NAIST-IS-DT0161004

博士論文

高スループット計算を指向したグリッド環境構築における タスク実行支援フレームワークに関する研究

蟻川 浩

2006 年 9 月 14 日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

蟻川 浩

審査委員：　砂原 秀樹 教授
山口 英 教授
藤川 和利 助教授

高スループット計算を指向したグリッド環境構築における タスク実行支援フレームワークに関する研究*

蟻川 浩

内容梗概

本論文は、シミュレーション実施者による対話的操作を可能にするためのタスク実行支援フレームワークを導入した高スループット計算向けグリッドの構築について論じたものである。広域に分散した計算機の資源を用いて構築されるグリッドはパラメータスィープ型シミュレーションのような、高スループット計算に適している。高スループット計算によるシミュレーションを実行したい要求が高まる一方で、シミュレーション実施者の知識や経験に基づいてプログラムの実行を柔軟に変更できる仕組みもまた求められている。

本論文では、高スループット計算向けグリッドにおける現状と問題意識を把握するとともに、シミュレーション実施者の意思決定に基づくシミュレーション実施の必要性についてシステム構築の観点から議論する。そして、シミュレーション実施者の知識や経験に基づいてプログラムの実行制御やパラメータの変更を可能にするためのタスク実行支援フレームワークを提案する。提案するフレームワークでは、まず、タスク実行の依存関係を考慮してタスクフローシステムに着目し、実行途中にプログラムに与えるパラメータの変更を可能にするタスクフロー差分実行機構を提案する。次に、大量タスクの実行を考慮して、特定のタスクを優先的に実行することを可能にする機構を提案する。これらにより、シミュレーション実施者の知識や経験に基づいたシミュレーションを実現できるようになる。

本論文で提案する技術の有効性を検証するために、タンパク質立体構造予測とマルチエージェントシミュレーションによる託児所配置問題を取り上げ、それぞれのグリッド環境を構築した。タンパク質立体構造予測向けグリッド環境では、タスクフロー差分実行機構により立体構造予測結果が改善されることを確認した。また、マルチエージェントシミュレーションによる託児所配置問題向けグリッド環境では、特定のタスクを優先的に実行することを可能にする機構により最適解の探索にかかる時間が改善されることを確認した。

* 奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文, NAIST-IS-DT0161004, 2006年9月14日.

以上の取り組みから、本論文で提案するタスク実行支援フレームワークによってシミュレーション実施者の対話的操作に基づいた高スループット計算向けグリッドを構築することに寄与できる。

キーワード

グリッド, 高スループット計算, タスク実行支援, 継続的改善, 対話的動作

Studies on the Task Control Support Framework in Grid for High Throughput Computing*

Hiroshi ARIKAWA

Abstract

This thesis describes the Grid for high throughput computing with decision making support framework. Recently, Grid that is one of the distributed computing systems can aggregate large amounts of computing power, and it enables virtual organizations to share geographically distributed resources. Grid is suitable for high throughput computing. Scientists and engineers are being watched to do simulation for high throughput computing using Grid environment. However, task control based on the experience in the user is difficult at present. The purpose of this study is to construct the computing grid for the high throughput computing with decision making support framework.

I discuss the concept of the Grid for high throughput computing and conventional Grid technology. From a practical point of view, the important task are user interaction mechanism. Scientists and engineers have to do execution applications for a simulation by trial-and-error process. I propose the task control support framework based on knowledge and experience which the user has. Proposed framework is composed of flexible taskflow mechanism and priority based task submitting mechanism. Flexible taskflow mechanism enables to change the parameter under execution of the process and to restart the process. Priority based task submitting mechanism has the function which gives priority to the execution of the specific task.

I apply the task control support framework to protein structure prediction system. By using proposed framework, scientists of protein structure prediction could be easily to obtain the suitable result by the continual improvement. Also, I had implemented a multi-agent based day care center allocation problem system using the task control support

* Doctor Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT0161004, September 14, 2006.

framework. By using priority based task submitting mechanism, economists and policical scientists could search the optimum result quickly.

By applying the task control support framework by this thesis, I can conclude that proposed mechanisms are functionally linked the interaction-based Grid for high throughput computing. This indicates that this work contributes to a grid for high throughput computing.

Keywords:

Grid, High Throughput Computing, Task Control Support, Continual Improvement, Interaction Process

目 次

図 目 次

表 目 次

第1章

序論

1.1. グリッドによる問題解決環境の構築

1.1.1 複雑な問題に対するシミュレーションによる挑戦

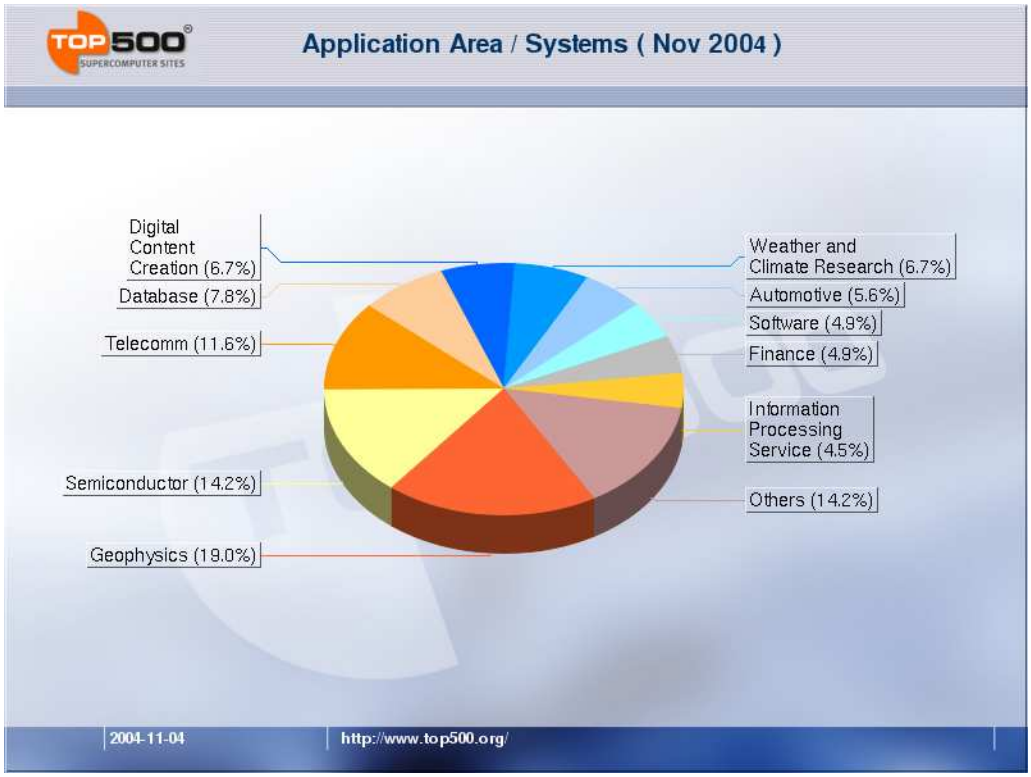
コンピュータが登場したころから、研究者や技術者はコンピュータによるシミュレーションに興味を持っており、シミュレーションによって様々な問題解決を行ってきた。

世界における高性能計算システムの動向を調査する目的で、年に2度、世界中の高性能計算システムの上位500システムの番付を作成しているTOP500プロジェクト[1]がある。TOP500プロジェクトでは高性能計算システムの番付作成時にプロセッサの種類や利用されている分野など高性能計算に関係する様々な情報を収集し、統計情報として公開している。2004年11月における高性能計算システムの分野別利用動向の統計 (図1.1) によると、地球物理学、気象予測、自動車開発、ナノテクノロジーを含む半導体材料開発といった科学技術分野で積極的に使われていることが伺える。また、金融、情報システム、コンピュータグラフィックスの映画への応用など、科学技術以外の分野で高性能計算システムが必要だと叫ばれるようになった。いまや、科学技術に限定されることなく、産業界のあらゆる分野でシミュレーションが用いられる時代となった。

日本国内においても様々な観点からシミュレーションの可能性について研究開発が行われている。バイオサイエンス、医学、薬学と情報技術の融合を実現するための研究プロジェクトであるバイオグリッド・プロジェクト[2] [1]、ナノテクノロジーと情報技術の融合を実現する

[1] http://www.top500.org

[2] http://www.biogrid.jp



出典: TOP500 November,2004

図 1.1　高性能計算システムの分野別動向

ための研究プロジェクトである超高速コンピュータ網形成プロジェクト NAREGI(National Research Grid Initiative)[3] [2]、地球変動予測のシミュレーションを行うための研究プロジェクトである地球シミュレータ[4] は代表的な研究開発プロジェクトである。これらのプロジェクトでは、大規模、マルチスケール、マルチフィジクスをキーワードとするシミュレーション環境の実現を目標として掲げている。具体的には、実験による研究開発の限界に対してシミュレーションを中心とする情報技術で支援し、新たな研究開発手法を提案することである。

このことから、いかに速く計算するか、いかに大量の計算を処理するかといったシミュレーション実施者の要求はとどまることを知らない。

[3] http://www.naregi.org

[4] TOP500 ランキングにおいて 2002 年 6 月から 2004 年 6 月まで世界最速のコンピュータの地位を保っていた

1.1.2 グリッド技術を利用した問題解決環境への期待

学術研究や製品開発におけるシミュレーションは、それに携わる研究者や技術者のツールである。ところで、計算機を駆使して複雑な問題に挑戦するためには、情報技術の知識が不可欠である。シミュレーション実施者の多くは、複雑な問題に挑戦するための知識には長けているが、コンピュータの取り扱いやシミュレーションツールを使いこなす知識が長けているとは限らない。このような状況を考慮に入れたシミュレーション実施環境を問題解決環境 (Problem Solving Environment; PSE) と呼ぶ。

