結果，前節で示した負荷不均衡状態の検出機構によって，計算資源 B を中心とするサブネットワーク内の計算資源間で動的負荷分散を行う必要があると判断される．また，計算資源 D を中心とするサブネットワークについても同様に動的負荷分散を行う必要があると判断される．このように計算資源間でのタスクの移動を介して，ある計算資源で動的負荷変動が発生したという情報がネットワーク間を伝播する．サブネットワーク内での動的負荷分散，およびサブネットワーク間でのタスクの移動により，分散計算環境全体での適切なタスクの割当て状態を実現する．

3.5 オーバレイネットワーク

各計算資源間を結ぶ通信リンクは，物理的なネットワークに制限されず，任意の計算資源間を論理リンクで接続することができる．そのため，P2P ネットワークには様々なネットワークトポロジを考えることができる．

ここで，各計算資源が持つ論理リンクの数と，動的情報を利用した負荷分散の効率について考える．負荷分散処理では，複数の計算資源の情報を収集し，それらを比較することで，負荷の偏りを検出する．そのため，計算資源が持つ論理リンク数を増やすと，1 度に多くの計算資源間で負荷の比較を行うことができ，負荷分散が効率良く実施できる．しかし逆に論理リンクを増やしすぎると，ネットワークの負荷が増大し，通信性能を低下させる．そのため，負荷分散の効率と情報収集にかかるコスト増大とのトレードオフから，適切な論理リンク数を持つオーバレイネットワークを構築する必要がある．

このような問題をふまえ，提案手法ではリング型のトポロジを持つ P2P ネットワークを利用する．リング型の論理ネットワークを用いることで，すべての計算資源がつねに同数の通信リンクを持ち，なおかつすべての計算資源が同等の関係となることを保証する．また，提案手法の動的負荷分散によって，高負荷な計算資源から低負荷な計算資源へと，リングネットワーク上を未処理のタスクが伝搬する．ただし，リング型ネットワークでは，通信リンクで直接接続されない計算資源に対し情報を転送するには，通信リンクで接続されている計算資源を中継して情報を転送する必要がある．そのため，計算資源の台数が増えると，前節で示した動的負荷分散における未処理タスクが伝播するホップ数が増大し，各計算資源に対するタスクの割当てが適切な状態になるまでの収束時間が問題となる．本研究では，この問題の影響を低減させるために，P2P ネットワークのトポロジとしてコーダルリング[5]を用いる．リング型のネットワークに対して，最長転送路を短くするような，いくつかの冗長リンクを追加する．冗長リンクを追加することで，より少ない伝播ホップ数で負荷の偏りを計算資源全体へと伝えることができるため，最適な負荷状態への収束速度を向上させることが期待できる．

4. 評価実験

本章では，処理能力が動的に変動する多数の計算資源による分散計算をシミュレートすることによって，提案する動的負荷分散手法の大規模分散環境における性能を評価する．

はじめに，大規模分散環境における動的負荷分散の有効性を評価する．静的スケジューリング，WorkQueue 型，AppLeS，および提案手法を用いる場合の並列アプリケーションの実効性能を比較する．

次に，動的情報を利用する手法である提案手法と AppLeS を，動的情報のモニタリングに要するコストに着目して比較する．多数の計算資源を利用する場合の再スケジューリング間隔，および動的情報の精度の問題について議論し，提案手法が AppLeS と比較して大規模環境でも動的負荷分散が有効に機能することを示す．また，提案手法によってタスクが適切に移動していることを確認し，その結果として性能が向上していることを明らかにする．

最後に提案手法のオーバレイネットワークに関し，計算資源の持つリンク数と動的負荷分散の性能についての評価を行う．

4.1 シミュレーションモデル

4.1.1 計算資源のモデル

ネットワーク上には様々な処理能力を持つ計算資源が存在する．ここで，計算資源の処理能力を単位時間あたりに処理可能な浮動小数点演算命令数で定義する．また，計算資源が分散計算に提供可能な処理能力は動的に変動すると仮定する．

まず，各計算資源の最大処理能力を Paranhos[17] らのモデルを用いて表す．Paranhos らのモデルに従え

ば，最も遅い計算資源の処理能力を 1 としたとき，ネットワーク中には 1，2，4，8，16 の処理能力を持つ計算資源が混在することになる．

計算資源の動的負荷変動はランダムウォークモデルを用いて表す[24]．ある計算資源における負荷の時間変動は次式によって表される．

$$x_n = x_{n-1} + v_n (0 \leq x_n \leq 100) \quad (1)$$

$$y_n = x_n + w_n (0 \leq y_n \leq 100) \quad (2)$$

ここで，v_n はランダムウォークモデルにおけるシステムノイズ，w_n は観測ノイズである．システムノイズ v_n は計算資源に発生する負荷変動の大きさを表しており，v_n が大きな値をとるほどその計算資源は性能が短時間で大きく変化することを意味する．y_n は 0 から 100 までの値をとり，計算資源が提供できる計算能力の割合 (%) を表す．y_n が 100 であれば，その計算資源は完全な遊休状態であり，CPU の全処理能力を分散計算に提供することができる状態である．