問題解決環境は、シミュレーション実施者がシミュレーションに専念できるようした計算機環境のことである。シミュレーション実施者には、ある問題を解くために特化したユーザインターフェイスを提供する。一方、計算機環境は問題を解くために必要な機能と処理能力を準備する。近年、問題を解くために必要な機能と処理能力を提供するものとして、グリッドが注目されている。

グリッド [3] とは、広域ネットワークに接続された計算資源を仮想化し、必要に応じて計算資源を仮想計算機や仮想組織という形で提供するものである。ここでいう計算資源とはパーソナルコンピュータやスーパーコンピュータといった汎用的な計算機だけではなく、データベース、センサ、実験装置などが含まれる。現在ではあらゆる装置が何らかの手段を通じてインターネットに接続できることから、インターネットに接続できる装置を計算資源ととらえることができる。グリッドが普及した背景には、20 世紀末に登場したインターネットの高帯域化が進んだこと、浮動小数点演算処理能力がギガオーダのプロセッサを搭載したパーソナルコンピュータが増加したこと、ソフトウェア技術の発展による質の高いオープンソースソフトウェアが流通したことが挙げられる。これにより、大規模な並列分散処理環境を安価で構築できるようになった。

グリッドという言葉を提唱し、グリッドの牽引役として第一線で活躍している米国アルゴンヌ国立研究所兼シカゴ大学教授の Ian Foster 博士は、以下の項目を満たしたものがグリッドであると文献 [4] で述べている。

1. 集中管理されていない分散した資源のコーディネート
2. オープンスタンダードなプロトコルやインターフェイスの利用
3. 単純には得られない質の高いサービスの提供

World Wide Web(WWW) による情報の公開や共有、File Transfer Protocol(FTP) を利用したファイルの交換、電子メールによる意思疎通は主として人間の情報交換を目的とし

て、インターネットに接続されている計算機を利用して行う。グリッドの登場は、科学技術シミュレーションや大容量データの検索といった大規模計算処理、特殊な望遠鏡や顕微鏡から得られる情報のリアルタイム加工といった、計算機資源の連携動作を可能にした。特に、グリッドが社会的に印象づけたのは、我々が普段利用しているパーソナルコンピュータが大規模計算処理の一プロセッサとして連携動作することで科学技術シミュレーションの可能性を見出したことである。このように、グリッドはインターネット上に存在する計算資源を必要なときに必要なだけ使える枠組みとして注目されていることが、ここ数年における問題解決環境の構築事例で明らかになっている。

日本における構築事例としては、例えば、気象予報アプリケーションのグリッド環境への適用 [5] や、皮膚画像を利用した皮膚状態解析エンジンのグリッド環境への適用 [6] などがある。

また、グリッドの概念を生み出した米国では、グリッドによる問題解決環境の構築は国家戦略のひとつとして捉えられている。例えば、高エネルギー物理学、化学、物質化学、気象などの研究開発を促進させるための DOE Science Grid[5] や、National Aeronautics and Space Admistration(NASA) が中心となって航空宇宙関係の研究開発のために構築した Information Power Grid[6] などがある。また、欧州では、グリッド向けアプリケーションツールキットの研究開発プロジェクトとして、The GridLab Project[7] が存在する。

このように、グリッドは問題解決環境を構築するための基盤技術として注目されているが、注目の過程において、グリッドの長所がより明確になってきた。とりわけグリッドの構成要素であるハードウェアによるところが大きい。スーパーコンピュータや PC クラスタのシステム構成は、ノードの追加が困難であるが、すべてのノードのプロセッサ処理能力が同一であるとともに、ノード間通信が同一速度である。一方、グリッドのシステム構成はノードの追加が容易だが、様々なプロセッサ処理能力を有するノードで構成されるとともに、ノード間の通信速度がノードごとに異なる。この違いから、グリッドの処理能力はノードの数が同じ場合にはスーパーコンピュータや PC クラスタに比べて劣るが、容易にノードの数を増加させることができるため、単純な処理の並列化においては、時にスーパーコンピュータや PC クラスタを超える性能を発揮する。グリッドでは、大規模連立一次方程式の並列解法のようにノード間で相互にデータの交換を必要する処理は不得意だが、パラメータスウィープやパターンマッチングといったノード間でデータの交換が不要な処理は得意とする。このような特徴から、パラメータスウィープやパターンマッチングなど、

[5] http://www.doesciencegrid.org

[6] http://wwww.ipg.nasa.gov

[7] http://www.gridlab.org

プロセッサの台数が確保できないと処理時間が短縮できない課題に対してグリッド技術が使われるようになった。

1.1.3 大量タスク実行による問題解決環境構築の課題

シミュレーション実施者が問題解決環境を必要とする理由は、シミュレーションによって何らかの結果を得るためである。例えば、グリッド技術による医薬開発や生体機能解明の支援は、創薬における生産プロセスの短縮に寄与する。ナノオーダの分子挙動シミュレーションのグリッド技術による支援は物質科学や材料科学の基盤研究の推進と、半導体材料や新素材の開発に対する開発期間の短縮に関係がある。また、自然科学分野以外の例としては金融業界が挙げられる。金融業界では、顧客から預った資産を確実に運用するために、多種多様な金融商品のリスク計算と日々蓄積される膨大な金融市場動向データの統計処理を行わなければならない。その処理をこなすためにグリッド技術が使われている。これらのシミュレーションはパラメータスウィープにもとづく最適解探索手法が土台となっている。グリッド技術の普及によって大量タスクの実行が可能になったため、全探索に近いかたちでの解探索手法が現実的な手法になった。

大量タスクによるシミュレーションが現実的なものになったとはいえ、大量タスク実行による問題解決環境をグリッド技術を用いて構築する際に考慮すべき課題がある。例えば、複雑なシミュレーションになるほど、複数のプログラムを有機的に実行して問題解決の糸口を発見することが望まれる。具体的にはパラメータの変更や異なるプログラムの利用が考えられる。また、既存のグリッドミドルウェアで構成した問題解決環境では、シミュレーションを実施するに必要な計算機資源の提供と、必要最低限の操作インターフェイスを提供しているに過ぎない。シミュレーション実施者の意思によるシミュレーション実施とグリッドミドルウェアとの間には隔たりがある。この隔たりによってシミュレーション実施者は問題解決の糸口を探すことのほかに、操作インターフェイスを駆使するといった労力が必要になる。

1.2. 研究の目的

学術研究や製品開発における競争の激化において、問題解決環境の処理能力を高めることが要求されている。従来、計算機環境そのものの処理能力を高めることで対応してきたが、限界に達している。

シミュレーション実施者は解かなければならない問題に対して、解決の糸口を発見することが目標である。目標を達成するためには、シミュレーション実施者の意思決定による操作を問題解決環境に取り込む必要があると考えた。

本論文は、高スループット計算向けグリッド環境における大量タスク実行に適した問題解決環境の構築を目的として、タスク実行の観点からシミュレーション実施者の意思決定による操作と大量タスク実行を連携させるフレームワークを提案する。そして、シミュレーション実施者の意思決定に着目することによる効果や将来展望について論じる。

1.2.1 本研究がもたらす効果

今日において、グリッドは科学技術計算等のシミュレーションを中心に、様々なかたちで広く利用されている。本論文では、グリッドの能力を十分に生かすことが可能である高スループット計算に焦点を絞った。従って、高スループット計算向けグリッド環境の構築において本研究の成果は寄与できる。

また、本論文では適用例としてタンパク質立体構造予測とマルチエージェントシミュレーションによる託児所配置問題を取り上げる。タンパク質立体構造予測では、複数のプログラムを使ってシミュレーションする場合を想定している。また、マルチエージェントシミュレーションによる託児所配置問題では、大量タスク実行を想定している。いずれも、本論文で提案するフレームワークがシミュレーション実施者の意思決定に寄与することを示す。また、問題解決環境を構築する上でシミュレーション実施者の意思決定が大切であることを示す。

1.3. 本論文の構成

本論文は以下の章で構成される。

第２章では、関連研究として既存のグリッドミドルウェアについて概説するとともに、本研究の位置付けを示す。

第３章では、利用者の意思決定と高スループット計算向けグリッドの関係について論じる。まず、高スループット計算の概略を説明する。そして、高スループット計算とシミュレーション実施者の意思決定との関係、シミュレーション実施者の意思決定が高スループット計算にもたらす効果について述べる。最後に、意思決定を考慮にいれた問題解決環境の構築についてタスク実行の観点から解決策を示す。

第4章では、高スループット計算向けグリッドにおけるタスク実行支援フレームワークについて説明する。シミュレーション実施者のシミュレーション実施における動作を分析するとともに、タスク実行支援フレームワークの必要性を示す。次に、タスク実行支援フレームワークの全体像、構成要素について説明する。最後に、タスク実行支援フレームワークに導入したタスクフロー差分実行機構とパラメータリスト優先実行機構について説明する。

第5章では、高スループット計算におけるデスクトップ計算機の利用について論じる。デスクトップグリッド環境に高スループット計算を適用するには、ノード選抜が必要であることを示す。そして、高スループット計算に適したノード選抜手法を提案する。

第6章と第7章では、第4章で提案するフレームワークを適用した問題解決環境構築の事例を述べるとともに、本論文で提案するフレームワークの有用性を示す。第6章では、タンパク質立体構造予測を取り上げ、タスク実行支援フレームワークを導入することによるシミュレーション結果の改善について論じる。特に、タスクフロー差分実行機構の効果について示す。第7章では、マルチエージェントシミュレーションによる託児所配置問題を取り上げ、タスク実行支援フレームワークを導入することによるパラメータの探索時間の改善について論じる。特に、パラメータリスト優先実行機構の効果について示す。