4.1.2 アプリケーションモデル

評価にはマスタ–ワーカ型の並列アプリケーションを用いる．マスタ–ワーカ型の並列アプリケーションは多数のタスクにより構成される．タスク 1 つがどの程度の計算量を持つかは，そのタスクを処理するために必要な浮動小数点演算数で表現する．並列アプリケーション全体の処理量は，タスク 1 つあたりの計算量と並列アプリケーションが持つタスク数の積によって決定される．また，各タスクが持つ計算量はすべて等しいものとする．

評価実験では，利用する計算資源の台数に比例して並列アプリケーションが持つタスク数を変化させる．すなわち，理想的な負荷分散が行われている場合には，計算時間は計算資源の台数によらずに一定となる．

タスクを計算資源間で移動させるのに必要な時間は，タスクの処理時間に比べて十分に短く，並列アプリケーションの実行には影響を与えないものと仮定する．

4.1.3 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，スケジューラごとに処理に必要なイベントの回数で，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されている．スケジューラに到達するイベント数が処理能力を超えるとそのスケジューラは過負荷状態になり，スケジューリング処理に遅延が生じるとする．過負荷状態のスケジューラで発生する 1 イベントあたりの処理遅延は，スケジューラの単位時間あたり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，隣接計算資源の動的情報の収集のために，リンク数に応じて 2 から 6 イベント必要になる．また，計算資源間でタスクを移動させるために，リンク数に応じて 2 から 6 イベント必要になる．その結果，提案手法全体では最大 4 から 12 イベント必要とする．一方 AppLeS 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のためにあわせて 2 イベント必要とし，利用する計算資源の台数に応じて全イベント数は線形に増加する．WorkQueue モデルでは，計算資源からの処理結果の返却と新たなタスクの割当てのために処理にそれぞれ 1 イベントが必要であるとする．

提案手法と AppLeS では，並列アプリケーションの実行を開始する前に，各計算資源への初期割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，スケジューラが各計算資源の最大処理能力を調査し，その最大処理能力の大きさにに比例した数のタスクを各計算資源に対し割り当てる．本実験では，静的スケジューリングに必要な処理時間は，AppLeS が動的負荷分散を 1 回行うために必要な処理時間と同じとし，利用する計算資源の台数に比例して処理時間が増加するものと仮定する．静的スケジューリングによって各計算資源に割り当てる初期タスク数を決定した後，その情報に基づいて各計算資源にタスクを転送する．これにより提案手法と AppLeS の両手法では，すべてのタスクが各計算資源に分配された状態から，並列アプリケーションの実行を開始する．動的負荷分散による計算資源間でのタスクの移動を，集中型で行うか分散協調型で行うかという点のみが両手法の相違点となる．また，各計算資源へのタスク転送には Multi-Round [7] の手法を用い，タスク転送処理が並列アプリケーションの実効性能に影響しないものとする．

最後に，分散協調型スケジューリングに用いるオーバレイネットワークについて述べる．本評価実験では，各計算資源が 2 本の論理リンクを持つリング型オーバレイネットワークと，各計算資源が 3〜6 本の論理リンクを持つ 4 種類のコーダルリング型オーバレイネットワークを用いる．1 つ目のコーダルリングは，リング型ネットワークを基にして，各計算資源がリング上の最も遠い位置にいる計算資源に対して，1 本のショートカットリンクを持つ構造である．2 つ目のコーダルリングは，リング型ネットワークを基にして，各計算資源が最も遠い位置にいる計算資源までのホップ数が最

表 1 シミュレーションパラメータ
Table 1 Simulation parameters.

パラメータ名	値
計算資源台数	16, 32, 64, 128, 256, 512, 1,024, 2,048, 4,096
計算資源の最大処理性能	100, 200, 400, 800, 1,600 [MFLOPS]
動的負荷変動の発生間隔	10 [s]
動的負荷変動 v_n	π^2, $\pi^2/2$, $\pi^2/3$, $\pi^2/4$, $\pi^2/5$, $\pi^2/6$
動的負荷変動 w_n	1
アプリケーション 1 つ当たりのタスク数	160 – 40,960 [tasks]
タスク 1 つの当たりの浮動小数点演算数	10^{10}, 10^{11}, 10^{12} [命令 / task]
スケジューラの処理能力	50 [イベント/s]

小にするような 2 台の計算資源へのショートカットリンクを持つ構造である．以下同様に，リング型ネットワークを基に 3 本のショートカットリンクを持つネットワーク，リング型ネットワークを基に 4 本のショートカットリンクを持つネットワークをそれぞれ用いる．4.3 節と 4.4 節では，分散協調型スケジューラは冗長リンクを 2 本持ったコーダルリング型を用いた評価実験の結果を示す．4.5 節と 4.6 節では，リング型および 4 種類のコーダルリング型のネットワークを用いた評価実験の結果を示す．

4.2 評価方法

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．

文献 6)，20)，22) などに代表される従来のタスクスケジューリング手法では，多くても 100 台程度の計算資源を用いた性能評価しか行われていない．本実験では最大で 4,096 台の計算資源を利用した並列アプリケーションの実行をシミュレートすることで，従来手法では十分に評価されていなかった大規模計算環境下での動的負荷分散手法の性能評価を行う．