最後に、第8章で研究の総括を行う。本研究で得られた知見を述べるとともに、問題解決環境構築における新たな研究方針を提示する。

第2章

関連研究

2.1. はじめに

製品開発や学術研究を円滑に進めるべく、近年では、様々なかたちの問題解決環境の構築が行われている。本章では、グリッドを中心とする問題解決環境を構築するための技術について概説する。

2.2. タスク管理ミドルウェア

2.2.1 複数拠点を接続するためのミドルウェア

Globus Toolkit

グリッドコンピューティングの先導役を担った技術である Globus Toolkit[7] は Foster らによって開発されたミドルウェアである。現在におけるグリッド技術の標準化作業は Globus のアーキテクチャを基礎として議論されている。

複数の拠点のコンピュータをネットタスクで結ぶことで大規模計算を行う枠組みは、Smarr と Catlett による Metacomputing[8] や、DeFanti らによる I-WAY[9] が提案されているが、Globus では、仮想組織 (Virtual Organization) を構成することを目的として、インターネットのような広域ネットタスクを意識してミドルウェアが設計されている。具体的には、スーパーコンピュータといった大型計算機や PC クラスタなどの計算機群に対して計算機資源の統一的な利用を可能するために公開鍵暗号基盤による認証環境 (Grid Security Infrastructure;

GSI) を提供していること、GSI に基づいたタスクの割り当てとデータ転送を可能にしていること、1 つのデータ転送に対して複数のコネクションを張る (パラレルデータ転送) 機能や複数の独立した計算機に分散転送する (ストライピング転送) 機能などをもった転送技術を提供していること、各計算機の状態を収集する機能を提供していることが挙げられる。

PUNCH

Kapadia らは大学内にあるコンピュータを利用した分散計算システムとして Purdue University Network-Computing Hubs(PUNCH)[10] を設計、実装している。

PUNCH では、シミュレーションツールそのものの特徴、およびシミュレーションツール実行時の特徴を利用者の要求としてとらえ、必要な計算機資源を提供する Demand-based Computing System を実現している。また、Web ブラウザから計算機環境へアクセスする仕組みを設けている。

UNICORE

ヨーロッパにおける広域分散計算資源に対するタスク管理ミドルウェアとしては、Fujitsu Labs Europe が中心となって UNICORE[11] が開発され、EUROGRID[1] で採用されている。

UNICORE は研究者および技術者のために、インターネットを通じて広域分散されているスーパーコンピュータのリソースをシームレスにアクセスするための枠組みを提供している。特に、インターネットを経由して遠隔地から計算リソースにアクセスされることを考慮し、Secure Socket Layer(SSL) プロトコルを利用して Gateway と呼ばれるモジュールに接続する方式を採用しているとともに、ファイアウォールとの親和性を図っている。また、PC クラスタ専用タスク管理ミドルウェアとの仲介を図るためのモジュールを提供することで、各拠点の管理ポリシーにしたがったタスク割り当てを可能にしている。

2.2.2 計算機の遊休状態を積極的に利用する技術

BOINC

SETI@home[12] や Folding@home の成功により、遊休状態の計算機を大量に利用することで解を探索する研究が数多く現れてきた。そこで、SETI@home の開発に携わった Anderson

[1] http://www.eurogrid.org

らはSETI@homeに似たシミュレーションを実現するための障壁を低くするために、Berkeley Open Infrastructure for Network Computing (BOINC)[13] を設計、実装した。

BOINC では、遊休状態の計算機を積極的に利用する目的から、Work Unit と呼ばれる概念を導入し、1 台あたりのタスク実行にかかる時間を数分程度に抑えている。悪意ある参加者からの信頼できない結果を排除する目的で、あえて冗長なタスク割り当てを行い、多数決による結果の正当性保証を実現している。また、インターフェイスの共通化によるプログラミングの容易さ実現している。これにより、大規模な解空間を持つ問題の中でも微小分割可能であれば BOINC によって容易にグリッド環境に適用可能になった。

Condor

1988 年に University of Wisconsin-Madison の Livny らが開発した Condor[14] は、研究室内にある計算機の遊休状態を有効利用するためのタスク管理ミドルウェアである。Condor のミドルウェアを導入した計算機の所有者は、他計算機の遊休状態を利用することが可能になる。見方を変えると、計算機環境全体としてプロセッサ能力を共有することが可能になる。

Condor では、異なるアーキテクチャの計算機をサポートするために仮想的な計算資源プールとして Condor Pool を提案している。Condor Pool に対してタスクを投入すると、その時点で遊休状態の計算機にタスクが割り当てられる仕組みである。また、計算機の遊休状態は、計算機の所有者による操作がない場合、かつ特定のタスクがプロセッサを占有していない場合としている。

複数計算機によるプロセッサ能力の共有化を可能にしているものの、計算機所有者の作業を第一とするポリシーを掲げているため、実行中タスクを停止する場合がある。これに対して、Condor ではチェックポインティング機能とタスクマイグレーションによる機能によるタスク実行の継続可能性とリスタート機能による他遊休計算機でのタスク実行を可能にしている。

XtremWeb

インターネット上での大規模分散処理を目的として大量タスクを大規模に実行できるミドルウェアとして XtremWeb[15] が提案されている。

XtremWeb のタスク割り当てはその時点で遊休状態の計算機にタスクが割り当てられる仕組みであり、Condor と同様の振る舞いである。Condor では 1 台のサーバで資源情報の

収集とタスク割り当てを行うが、XtremWeb では複数のサーバに分散できるため資源とタスクの増加による負荷への対応が可能になるところが違いである。また、インターネット上の大規模分散処理を実現するために、Java による実行基盤を採用していることも特徴である。

2.2.3 PC クラスタ専用ミドルウェア

Portable Batch System

複数のシステム利用者によるバッチ処理に対して、計算機資源を効率的な利用するために米国航空宇宙局 (NASA) では Portable Batch System(PBS)[16] が開発された。現在では主に PC クラスタ向けのタスク管理ミドルウェアとして利用されている。

PBS では UNIX のシェルスクリプトで記述されたものをタスク定義ファイルとして利用できることが特徴である。シェルスクリプトのコメント行に必要とする計算機の台数、タスクの優先度を定義することができる。

また、PBS では、タスク割り当てに関して複数のタスクキューを定義できる。同時利用可能なプロセッサの台数を基準にクラス分けができるので、例えば、長時間のタスク実行を必要とする利用者と短期間のタスク実行を必要とする利用者のクラス分け、タスク間通信が密の処理を実行する利用者とバッチ処理を専用とする利用者のクラス分けが可能である。

SCore

1992 年から 2002 年 3 月まで通商産業省 (現:経済産業省) の 1 プロジェクトであった新情報処理開発機構が開発した SCore はコモディティな計算機を用いて並列計算専用機を構築するためのミドルウェアである [17]。

特別なハードウェアを必要とせずすべて市販の計算機、ネットタスク機器を用いて PC クラスタを構築する手法としては Beowulf System[18, 19] もあるが、SCore は主に並列処理のためのソフトウェア技術の向上を目標として開発された。既存 OS カーネルに対して、OS のカーネルを介さないで通信するユーザレベル通信やプロセッサによるデータコピーを介さないで通信するゼロコピー通信をもつ PM 通信ライブラリ [20] の導入や、同時刻に全計算機がプロセスを切替え、かつ同一アプリケーションが実行されるスケジューリング手法であるギャングスケジューリング [21] が導入されている。プロセッサ間でデータを交換しながら処理する形態のプログラム実行に強力である。

なお、SCore 単体ではタスク管理ミドルウェアは存在しないため、SCore 向けに改良を施した PBS をインストールすることで大量タスク実行が可能になる。

2.3. アプリケーション実行支援

2.3.1 グリッドポータル

GPDK

Globus を用いてグリッド環境を構築した場合、Globus では原則としてコマンドラインでの操作を要求する。シミュレーション実施者によるコマンドラインでのタスク実行は操作面で負担が大きい。そこで、Web ブラウザを中心としたユーザ操作環境を構築することで、シミュレーション実施者の操作面での負担を緩和させる技術としてグリッドポータルがある。

多くのグリッドポータルでは、シミュレーションに特化した独自の操作体系を実現すべく、Perl/CGI、Java Server Pages(JSP)、PHP といった Web システム開発用言語を用いてアプリケーション依存なグリッドポータルを構築している。これにより、シミュレーション実施者の操作面での負担を緩和させているが、グリッドポータルを構築する側が負担を強いられることは否めない。また、グリッドポータル構築のためにシミュレーション実施者は Web システム開発用言語に精通しなければならない。この負担を軽減するために Novotny は The Grid Portal Development Kit(GPDK)[22] を提案している。

GPDK は Java ベースのグリッドポータル構築キットで、Globus をプログラミング言語 Java を使ってアクセスする Java Commodity Grid Kit(Java CoG)[23] を利用してタスク実行命令、計算結果回収命令を容易にしている。また、JSP との親和性が高いことが特徴である。

GridSpeed

GPDK ではタスク実行やデータ転送に必要なライブラリを提供しているだけに過ぎず、Perl、JSP といった Web ポータル開発言語の知識が必要である。これに対し、鈴村らは GridSpeed[24] を提案し、グリッドポータル構築負担の低減化を図っている。

GridSpeed では、単一タスクやパラメータスウィープ型タスクといった複数のタスク実行に対応したポータルの生成を可能にするとともに、Globus や Condor といった様々なミ

ドルウェア、scp、GridFTP といった複数のプロトコルが使われているグリッド環境でもポータルが作成できる特徴を有する。

PCT4G

ユーザフレンドリーなグリッドポータルが数多く構築されているが、すべての計算ノードへのアプリケーションのインストール作業は問題解決環境を管理する者の負担となり、依然として高い。そこで、これらの作業を自動化することで管理者の負担を緩和させるものとして、白砂らは PCT4G[25] を開発した。

PCT4G は Web フロントエンド、ポータルバックエンド、データマネージャの 3 層構造のシステムであり、アプリケーションのインストール機能、アプリケーションを最新状態に保つ機能、シミュレーション実施者が開発したアプリケーションをグリッドで即座に利用できるためのポータル作成機能を提供している。

2.3.2 タスクフローシステム

Gridbus Workflow

複雑なシミュレーションの実施を必要とする昨今において、複数のシミュレーションプログラムを組み合わせてシミュレーションを実施することが一般的になっている。適切な順序でシミュレーションプログラムを実行するためのフレームワークとしてタスクフローシステムがある。Yu らは Gridbus Toolkit[26] 向けのタスクフローシステムとして Gridbus Workflow として [27] を提案している。

Gridbus Workflow ではプログラム実行とデータ転送の作業を明確に分けていること、また、XML ベースのタスクフロー記述言語を独自に定義していることが特徴である。

Grid Application Definition Language

Fraunhofer Resource Grid(FhRG) で用いられているタスクフローシステムでは、Grid Application Definition Language (GADL)[28] を提案している。

GADL では、並行動作、分散状態、資源の概念を表現可能にするために、高度な数学的検証能力をもつ非同期並行システムの記述言語である Petri Net[29] を採用している。Petri Net により事象と条件による二部有向グラフをもつタスクフローを表現できることが特徴である。また、タスクフローは XML に基づいて記述される。

DAGMan

Condor において依存関係をもつ複数のタスクを実行するために DAGMan[30] と呼ばれるタスクフローシステムがある。

DAGMan では非循環有向グラフ (Directed Acyclic Graph) に基づいた依存関係をもつタスク群の実行を適切に行うために、タスク間の親子関係を明確に定義するところが特徴である。このとき、DAGMan では独自の書式でタスク間の親子関係を記述する。また、Condor は異なるハードウェアアーキテクチャの計算機でひとつの仮想的な計算機資源を提供しているため、DAGMan ではハードウェア依存のアプリケーションに対しても適切に割り当てることが可能である。

2.4. アプリケーション開発向けミドルウェア

Ninf

グリッドのように広域分散環境下での分散並列処理環境では、計算速度に対する通信速度の比率が低いことから、非常に大量な乱数値に対して特定の演算を行うもの、特定のパラメータを走査しながら計算するものといった、粗粒度での並列実行に適している。一方で、並列実行プログラムを記述するためには、メッセージパッシングに基づいたライブラリを利用する、独自の通信ライブラリを実装したものを利用するのいずれかを選択せねばならず、シミュレーション実施者はプログラムを実装する際に負担を強いられる。このような状況から、中田らは広域分散並列計算に適した分散計算の枠組みとして Ninf[31] を設計、実装した。

Ninf では計算を粗粒度な要素ルーチンに分割してネットタスク上に分散した計算機に配置し、それらの間にデータフローを構成することで計算を行う。要素ルーチンはライブラリのかたちで提供される。膨大なソフトウェア資産をもつシミュレーション実施者は特に通信部分の実装を考慮する必要がないため、粗粒度な並列実行プログラムを記述する際の負担が軽減できる。データフローの依存関係を考慮したタスクの割り当て、トランザクション呼び出しの失敗による他ノードへのタスク割り当てを Ninf 側のメタサーバが実現可能にしていること、シミュレーション実施者とのインタラクションを可能にするためにコールバック機能が付加されていることが特徴である。

Globus との親和性を図ることで広域分散計算に対応した Ninf-G[32] や、Globus との親和性を維持しつつも実行プログラム起動時間の短縮と通信効率の向上を可能にした Ninf-

G2[33]、Condor のチェックポインティング機能を利用することで耐故障性を可能にした Ninf-C[34] として拡張されている。

NetSolve

Ninf 以外のグリッド向け RPC システムとしては、Casanova と Dongarra らによって開発された NetSolve[35] がある。Ninf と同様、プログラマに対して要素ルーチンに分割されたライブラリを提供している。

NetSolve は、計算を依頼する側である Client、計算を実施する側である Server、利用可能な Server 情報の管理とタスク実行時の Server 選択を行う Agent で構成される。Agent では、Server の利用状況に応じて要素ルーチンの実行先を適切に切替える機能を有する。これにより、タスク実行の負荷分散を軽減する。

NetSolve では、C 言語、FORTRAN、Mathematica、Matlab、Octave 等のプログラミング言語から利用できるライブラリを有していること、引数の内容は異なるが同じリクエストを複数実行して結果を得る Request Farming 機能を有していることが特徴である。

OmniRPC

Ninf や NetSolve では NAT 技術を用いて構成された PC クラスタやファイアウォール内の PC クラスタに対して外部からアクセスすることができない。また、認証技術の未対応が問題である。そこで、佐藤らはクラスタからグリッドまでシームレスな並列プログラミングを可能にする OmniRPC[36] を設計、実装した。

OmniRPC の特徴は、パラメータ検索などの並列アプリケーションにおいて大量のデータ転送や再計算が必要な場合に再利用するために自動初期化実行モジュール機能を提供していること、複数拠点にある PC クラスタのノードをすべてリモートホストとして取り扱える機能を提供していること、プライベートアドレスを用いて構成された PC クラスタであっても広域分散計算ミドルウェアとして利用できること、プロキシ機能によってファイアウォール超えが可能になったことが挙げられる。

2.5. 統合環境構築ミドルウェア

NAREGI Grid Middleware

国立情報学研究所グリッド研究開発推進拠点NAREGIでは、日本におけるグリッド技術を用いた大規模計算環境構築の研究として、実運用に耐える品質でのグリッドミドルウェアの開発 [2] と、最先端のナノサイエンス研究を推進するためのグリッド向けアプリケーションの開発を行っている。

その成果物であるグリッド基盤ソフトウェアNAREGI Grid Middleware[37, 38] はナノサイエンスや分子科学分野のシミュレーション実施者に対してシステムに依存しない使いやすい計算機環境の提供、マルチフィジクス、マルチスケールに対応した異機種計算資源の利用と複数シミュレーションとの連携動作を実現することを目標としている。NAREGI Grid Middlewareは、既存のミドルウェアであるGlobusやUNICOREとの連携を可能にするモジュールの提供、GridRPCやGridMPIといったグリッド向けプログラミング環境の提供、タスクフローシステムの提供、各拠点の認証ポリシーを吸収した統一認証基盤の提供、を実現している。

Triana

Taylorらは既存のグリッドミドルウェアを使わないでグリッド環境を構築するための枠組みとしてTriana[39] を開発した。

Trianaは、Peer-to-Peer基盤システムであるJXTAとWebサービスをベースに、Grid Application Prototype Interface(GAP) と呼ばれるインターフェイスを提供し、ひとつのグリッド基盤を構築している。タスク割り当てにおける計算機との通信はJXTAのフレークタスクを利用しており、GlobusやCondorといったミドルウェアを使っていない。

Trianaでは独自のタスクフローシステムを有している。Trianaでは高速フーリエ変換や数学関数のタスクがあらかじめ準備されているため、シミュレーションを実施するためのプログラム作成が不要である。また、Triana GUIと呼ばれるタスクフロー記述用ユーザインターフェイスで記述されたタスクフローデータはThe Workflow Management Coalition(WfMC)で定義されたBPEL4WS (Business Process Execution Language for Web Services)[40] に準拠したフォーマットで出力される。

CyberGRIP

シミュレーション技術の発展により、製品開発期間の短縮や実機実験回数の低減といった製品開発にかかるコストの削減、マーケティング戦略立案のためのデータ分析計算と未来予測を可能にする。しかしながら、これらのシミュレーションにはそれ相応の計算機能力を準備しなければならない問題点がある。この問題点に対し、富士通研究所では企業内の事務用計算機のプロセッサ使用状況に注目し、事務用計算機を有効活用する枠組みとして CyberGRIP[41] を開発した。

CyberGRIP では Organic Job Controller(OJC) と呼ばれるタスク実行制御モジュールを中心にシステムが構成される。OJC では依存関係をもつタスクの実行が可能である。また、大量タスク実行を重視しており、パラメータ設定ミスの軽減を図るために、タスクの再実行を行うためのインターフェイスが準備されていること、動的タスク制御機能と呼ばれるタスクの実行結果に基づくタスク投入制御が実現されている。これにより、大量タスク実行におけるパラメータ設定ミスの低減と無駄なタスク実行を削減できるため、シミュレーション実行時間の削減が可能になった [42] と報告されている。

2.6. グリッド技術による問題解決環境構築事例

近年では、様々な分野でグリッド技術を用いた問題解決環境の構築が行われている。本節では、グリッド基盤技術、グリッド用インターフェイス、統合環境を用いた問題解決環境の構築について特徴のある事例を示す。

タンパク質立体構造予測

Uk らは経験的エネルギ関数による生体高分子システムモデリングパッケージ CHARMM (Chemistry at HARvard Macromolecular Mechanics) を United Device 社の MetaProcessor を用いたデスクトップグリッド環境に展開した [43]。