本実験では，Intel Core2Duo 2.66 GHz の実効性能が約 1,600 MFLOPS であることから[23]，最も高速な計算資源の処理性能を 1,600 MFLOPS と設定する．また Paranhos らのモデルに従い，最も高速な計算資源の性能が最も低速な計算資源の性能の 16 倍となるよう各計算資源の最大処理性能を定め，評価実験に用いる．

タスク 1 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．SETI@home[3] ではタスク 1 つあたりの浮動小数点演算数は約 3.9×10^{12} 命令であることから，粗粒度並列アプリケーションにおけるタスク 1 つあたりの浮動小数点演算数として 10^{12} 命令を設定する．また細粒度並列アプリケーションとして，タスク 1 つあたりの浮動小数点演算数に 10^{10} 命令と 10^{11} 命令を設定する．

本実験では 1 つの並列アプリケーションが持つ総タスク数が，利用する計算資源の台数の 10 倍になるように設定する．

文献 4) 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問合せに応答している．このため，本評価実験では CPU を 1 基搭載した計算資源を想定し，毎秒 50 回の問合せに応答可能であると仮定する．本実験条件下における分散協調型スケジューラと AppLeS の初期静的スケジューリング，および AppLeS の 1 回の動的負荷分散処理を行うために必要な時間を 4.1.3 項のスケジューラのモデル定義を基に計算すると，計算資源の台数が 4,096 台の場合で約 164 秒のスケジューリング時間が必要である．

シミュレーションではまず，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生成する．負荷の変動しやすい計算資源と変動しにくい計算資源を表す負荷変動のパターンを作るために，システムノイズ v_n のみに異なる 6 通りの値を設定する．そして，生成した動的な性能変化のパターンを持つ計算資源を利用して，各スケジューリング手法での分散計算の挙動を評価する．以上の流れを 1 回とし，パターンを替えたシミュレーションを計 20 回行う．性能評価には，パターンごとに得られたアプリケーションの計算時間を平均した値を用いる．ここで，並列アプリケーションの実行時間は，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，計算資源で処理されたタスクの結果がサーバへすべて返却されるまでの時間とする．また，本実験では資源集約機構などによる計算資源の収集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すぐにアプリケーションの処理が開始されると仮定する．なお，提案手法および AppLeS における並列アプリケーションの実行時間には，静的スケジューリングと各計算資源へのタスクの分配に必要な処理時間も含めるものとする．

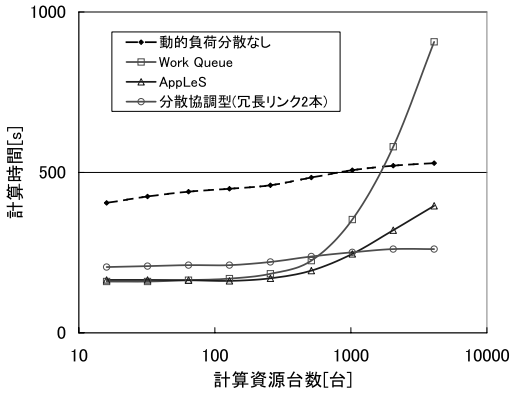

図 4 動的負荷分散の評価（タスクの計算量：10^{10} 浮動小数点演算命令）

Fig. 4 Effectiveness of dynamic load-balancing (Task size: 10^{10} floating-point instructions).

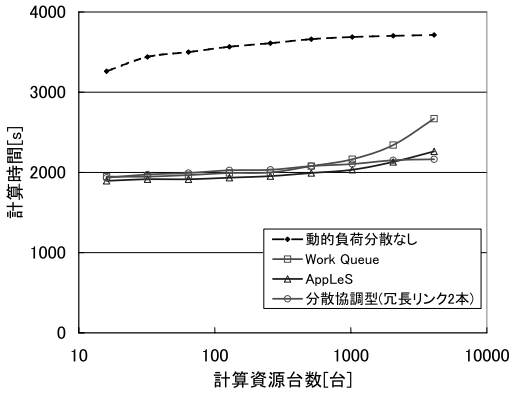

図 5 動的負荷分散の評価（タスクの計算量：10^{11} 浮動小数点演算命令）

Fig. 5 Effectiveness of dynamic load-balancing (Task size: 10^{11} floating-point instructions).

4.3 各負荷分散手法の性能比較

計算資源の実効性能が動的に変動する計算環境で，各スケジューリング手法の負荷分散の性能を評価する．また各手法の特性についてまとめ，提案手法の有効性を明らかにする．本評価実験では，提案手法のトポロジとして冗長リンクを 2 本持つコーダルリングを用い，AppLeS の再スケジューリング間隔を 100 秒と設定した．

図 4，**図 5**，**図 6** に提案手法，AppLeS，WorkQueue 型の各スケジューリング手法を用いたきの並列アプリケーションの計算時間を示す．タスク 1 つあたりの浮動小数点演算数が図 4 では 10^{10} 命令のとき，図 5 では 10^{11} 命令のとき，図 6 では 10^{12} 命令のときの結果をそれぞれ示している．また，アプリケーション開始時の静的スケジューリングのみを行い，動的負荷分散を実施しなかった場合の計算時間も図 4，図 5，図 6 にあわせて示す．