また、Krishnan はバイオインフォマティクスで最も広く使われる BLAST と呼ばれるアプリケーションの並列実行アプリケーションとして GridBLAST を実装した [44]。BLAST は DNA 塩基配列もしくはアミノ酸配列のシーケンス断片をもつデータベースから、類似したシーケンス断片を検索するものであり、検索作業の並列化によって類似度の高いシーケンス断片の発見にかかる時間が削減できる。これに対し、GridBLAST では、Globus Toolkit を用いて類似度検索部分を並列実行可能にし、高スループット計算を実現している。

高精度分子シミュレーションのグリッド技術の適用

ナノマテリアル、生体高分子といったナノスケールの領域において、分子レベルでの大規模シミュレーションは必要であるが、Globus などのミドルウェアを用いてグリッド環境し、分子レベルの大規模シミュレーションを実現するのは一般の自然科学者にとってハードルの高い内容である。

池上らは複数のサイトに散在する計算機を用いた分子レベルでの大規模シミュレーションを可能にすることを目的として、GON ライブラリを開発した [45]。提案する GON ライブラリは、GridRPC ベースの実装で、バッチキューシステムを模した API、シミュレーションで使うファイルの自動転送機能、シミュレーション途中でのパラメータ変更を可能にする機能を提供していることが特徴である。

ボランティアコンピューティング

インターネットに接続されている計算機の遊休状態を利用するグリッド技術のうち、遊休状態の時間を自主的に提供する枠組みをボランティアコンピューティングと呼ぶ。

プエルトリコのアレシボ電波望遠鏡のからのデータをインターネットに接続しているパーソナルコンピュータで手分けして解析し、地球外知的生命体探査を行う SETI@home プロジェクト[2] [12] はボランティアコンピューティングの代表例である。他にも RSA 社が主催する暗号解読コンテストへの参加プロジェクト distributed.net[3] 、メルセンネ素数の発見[4] や、タンパク質立体構造予測[5] が有名である。最近では気象予測、創薬シミュレーション、ライフサイエンス、金融工学などにも適用している。

distributed.net の 1997 年 10 月 22 日のプレス[6] では、のべ 50 万台の様々な処理能力を有する計算機の遊休状態を 250 日間利用して、RC5-32 56bit の暗号鍵解読を解いた、と報告されている。これは当時主流であった Intel 社製 Pentium 200MHz のプロセッサ能力を例に台数換算すると約 26000 台分の計算機[7] が得られたことになる。言い替えると、約 26000 プロセッサが搭載されている並列計算機がインターネットに接続された計算機によって作り出されたことになる。

[2] http://setiathome.ssl.berkeley.edu

[3] http://www.distributed.net

[4] http://mersenne.org

[5] http://predictor.scripps.edu

[6] http://www.distributed.net/pressroom/56-announce.html

[7] 同様に IBM 社、Motorola 社、Apple 社が共同開発した PowerPC 604e/200MHz のプロセッサ能力を例に台数換算すると約 11000 台分の計算機が得られたことになる

これらの事例の成功から、近年ではボランティアコンピューティングの商用化についても検討されている。例えば、cell computing[46] が 2002 年 12 月から 2003 年 4 月の約 4ヶ月にわたって行われた大規模実証実験では、12000 台を超える計算機の遊休状態を利用したこと、それらによって得られたピーク時性能は 3Tflops を超えたと報告されている[8] 。

distributed.net の事例や cell computing の大規模実証実験の成果は、従来、有限時間では解くことが困難とされていた問題が大量の計算機を利用することで有限時間内に解けるようになったことを証明した。すなわち、高スループット計算によるシミュレーションの可能性を示唆した。

皮膚診断サービスへの応用

化粧品の販売においては、顧客が求める商品を販売することが第一と考えられている。具体的には、顧客の肌にあった化粧品を販売することが求められている。このとき、販売員は顧客の肌状態を判断しなければならない。そこで、平石らはグリッド技術を適用した肌状態診断システム Skin-Expert[6] を開発し、販売員による顧客の肌診断をサポートしている。

Skin-Expert では、カメラ付き携帯電話をユーザインターフェイスとしたシステム構成になっている。携帯電話で顧客の肌画像を撮影するとともに肌画像をグリッドシステムに転送する。転送された画像はグリッド技術で構築された肌状態診断システムで解析され、結果を携帯電話に転送する。Skin-Expert のグリッド適用によって、10 万件の診断要求が同時に到着しても 1 時間半で処理することが可能になった。

金融分野におけるグリッドの支援

今日、資産運用のために株式など投機性の高い金融商品を保有する場合ことも少なくない。投機性の高い金融商品ほどそれによって生じるリスクも高いため、リスク回避を考慮した資産運用が求められる。状況が時々刻々と変化する金融市場において、様々な情報を駆使しながらリスク回避策を講じるためには、大規模なシミュレーションが必要になる。

これに対し、IBM 東京基礎研究所では、金融グリッド・マネージャー [47] と呼ばれるミドルウェアを開発し、金融工学におけるグリッド技術を展開している。金融グリッド・マネージャーは、暗号化技術の導入による機密情報の隠匿、異機種環境における計算結果の保証、デッドラインスケジューリングの適用による指定時刻までの結果出力を可能にして

[8] http://www.cellcomputing.jp/example/test6.html

いる。

政策立案支援におけるグリッド応用

経済学、政治学などの社会科学の研究分野では様々な実証研究が行われているが、人間の行動に基づいた研究分野であることから、実験的な研究ができない。また、国や地方公共団体では様々な政策を打ち出しているが、これらが及ぼす影響について実験的な見地から議論されていないため、効果がわからないまま政策が実施されている現状がある。

これに対し、鵜飼らは政策立案支援向け大規模社会シミュレーションを実現するために、グリッドの枠組みを積極的に利用した環境整備に関する研究を開始した [48]。鵜飼らの研究では、政策上重要となるシミュレーション手法を社会科学の立場から研究するとともに、既存のグリッドミドルウェアを使って社会シミュレーションを実現するプログラムの開発を行っている。この研究をとおして政策変数の変化による社会の振舞いをコンピュータによって推測できることを目指している。

2.7. 関連研究の分類と本研究の位置付け

本節では、先に述べた関連研究についてグリッドのプロトコルアーキテクチャに分類するとともに、本研究の位置づけを示す。

2.7.1 グリッドプロトコルアーキテクチャ

本論文では、Foster らが文献 [49] で提唱したグリッドのプロトコルアーキテクチャ(図 2.1) にしたがって関連研究を分類する。

- Fabric 層
  プロセッサ、ストレージ、ネットタスクといった物理的資源とそれをコントロールするオペレーティングシステムで構成された部分を規定する。

- Connectivity 層
  計算資源にアクセスする際の認証方法を規定する。

- Resource 層
  複数の計算資源を仮想化する仕組みを規定する。

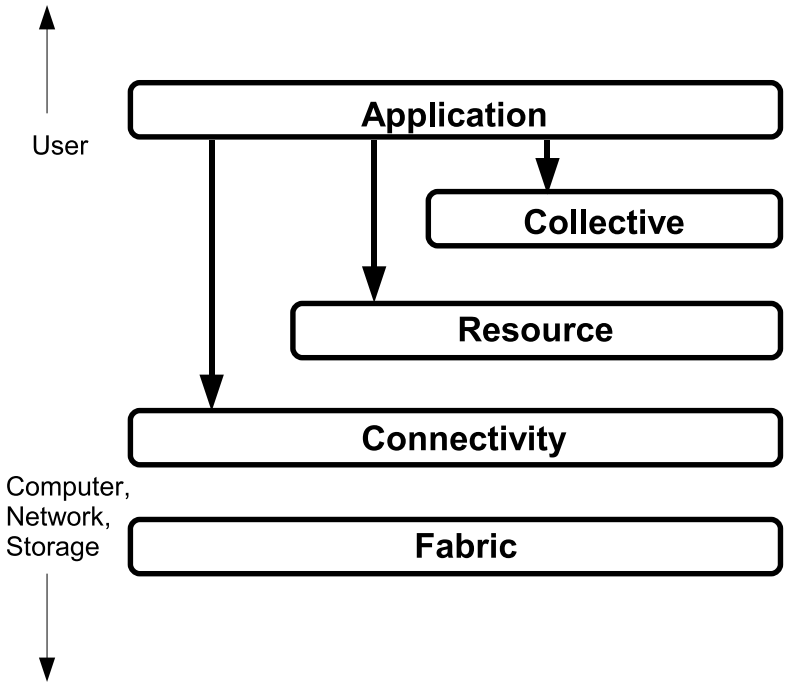


図 2.1　グリッドプロトコルアーキテクチャ

- Collective 層
  計算資源に対してタスク割り当てやデータ転送の仕組みを規定する。また、タスクや計算資源を管理する仕組みを規定する。

- Application 層
  仮想的な計算環境で実行するプログラム、シミュレーション結果を可視化するプログラムなどの記述方法を規定する。

2.7.2　関連研究の分類

本章で紹介した Globus などの複数拠点を接続するミドルウェア、Condor といった計算機の遊休状態を積極的に利用する技術、PBS といった PC クラスタ専用ミドルウェア、GPDK などのグリッドポータル、Gridbus Workflow といったタスクフローシステム、Ninf などのアプリケーション開発向けミドルウェア、NAREGI Grid Middleware といった統合環境構築ミドルウェアとグリッドプロトコルアーキテクチャの関係、および本研究の位置づけを図 2.2 に示す。

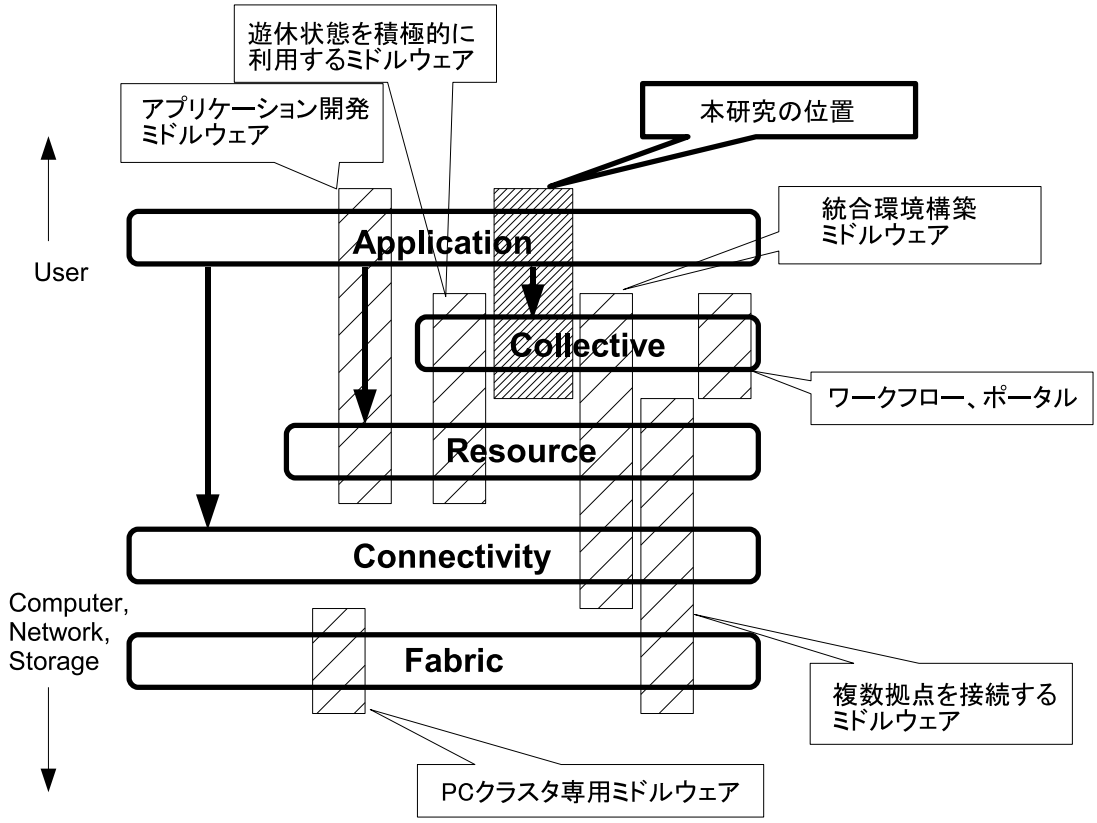


図 2.2 関連研究とグリッドプロトコルアーキテクチャの関係

複数拠点を接続するミドルウェアは、計算資源の仮想化と異なる拠点の計算機を利用するための認証技術が中心なので Resource 層、Connectivity 層、Fabric 層にまたがっている。

計算機の遊休状態を積極的に利用する技術は、遊休状態の計算機を仮想的な 1 つの計算機としてみせる技術と遊休状態の計算機に対して適切にタスクを割り当てる技術が中心なので Collective 層と Resource 層にまたがっている。

PC クラスタ専用ミドルウェアは、ひとつの並列計算機を構築する目的から Fabric 層に位置する。

グリッドポータルとタスクフローシステムは、アプリケーションを適切に操作する目的が含まれていること、適切な計算機へのタスク割り当てと結果の可視化を支援する目的から、Collective 層に位置する。

アプリケーション開発向けミドルウェアは、仮想化された計算資源上で動作するプログラムを実装するためのライブラリという側面が強いため、Application 層と Resource 層との関係をもつ部分に位置する。

統合環境構築ミドルウェアは、他のグリッドミドルウェアとの連携を図るための意識が強いため、Collective 層、Resource 層、Collective 層にまたがっている。

図 2.2 による分類から、グリッドに関する研究の中核をなす部分が Collective 層と Resource 層であることがわかる。多くのミドルウェアでは、複数の計算機をソフトウェアによってひとつの仮想的な計算機として提供することに注力しているため、Collective 層と Resource 層での研究開発が盛んであることが伺える。

2.7.3 本研究の位置付け

図 2.2 に示したように、グリッドによる問題解決環境の構築においては Appication 層を中心として Application 層と Connectivity 層に注目した場合、Application 層と Resource 層に注目した場合、Application 層と Collective 層に注目した場合の 3 方式がある。以下では、3 方式についてアプリケーションの実装方針の違いによる問題解決環境の構築指針について議論する。

- **Application 層と Connectivity 層に注目した場合**
  Application 層と Connectivity 層に注目した場合について、基盤面ではこれに相当する技術はない。応用面でボランティアコンピューティングに適用されている。ボランティアコンピューティングでは、それ専用の計算ができればよいため Collective 層や Resource 層の技術を持ち込むことを考えなくてよい。

  一方で、適切なタスク割り当てやデータ転送を行うためにはシミュレーションプログラムに Collective 層や Resource 層の技術を盛り込まなければならない。Application 層と Connectivity 層に注目した場合では、計算資源を熟知したプログラム開発が必要になるため、シミュレーション実施者に問題解決環境を構築した経験がなければ難しい。

- **Application 層と Resource 層に注目した場合**
  Application 層と Resource 層に注目した場合について、Ninf などアプリケーション開発向けミドルウェアの研究が相当する。また、応用面では、高精度分子シミュレーション用ライブラリである GON ライブラリが相当する。大規模並列計算環境を意識したプログラミングモデルの提案が主である。

  仮想的な大規模並列分散計算環境を意識したシミュレーションプログラムの開発が可能になるものの、シミュレーションプログラムに対してタスク割り当てやデータ転送

といったCollective層の機能を独自に組み入れなければならない課題は残されている。

- **Application層とCollective層に注目した場合**
  Application層とCollective層に注目した場合について、既存のタスク割り当て技術を利用する方針でシミュレーションプログラムを開発するため、プログラム内にタスク割り当て機能を組み込む必要がない。

  高スループット計算のように複数の計算機に複製配置したプログラムに対して大量のパラメータを分散投入し、演算結果を得るといった戦略の場合、シミュレーションプログラムにタスク割り当て機能を組み込む必要がないため、プログラム実装時のコストを大幅に削減できる。但し、武宮らが文献[5]でGlobusでのタスク起動時間の遅延について指摘しているように、ミドルウェアによるタスク起動時間のコストが発生するため、シミュレーション時間の短縮を優先する場合には、ミドルウェアによるタスク起動のコストを十分に検討しなければならない。

これらの比較結果から、Fabric層に近い層を意識するか否かでシミュレーション用プログラムの実装方法が異なることがいえる。このような状況下で、シミュレーション実施者がシミュレーション結果の改善をするための方法を考える。

Application層とConnectivity層に注目した場合とApplication層とResource層に注目した場合では、シミュレーション実施者はグリッド環境に適した大量タスク実行のためにシミュレーション用プログラムの改良を必要とする。具体的には、計算機環境に適するように依存関係に基づく計算を実施するプログラムの実装と大量タスク実行をするプログラムの実装をしなければならない。したがって、シミュレーション結果の改善を考えた場合、シミュレーション用プログラムに改善するためのコードを埋め込む必要がある。

一方、Application層とCollective層に注目した場合では、依存関係に基づくアプリケーションの実行を可能にする機能や大量タスク実行を可能にする機能が既に提供されているため、グリッドミドルウェアを利用するための知識が必要になるだけでよい。シミュレーション結果の改善はプログラム実行操作とパラメータ変更を適切に行う仕組みを提供すればよい。

本論文では、単一の計算機で実行できるプログラムを改良することなく、複数拠点の計算機を積極的に利用することを着目した。Application層とCollective層の2層に注目し、シミュレーション実施者の意思決定を円滑に行うためのフレームワークについて、システム構築の観点から研究を進めたことが、関連研究との違いである。

2.8. まとめ

グリッドによる問題解決環境の構築を行うためのミドルウェアは、計算機により近いものからシミュレーション実施者に近いものまで様々なかたちで提供されている。

すべてのミドルウェアに共通することとして、ミドルウェアはアプリケーションレベルで実現されていることである。インターネットのような広域通信を可能とする技術の上に成り立っており、特別なハードウェアを要求しない。また、ミドルウェアはより簡単に利用すること、もしくはより速く解を得ることに注力されている。シミュレーションを行いながらパラメータを変更するといった、試行錯誤動作によるシミュレーションの実施を既存のミドルウェアで実現するのは困難である。

以降の章では、Collective 層と Application 層の 2 層に注目し、シミュレーション実施者の意思決定を円滑に行うためのフレームワークについて、タスク管理を中心としたシステム構築の観点から論じる。

第3章

シミュレーション実施者の意思決定と高スループット計算向けグリッド

3.1. 高スループット計算とは

製品開発や学術研究において、グリッド技術を利用した問題解決環境の構築が進められている。その過程において、グリッドの長所が明確になってきた。シミュレーションを主とするコンピューティング・グリッドにおいては、パラメータスウィープ、モンテカルロ法によるシミュレーションに対して積極的に使われている。これらのシミュレーション手法の特徴は、

1. 計算対象がもつ解の空間 (解空間) が容易に分割できること、
2. 分割した解空間が並列に実行できること、
3. 並列実行中のプログラム間でデータの交換が殆んど行われないこと、