図 4 の計算資源台数が 2,048 台と 4,096 台の場合を除き，すべての実験条件で負荷分散を行わないときの計算時間が最も長くなっている．これは，アプリケーション開始時の静的スケジューリングだけでは，負荷変動によって各計算資源の実効性能と割り当てられているタスク数との比例関係が崩れ，計算資源間で割り当てられたすべてのタスクの処理が完了までの時間に大きな差が生まれるためである．動的負荷分散を行わない場合にはタスクを処理しない計算資源が発生し，アプリケーション全体の実効性能が低下する．これに対し提案手法，AppLeS，WorkQueue 型の計算資源の実効性能に応じて割り当てるタスク数を決定するスケジューリング手法では，各計算資源でのタスクの処理時間を均一化することができるため，アプリケーションを短時間で完了させることができている．

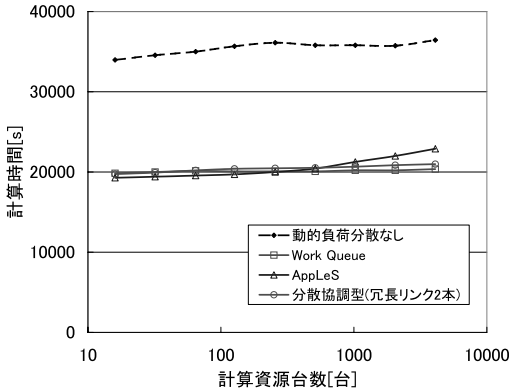

図 6 動的負荷分散の評価（タスクの計算量：10^{12} 浮動小数点演算命令）

Fig. 6 Effectiveness of dynamic load-balancing (Task size: 10^{12} floating-point instructions).

次に WorkQueue 型の手法について見てみると，図 6 ではすべての計算資源台数で同じ計算時間を示しており，計算資源台数に比例した実効性能を実現していることが確認できる．同じく図 4，図 5 の WorkQueue 型の手法の結果を見てみると，図 4 では計算資源台数が 128 台以下の場合，図 5 では計算資源台数が 256 台以下の場合で計算資源台数に比例した実効性能が得られていることが確認できる．しかし図 4 の計算資源台数が 256 台以上での部分と図 5 の計算資源台数が 512 台以上での部分では，計算資源が増えるほど WorkQueue 型の手法を用いたときの計算時間は増大している．特に図 4 の計算資源台数が 2,048 台と 4,096 台のときには，静的スケジューリングのみで動的負荷分散を行わなかった場合よりも計算時間が長くなる結果となった．図 4 や図 5 の結果のように，計

算資源台数が多い領域で WorkQueue 型の手法の実効性能が低下してしまうのは，スケジューリングサーバによる各計算資源へタスク分配処理が間に合わないためである．一方，図 6 は粗粒度並列の特徴を持つアプリケーションを用いた実験結果であり，計算資源がサーバへタスクを要求する頻度が低いため，より多くの計算資源を利用した並列アプリケーションの実行が可能になっている．文献 2)–4) でも示されているとおり，WorkQueue 型のタスクスケジューリングはタスク 1 つの処理時間が長い粗粒度並列アプリケーションを多数の計算資源を利用して処理する用途に適しているといえる．また WorkQueue 型を用いて粒度の細かい並列アプリケーションを実行する場合には，計算資源の台数が増加したときのサーバへの負荷の集中が問題となり，高い実効性能を実現できないことが明らかになった．

AppLeS を用いた実験では，計算資源台数が 256 台以下のときには一定の計算時間でアプリケーションが終了している．図 4，図 5，図 6 より，並列アプリケーションの粒度にかかわらず同様の傾向を確認することができる．このことから AppLeS は，256 台までの計算資源を利用した並列アプリケーションの実行において適切な動的負荷分散を実現し，計算資源の台数に応じた性能向上を実現することが確認できる．しかし，計算資源の台数が 512 台以上になると動的情報の収集と再スケジューリングのコストの増加により，台数が少ないときと同じ性能を達成できないことが示された．

図 4，図 5，図 6 の実験結果では，提案手法を用いたときのアプリケーションの計算時間は計算資源台数によらずほぼ一定の値であり，並列アプリケーションの粒度によらずに同様の傾向を示している．さらに，AppLeS とは異なり，計算資源台数が一定以上になっても動的負荷分散の性能低下が生じることはなく，より多くの計算資源を効率的に利用した並列アプリケーションの実行が可能である．一方，計算資源の台数が少ない場合，提案手法を用いたときの計算時間は，AppLeS を用いたときの計算時間よりも数十秒から数百秒ほど長い．この計算時間の差は，提案手法が完全分散型の負荷分散を行うためのに発生するスケジューラのオーバヘッドや，提案手法が局所的な負荷分散しか行わないことによって生じる誤差によるものである．しかし計算資源の台数が多い環境であれば，計算資源台数を増やすことによって得られる台数効果でこのオーバヘッドを打ち消すことができる．