である。このような特徴をもつ計算手法を総称して「高スループット計算」と呼ぶ。より具体的には、以下3種類のアプリケーションが高スループット計算とされている。

- データ並列アプリケーション
  ある問題の解空間が分割可能で、分割された空間が他の空間に影響を及ぼすことがないタイプのアプリケーション。

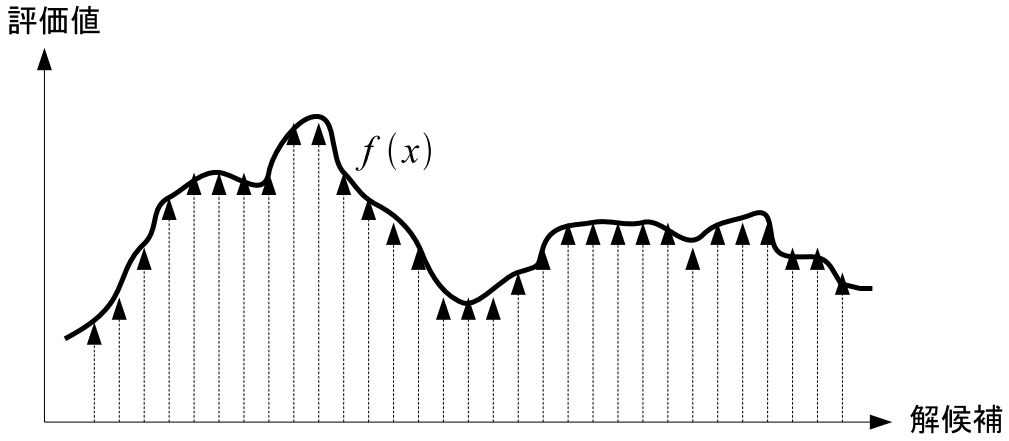


図 3.1 高スループット計算における解候補と解探索の関係

- パラメータスウィープアプリケーション
  ある問題の解を見つける際、入力となるパラメータを考えられるあらゆる範囲で準備し、ひとつずつ実行するタイプのアプリケーション。

- 確率論的アプリケーション
  ある問題の解が確率分布と関係があり、入力となるパラメータにある確率分布もしくは乱数を使い、統計的に十分な数の計算を繰り返すことで、解が得られるタイプのアプリケーション。

一般的に高スループット計算は、解候補が複数存在する問題に適用される。解 x は評価関数 $f(x)$ のパラメータであり、ひとつの評価値をもつ (図 3.1)。解候補すべてを評価関数にあてはめ、評価値が最大となる解をみつける極めて単純なアルゴリズムで最適解を探索する。

高スループット計算は演繹的な解法ではなく、帰納的な解法であることが特徴である。例えば、最適値を探す問題を考える。高スループット計算の場合の演算は、条件式に解候補をひとつずつ当てはめる方法が採用される。解候補の数を大量に準備することで、演繹的な手法に比べてより精度の高い最適値を導出することが可能になる。

高スループット計算はしばし驚異的並列 (embarassingly parallel) 計算と呼ばれるように、一連の計算処理を完全に独立な部分に分割して個々のプロセッサで並列実行できる特徴を有する。同一処理能力を有するプロセッサが複数存在した計算機環境においては、プロセッサの数だけ処理時間の大幅な短縮が見込める。また、異なる処理能力を有するプロセッサ

が複数存在した計算機環境では、処理能力に応じてタスクの割り当て数を変更することで処理時間の大幅な短縮が見込める。

3.2. 意思決定に基づくシミュレーション実施の必要性

3.2.1 シミュレーション実施者による意思とは

シミュレーション実施者は、「シミュレーションによって問題を解決する」ことが目標である。例えば、シミュレーションにふさわしいプログラムを選択すること、プログラムを計算機で実行すること、計算した結果を解析して問題が解決できるか確認すること、といった行動を起こす。そして、シミュレーション実施中に様々なこと考えながら、シミュレーション結果を得たり、結果の改善を図る。

このように、シミュレーション実施者は問題解決の糸口をみつけるためには様々な行動に基づいてシミュレーションを行っている。本論文では、シミュレーション実施者の意思と意思決定について次のように定義する。

- **意思**
  シミュレーション実施者によって問題解決の糸口を見つけるために必要とされる複数の行動

- **意思決定**
  問題解決の糸口を見つけるための複数の行動からひとつの行動を決定する行為

シミュレーションの実施にあたり想定される場面と具体的な意思決定について示す。

- シミュレーション実施前
  解くべき問題に合うシミュレーションプログラムの選択、プログラムに与えるパラメータの検討が行われる。

- シミュレーション実施中
  途中結果を確認して、シミュレーションを継続すべきかどうか、パラメータを変更して再実行を試みるか判断する。

- シミュレーション終了時
  次に行うシミュレーションの準備をする。

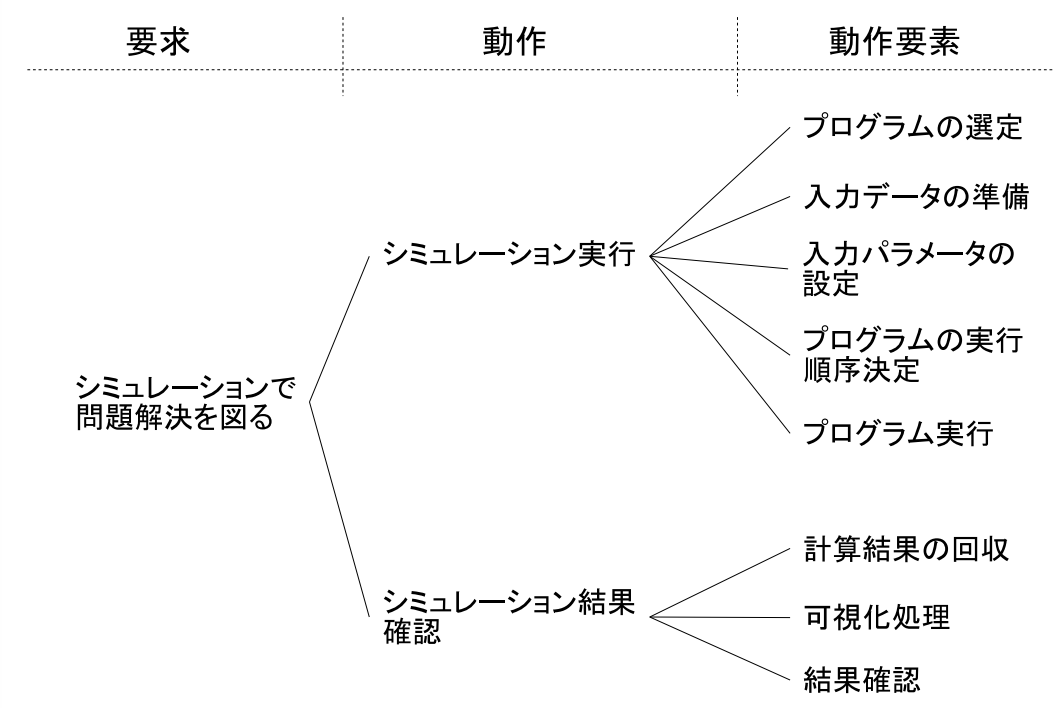


図 3.2　シミュレーション実施者による動作の内訳

また、シミュレーション実施中、想定した結果が得られない場合は以下に示す意思決定が行われる。

- 大量タスク実行を行う場合
  大量タスク実行では、シミュレーションで用いるプログラムは１種類であるが、プログラムに異なるパラメータを与えて実行する。シミュレーション実施者はどのような種類のパラメータの組み合わせをどれだけ用意するべきか決定するとともに、特定のパラメータの組み合わせに絞ったタスク実行を行うかの決定をする。

- タスク実行に依存関係がある場合
  依存関係のあるタスク実行では、シミュレーション実施者は最初からシミュレーションしなおすか、途中結果を使って再計算しなおすかの選択をする。

3.2.2 シミュレーション実施者による動作の内訳

図 3.2 はシミュレーション実施者が問題解決を行う際にとる動作を分類したものである。シミュレーション実施者は、「問題を解決する」という目的を達成するために、実施者が考えた数理モデルをコンピュータで演算し、有益な情報を見出そうと試みる。このとき、シミュレーション実行の動作はシミュレーション実施者から計算機環境への伝達が、シミュレーション結果確認の動作は計算機環境からシミュレーション実施者への伝達が発生している。シミュレーション実施者の継続的な作業を考えるとシミュレーション実施者と計算機環境との間に対話的な動作が発生する。

シミュレーション実行の動作およびシミュレーション結果確認の動作それぞれにおける具体的な意思決定動作を以下に示す。

- シミュレーション実行
  シミュレーションを開始する時点で発生する動作である。コンピュータを駆使してシミュレーションを行うということは、プログラムを実行することである。より具体的には、プログラムの選定、入力データの準備、プログラムに与えるパラメータを設定すること、プログラムの実行順序の決定、プログラム実行操作、が動作要素になる。

- シミュレーション結果確認
  シミュレーションが完了した後に発生する動作である。シミュレーションを実施して得られるものは、一般的に結果は数値もしくは文字列で構成されたデータである。出力されたシミュレーション結果はシミュレーション実施者が判断できる状態ではない場合が多いため、シミュレーション結果の分析を行いやすくする操作が必要になる。そのため、計算結果を回収すること、可視化処理を行うこと、結果確認を行うこと、が動作要素になる。

3.2.3 シミュレーション実施者によるシミュレーション実行制御の必要性

現在では、様々な分野でグリッド技術が注目され、グリッド技術を用いた問題解決環境が構築されている。計算機の性能向上やグリッド基盤技術の改良によって、シミュレーション実施時間がより短縮される状況になってきたが、シミュレーション実施者が希望する作業とグリッドミドルウェアの操作体系に差があるため、依然としてシミュレーション実施者のグリッド環境利用における負担は大きい。ここでは、シミュレーション実施者が希望するシミュレーション実行制御について以下に示す。