以上の実験結果より，提案手法の大規模計算環境の動的負荷分散における優位性が明らかになった．AppLeS の特徴は計算資源の動的情報を利用して動的負荷分散を行うことで，実行効率の高い並列計算を実現できることである．しかし AppLeS の動的負荷分散手法では，利用する計算資源の台数に応じて負荷分散のコストが増加する．そのため計算資源台数が増加すると，動的情報を利用した動的負荷分散を行う利点が，動的負荷分散のコストによって相殺されてしまう．一方，WorkQueue 型では実行する並列アプリケーションのパラメータを適切に設定する必要がある．並列アプリケーションのパラメータが適切でない場合，WorkQueue 型ではサーバの処理能力がボトルネックとなり，計算資源の台数が増加しても全体の性能が向上しない．これらに対し，提案手法では負荷分散処理を分散方式で行うため，計算資源の台数が増えても負荷分散処理自体が全体の性能向上のボトルネックにはならない．そのうえ，AppLeS と同様に，計算資源の動的情報を利用した動的負荷分散を行うことが可能であり，分散計算の実行時間を並列アプリケーションの粒度によらずに削減することが可能である．

4.4 動的情報を利用した動的負荷分散手法の評価

本節では計算資源の動的情報を利用した負荷手法である提案手法と AppLeS の手法について，動的負荷分散の性能を比較する．

図 7 に AppLeS と提案手法を用いたときの並列アプリケーションの実行時間を示す．ここで，提案手法ではオーバレイネットワークのトポロジとして冗長リンクを 2 本持つコーダルリングを用いている．図 7 では，タスク 1 つあたりの計算量を 10^{11} 命令としたときの結果を示している．AppLeS の再スケジューリング間隔は 100 秒，500 秒，1,500 秒と設定している．一方，提案手法では各計算資源が 1 つのタスクを処理するたびに動的負荷分散を行うか判断するため，AppLeS のような再スケジューリング間隔の設定を行う必要はない．

図 7 より，AppLeS では再スケジューリングの間隔を短く設定するほど短時間で計算時間が終了し，再スケジューリング間隔の増加とともに全体の処理性能が低下することが分かる．これは，再スケジューリング間隔が短いほど，計算資源の動的負荷変動による負荷の偏りを素早く発見，均一化することができるためである．しかし，現実には間隔を短くするほど，スケジューリングサーバにおける処理負荷が増大するため，設定可能なスケジューリング間隔は制限される．また，計算資源の台数を増加させると，スケジューリングサーバでの処理負荷増大と情報取得に要する時間増大による精度低下によってアプリケーションの実行

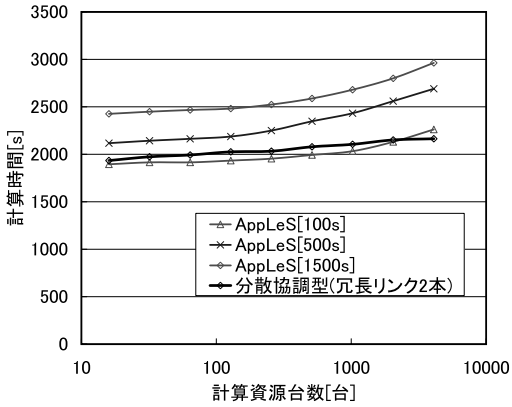

図 7 再スケジューリング間隔と負荷分散能力
Fig. 7 Effectiveness of load-balancing versus rescheduling interval.

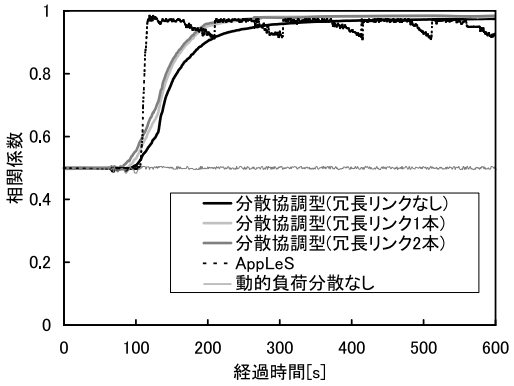

図 8 相関係数の変化
Fig. 8 Changes in convergence.

時間が大幅に増加し，動的負荷分散がうまく機能していないことが分かる．

一方提案手法では，計算資源の台数が増加しても計算時間はほぼ一定である．今回の実験では 2 本の冗長リンクを持つコーダルリングを用いているため，提案手法では論理リンクで接続された 4 台の隣接計算資源の情報のみを収集するだけで動的負荷分散を行うことができ，計算資源の総数によって動的負荷分散処理自体が影響されることがない．そのため提案手法では，最短再スケジューリング間隔には計算資源の台数による制限が生じないため，大規模計算環境において有効に動的負荷分散が行えることが確認できる．

次に，再スケジューリングを行う間隔の決定の仕方と動的負荷分散の性能に関して議論する．AppLeS では一定時間ごとに再スケジューリングを行っているため，適切な再スケジューリング間隔をどのようにして決定するかが重要な問題とされている[6]．一方提案手法では，一定時間ごとに再スケジューリングを行うのではなく，計算資源の負荷変動を検出し，そのつど再スケジューリングを行う．そのため，AppLeS のように適切な再スケジューリング間隔をあらかじめ決めなくても，分散協調型スケジューラによって適切なタイミングで動的負荷分散処理が行われる．図 7 の実験結果から，提案手法を用いたスケジューリングを行ったときのアプリケーションの実行時間は，AppLeS の再スケジューリング間隔を 100 秒とした場合の実行時間より長いが，再スケジューリング間隔を 500 秒とした場合の実行時間よりは短くなった．このことから提案手法では再スケジューリング間隔に準最適値が自動的に選択されていることが確認できる．よって，計算資源の動的負荷変動を検出することを，再スケジューリングの実施条件にすることは有効であるといえる．

以上のことから提案手法では，計算資源の台数と再スケジューリング間隔のとり方に関係なく，つねに適切な負荷分散が行えることが明らかになった．

4.5 動的負荷分散によるタスク移動の確認

本節では，各シミュレーション時刻における計算資源の実効性能と，その計算資源に割り当てられているタスク数との相関係数の時間変化により，動的負荷分散処理が適切に行われていることを確認する．相関係数を用いることで，実行する並列アプリケーションの特性に依存せず，動的負荷分散によってタスクの移動が適切に行われていることを定性的に評価することができる．高い処理能力を持つ計算資源に対し多くのタスクが割り当てられていれば，相関係数は 1 に漸近する．

図 8 に分散協調型スケジューラと AppLeS を用いて動的負荷分散を行ったときの相関係数の時間変化を示す．また，動的負荷分散を行わなかったときの相関係数の時間変化についても図 8 にあわせて示す．図 8 では，計算に用いた計算資源が 4,096 台，タスク 1 つあたりの浮動小数点演算数が 10^{11} 命令のときの結果を示している．AppLeS の再スケジューリング間隔は 100 秒と設定した．

提案手法と AppLeS では計算資源の最大処理性能に基づいて静的スケジューリングを行っており，各計算資源の実効性能と静的スケジューリングによって割り当てられたタスク数は必ずしも比例しないため，アプリケーション開始直後の相関係数が 1 になるとは限らない．アプリケーション開始直後の静的スケジューリングのみが行われた状態での相関係数は，図 8 からわかるように約 0.5 である．また，動的負荷分散を行わない場合相関係数は時間が経過しても約 0.5 で変化していない．このことから本評価実験の条件下では，

相関係数の値が 0.5 に近いほど動的負荷分散処理による計算資源間でのタスク移動が行われていないと判断することができる．一方，分散協調型スケジューラと AppLeS による動的負荷分散を行うとことにより，相関係数が 0.5 から 1 に漸近することが確認できる．この相関係数の増加は計算資源間でタスクが移動するために生じるものであり，動的負荷分散処理に起因する計算資源間でのタスク移動が生じていることが確認できる．

図 8 からはまた，分散協調型スケジューラと AppLeS のそれぞれの相関係数の時間変化の仕方に特徴があることが確認できる．AppLeS では動的負荷分散処理を 1 回行うと，相関係数が瞬時に 1 に近い値となる．しかし AppLeS の手法では一定期間は動的負荷分散は行われないため，計算資源の動的負荷変動によって相関係数が徐々に減少する．AppLeS を用いた場合には，再スケジューリング間隔である 100 秒を周期とする相関係数の増減が生じている．一方，分散協調型スケジューラでは各計算資源が自律的に動的負荷変動に応じた動的負荷分散を行う．提案する動的負荷分散手法では局所的なタスク移動を繰り返し行うため，AppLeS と比べて緩やかな相関係数の変化となる．分散協調型スケジューリングにおいて計算資源が持つ論理リンクによって，相関係数がどのように変化するかを評価すると，計算資源が持つ論理リンク数が多いほど相関係数が急激に変化していることが分かる．このことから計算資源 1 台が持つ論理リンク数が多いほど，提案する動的負荷分散処理が迅速に行われることが確認できる．

4.6 オーバレイネットワークトポロジの評価

本節では，分散協調型スケジューラのオーバレイネットワークとしてコーダルリングを用いることの有効性を評価する．また，各計算資源が持つ論理リンクの本数が動的負荷分散の性能にどのように影響するかを議論する．

各計算資源が持つ論理リンク数を変化させながら，分散協調型の動的負荷分散処理を行ったときの並列アプリケーションの計算時間を**図 9** に示す．図 9 では，横軸に各計算資源が持つ論理リンクの本数をとり，縦軸に計算時間をとっている．タスク 1 つあたりの浮動小数点演算数は 10^{11} 命令とする．

図 9 より，計算資源が持つ論理リンクの本数が多いほどアプリケーションの計算時間が短くなるという結果が得られた．また計算資源の台数が多いほど，論理リンクの本数の増加による実行時間の短縮割合が大きくなっている．これは，1) 計算資源が持つ論理リンク数が多いほど，1 度の負荷分散処理でより多くの計算資源間でタスクの計算時間を均一化することが可能なためであることと，2) 冗長リンクを持つことでオーバレイネットワーク上での任意の計算資源間のネットワーク長が短くなり，全計算資源の計算時間が均一になるために必要になる負荷移動の回数が少なくすむためであると考えられる．このことから，提案手法では計算資源が持つ論理リンク数を増やすことで，計算時間の不均一状態をより短い時間で解消することができることが確認できる．特に多くの計算資源を利用する場合ほど，論理リンクの本数が動的負荷分散の性能に対して大きな影響を与えることが示された．一方，図 9 において論理リンクが 2 本から 3 本になったときの計算時間の短縮量に比べ，論理リンクが 5 本から 6 本になったときの計算時間の短縮量の方が小さい．この結果から各計算資源が持つ論理リンク数をある一定数以上に増やしても，動的負荷分散の性能にはほとんど影響しないことが分かる．逆に論理リンク数を増やしすぎると，計算資源やネットワークに対し過剰な負荷の発生を招き，アプリケーション全体の実効性能を低下させる恐れがある．

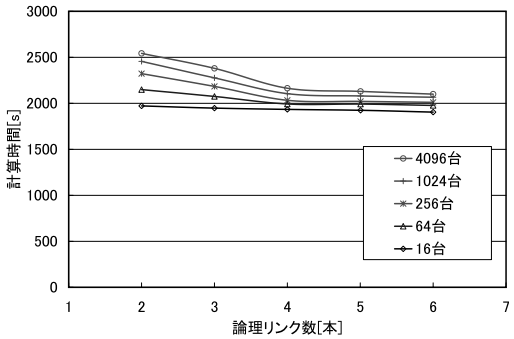

図 9 論理リンク数と計算時間
Fig. 9 Effectiveness of redundant links.

以上の評価結果から，リンク数を増やすことによって得られる動的負荷分散の収束時間の短縮と，計算資源やネットワークに生じるオーバヘッドの増加とのトレードオフにより，各計算資源が持つべき最適な論理リンク数を決めることができる．本評価実験の条件下では，各計算資源にたかだか 2 本の冗長リンクを加えるだけで負荷分散の収束時間を大幅に短縮することが可能であり，集中型の動的負荷分散手法を用いた場合と同等の計算時間を実現することができた．このことから分散協調型の動的負荷分散を行ううえで，計算資源が複数の論理リンクを持つコーダルリング型のオーバレイネットワークを利用することは有効な手段である．また各計算資源が持つ論理リンク数が全計算資源

数と比べて非常に少数だとしても，動的負荷分散処理をより効率的に実施することが可能であることが示された．

5. おわりに

本論文では，大規模分散計算環境におけるタスクスケジューリングの問題について述べ，分散協調型の管理機構を持つ動的負荷分散手法を提案した．これまで大規模分散計算環境では，動的情報を利用することによる効率的な負荷分散と，利用する計算資源の台数の増加がトレードオフの関係にあった．提案手法では負荷分散処理を並列分散化し計算資源自体に負荷分散処理をさせることを考え，1) 分散計算環境上の計算資源をいくつかの部分集合に分割し，部分集合内および部分集合間の動的負荷分散を実施し，2) 各計算資源が自律的に動的負荷変動を検出し，イベント駆動型の動的負荷分散を実施することで，大規模環境下での適切な動的負荷分散を実現した．

評価実験では最大 4,096 台の計算資源を用いた大規模分散計算をシミュレートし，提案する動的負荷分散手法の有効を評価した．その結果，動的負荷変動が存在する大規模分散計算環境では，動的負荷分散を行うことで並列アプリケーションを短時間で計算可能であることが示された．

動的負荷分散の実施方法として集中型のタスクスケジューリングを用いた場合，計算資源の増加にともない動的負荷分散が効率的に行えなくなり，大規模環境では集中型の手法では期待する性能を実現できなかった．それに対し，提案手法では計算資源の台数が増加しても動的負荷分散を効率的に実施することができた．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．

今後は提案した分散協調型のスケジューリング機構を持つミドルウェアの実装を行う．そして，実装したミドルウェアを用いて実際に並列アプリケーションを実行し，その有効性を評価する予定である．また，本論文では，ネットワークがボトルネックにならないと仮定して評価実験を行った．しかし実際の分散計算環境では，計算資源どうしをつなぐネットワークの性能がボトルネックになる可能性が高い．そのため，ネットワークの性能を考慮して提案手法の評価を行い，その問題点を明らかにする必要がある．そして最終的に，ネットワーク性能を考慮したうえで動的負荷分散を行う分散協調型のタスクスケジューラを用いた，高性能な大規模分散計算を実現する．

謝辞 本研究の一部は，文部科学省科研費特定領域研究（18049003），および総務省特定領域重点型研究開発（061102002）の支援を受けている．

参考文献

1) Abramson, D., Giddy, J. and Kotler, L.: High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?, *International Parallel and Distributed Processing Symposium* (*IPDPS*), pp.520–528 (2000).
2) Anderson, D.P.: BOINC: A System for Public-Resource Computing and Storage, *5th IEEE/ACM International workshop on Grid Computing*, pp.4–10 (2004).
3) Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M. and Werthimer, D.: SETI@home: An experiment in Public-Resource Computing, *Comm. ACM*, Vol.45, No.11, pp.56–61 (2002).
4) Anderson, D.P., Korpela, E. and Walton, R.: High-Performance Task Distribution for Volunteer Computing, *1st IEEE International Conference on e-Science and Grid Technologies*, pp.196–203 (2005).
5) Arden, B.W. and Lee, H.: Analysis of chordal ring network, *IEEE Trans. Comput.*, Vol.30, No.4, pp.202–206 (1986).
6) Berman, F., Wolski, R., Casanova, H., et al.: Adaptive Computing on the Grid Using AppLeS, *IEEE Trans. Parallel and Distributed Systems*, Vol.14, No.4, pp.369–382 (2003).
7) Bharadwaj, V., Ghose, D. and Mani, V.: Multi-installment load distribution in tree networks with delays, *IEEE Trans. Aerospace and Electronic Systems*, Vol.31, No.2, pp.555–567 (1995).
8) Casanova, H. and Berman, F.: A decoupled scheduling approach for the grads program development environment., *Proc. 2002 ACM/IEEE Conference on Supercomputing* (2001).
9) Casanova, H., Hayes, J. and Yang, Y.: Algorithms and Software to Schedule and Deploy Independent Tasks in Grid Environments, *Proc. Workshop Distributed Computing, Metacomputing, and Resource Globalization* (2002).
10) Casanova, H., Legrand, A., Zagorodnov, D. and Berman, F.: Heuristics for Scheduling Parameter Sweep Applications in Grid Environments, pp.349–363 (2000).
11) distributed.net. http://www.distributed.net/
12) Fedak, G., Germain, C., Neri, V. and Cappello, F.: Xtremweb; A generic global computing system, *Proc. 1st IEEE/ACM International Symposium on Cluster Computing and*

the Grid, pp.1–12 (2002).
13) Galstyan, A., Czajkowski, K. and Lerman, K.: Resource Allocation in the Grid Using Reinforcement Learning. http://www.isi.edu/lerman/papers/papers.html
14) GPU. http://sourceforge.net/projects/gpu/
15) Litzkow, M., Livny, M. and Mutka, M.: Condor - A Hunter of Idle Workstations, *Proc. 8th International Conference of Distributed Computing Systems*, pp.104–111 (1988).
16) Shudo, K., Onishi, J., Tanaka, Y. and Sekiguchi, S.: P3: P2P-based Middleware Enabling Transfer and Aggregation of Computational Resources, *IEEE International Symposium on Cluseter Computing and the Grid 2005* (*CCGrid 2005*), Vol.1, pp.259–266 (2005).
17) Paranhos, D., Cirne, W. and Brasileiro, V.: Trading Cycles for Information: using Replication to Schedule Bag-of-Tasks Applications on Computational Grids, *International Conference on Parallel and distributed Computing* (*Euro-Par*), pp.169–180 (2003).
18) Verbeke, J., Nadgir, N., Ruetsch, G. and Sharapov, I.: 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).
19) Wolsk, R., Spring, N. and Peterson, C.: Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service, *Supercomputing '97: Proc. 1997 ACM/IEEE conference on Supercomputing* (1997).
20) Yang, Y. and Casanova, H.: A Multi-Round Algorithm for Scheduling Divisible Workload Applications: Analysis and Experimental Evaluation, Technical Report of Dept. of Computer Science and Engineering, University of California CS20020721 (2002).
21) 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, No.SIG 18 (ACS 16), pp.92–114 (2006).
22) 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，情報処理学会電気通信処理学会並列シンポジウム JSPP2001 論文集，pp.263–270 (2006).
23) 姫野ベンチマーク．http://w3cic.riken.go.jp/HPC/HimenoBMT/
24) 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).

(平成 19 年 6 月 11 日受付)
(平成 19 年 12 月 4 日採録)

村田　善智

昭和 55 年生．平成 15 年東北大学工学部地球工学科卒業．平成 17 年東北大学大学院情報科学研究科修士課程修了．東北大学大学院情報科学研究科博士課程在学中．Grid コンピューティング，ボランティアコンピューティングおよび P2P コンピューティングに関する研究に従事．

稲葉　　勉（正会員）

昭和 44 年生．平成 7 年東北大学大学院情報科学研究科修士課程修了．同年日本電信電話株式会社入社．NTT ネットワークサービスシステム研究所，NTT 東日本研究開発センタを経て，現在 NTT 東日本宮城支店に勤務．東北大学大学院情報科学研究科博士課程在学中．VoIP をはじめ P2P，Grid コンピューティングの研究開発に従事．

滝沢　寛之（正会員）

昭和 47 年生．平成 11 年東北大学大学院情報科学研究科博士後期課程修了．博士（情報科学）．平成 11 年新潟大学総合情報処理センター助手，平成 15 年東北大学情報シナジーセンター助手，平成 16 年同大学大学院情報科学研究科講師．現在，高性能計算システム，コンピュータアーキテクチャ，およびそれらの応用に関する研究に従事．平成 18 年船井情報科学奨励賞受賞．電子情報通信学会，IEEE 各会員．

小林　広明（正会員）

昭和 63 年東北大学大学院博士課程修了．同年東北大学助手．平成 3 年東北大学講師．平成 5 年東北大学助教授．平成 13 年東北大学教授（情報シナジーセンター副センター長兼任）．平成 18 年より NII 客員教授併任．コンピュータアーキテクチャ，並列処理システムとその応用に関する研究に従事．工学博士．IEEE Senior Member，ACM，電子情報通信学会各会員．