パラメータを変化させたシミュレーションの実施

シミュレーション実施者にとって、問題解決環境を利用することは、シミュレーションによって得られた結果から問題解決の糸口を発見することである。問題解決の糸口を発見するためには、シミュレーション実行を何度も繰り返すことが大切である。

例えば、量子化学計算分野のシミュレーション実施者は途中結果を検討してシミュレーションの妥当性を判断する作業が一般的に行われる。シミュレーション結果に妥当性が得られなかったと判断したら、シミュレーション実施者は実行中のシミュレーションを途中で放棄する意思決定を下すとともに、パラメータを変更して再実行する操作を行う。また、シミュレーションの妥当性を判断するもう一つの方法として、パラメータを変えて複数のシミュレーションを同時並行で実施する。

これに対して、池上らが開発した高精度分子シミュレーション用ライブラリである GON ライブラリ [45] では、途中結果を得られる機構を導入している。これにより、シミュレーション実施者の意思決定によって実行中のシミュレーションを放棄する、もしくはパラメータを変更して再実行するといった動作が可能になる。

このように、シミュレーション実施者はあるひとつのシミュレーション実施に対して希望する結果が得られるまで改善する動作を要求している。そのためには、シミュレーション実施者による**パラメータの値を修正する行動に対して、問題解決システム側で実行中タスクの放棄と、パラメータの値を変更したタスクを再実行できること**が求められる。

シミュレーション結果の管理と再利用

量子化学計算の分野で最も使われるシミュレーションプログラム Gaussian では、Gaussian 固有のキーワードや分子構造などの情報を多様に変化させる計算を行う。このとき、異なる利用者が同一の計算を実行するケースが多い。Gaussian によるシミュレーション結果の取得には長時間を要するため、既に計算された結果がある場合は再利用される。同一パラメータのシミュレーションに対して重複実行するのはシミュレーション時間と計算機資源の無駄づかいである。

この状況を改善するには、**問題解決システム側で再利用を考慮したシミュレーション結果の管理**が求められる。具体的にはシミュレーション実施者はパラメータの値をキーとしてシミュレーション結果を検索するので、パラメータの値とシミュレーション結果を対にした管理が必要になる。

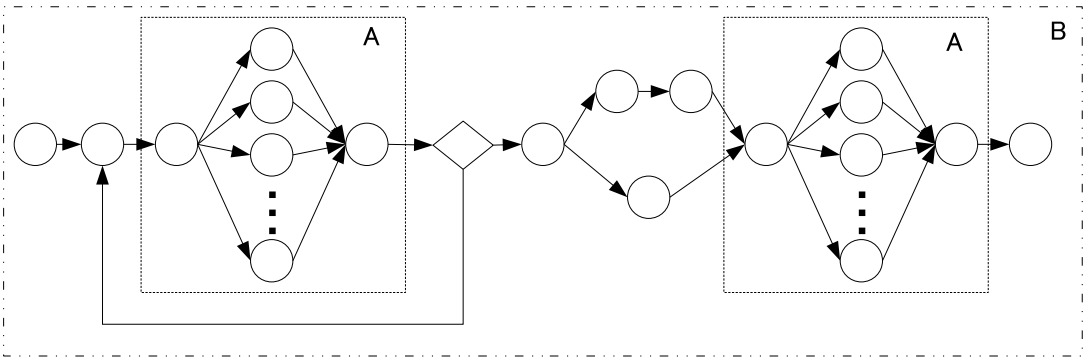


図 3.3 タスクフローの例

試行錯誤によるシミュレーションの実施

パラメータを変化させたシミュレーションの実施とシミュレーション結果の管理と再利用については、主に単一タスクにおけるシミュレーション実施者の要求である。

一方で、解くべき問題が複雑になるほど問題解決のための行動も複雑になる。様々なプログラムを有機的に利用して問題解決の糸口を発見することになる。上島らは多種多様なシミュレーションの実施と出力された計算結果の膨大さから問題解決の糸口となる結論を導出するための再現性が困難になっていることを示唆している [50]。この課題を解決するためには、シミュレーション実施時に与える各種パラメータの履歴やプログラム実行のトレースができることが望ましいと提言している。

例えば、図 3.3 のようなタスクフローを考える。図 3.3 は、領域 B で囲まれたタスクフロー全体に対して、領域 A で囲まれた大量タスク実行と単一タスク実行の間に依存関係がある場合の単純なモデルである。図 3.3 のようなシミュレーションが行われる場合において、問題解決の糸口を発見するために、シミュレーション実施者は大量タスクの実行においては想定したパラメータの組を確認する行為、依存関係のタスク実行においてはタスクの実行状況を確認する行為をとる。また、各タスクにおいて想像どおりの結果が得られているか確認し、想定外の結果が得られた場合には、シミュレーションを放棄する行為をとったのち、現在よりも前のタスクのパラメータを変更し、シミュレーションを継続する行為をとる。

このように、問題解決システムでは、複雑なタスク実行を行うようなシミュレーションにおいてもシミュレーション実施者によって試行錯誤を可能にしたシミュレーションを実現することが求められている。具体的には、タスクフローを構成するようなシミュレーショ

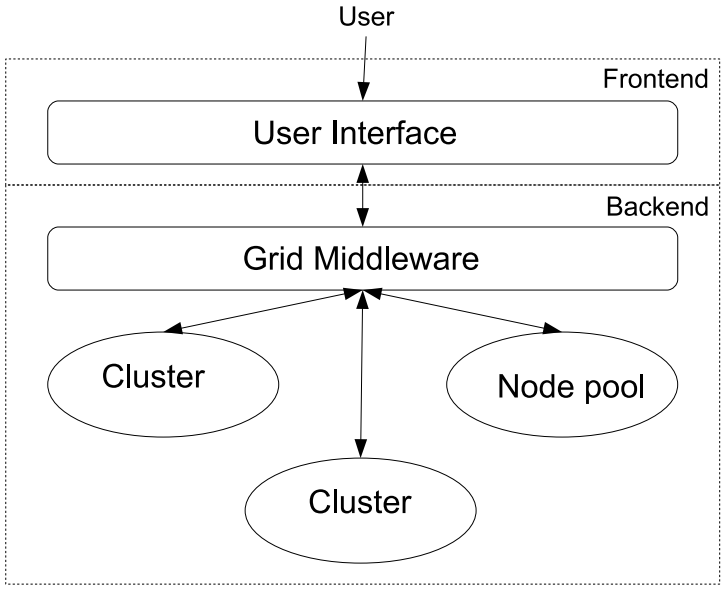


図 3.4　グリッドシステムの構成

ンの実施時に**タスクフローと連携して途中結果を確認できること**と**途中結果の確認に基づいて動作中のプログラムを放棄し、パラメータ修正をしたタスクの位置からシミュレーションが再実行できること**が問題解決システムに求められる。

3.3. 高スループット計算向けシステム構築の現状と問題点

近年では様々なグリッド技術を駆使して問題解決システムが構築されている。問題解決の糸口を発見するためのシミュレーション環境を既存のグリッド技術だけで実現することは難しい。特に、シミュレーション実施者によるシミュレーション実行制御が困難である。

本論文では、図 3.4 のように計算機システム側の要素技術を「バックエンド」、シミュレーション実施者側の要素技術を「フロントエンド」に分類し、高スループット計算向け問題解決システムを構築する上での問題点を示す。

3.3.1　フロントエンドにおける問題点

フロントエンドは、シミュレーション実施者とグリッドミドルウェアとの間を取り持つ。問題解決の糸口を発見するようなシミュレーションを実施するためには、シミュレーショ

ン実施者はフロントエンドと密に接することになる。

グリッドポータルの場合、単一タスクを繰り返すようなシミュレーションのインターフェイスとして適している。ところが、依存関係のある複数のタスクを実行するシミュレーションの場合は想定されていない。また、Web ブラウザを介してシミュレーション実行および放棄の操作を行うことになるため、途中結果を確認しながらパラメータの値を変更してシミュレーションを実施する場合には、高度な Web プログラミングが要求される。

ワークフローシステムの場合、処理の依存関係を記述することと、それに基づいたプログラム実行を可能にする機能は有している。依存関係に基づくプログラム実行に忠実であり、ワークフローによる実行順序のトレースをすることは想定していないため、現在実行中のタスクと途中結果の確認はできない。また、途中結果を再利用するとともに、プログラムに与えるのパラメータを修正して、ワークフローの途中から実行することができない。

シミュレーション実施者の意思決定に基づいてシミュレーションが実施する環境をフロントエンドで支援するためには、以下の機能を追加しなければならない。

- シミュレーションの進捗状況を確認する機能
  複数拠点の計算機を利用する問題解決システムでは、各拠点でグリッド環境の管理ポリシや利用ポリシが異なる。すべての拠点でミドルウェアが統一されていないことを想定し、複数のグリッドミドルウェアに対応したタスク実行状況を取得する。

- シミュレーション途中でパラメータの値を変更するとシミュレーション実行に反映される機能
  依存関係のタスクを実行する場合、シミュレーション実施者の意思に基づいてタスクのパラメータが変更されると、実行中のタスクを中断し、変更されたタスクの位置から再実行する。

- パラメータ変更に対してシミュレーション途中結果を再利用できる機能
  パラメータ変更が発生した場合、どの結果を利用するかを検知し、途中結果を使ってシミュレーションを再開する。

3.3.2 バックエンドにおける問題点

各拠点の計算機は、PBS や SCore などを利用して構築された PC クラスタや、Condor や XtremWeb を利用して事務処理用計算機の遊休状態を利用したものが考えられる。このように、各拠点ごとにタスク管理ポリシが存在し、各拠点のポリシにしたがってタスクが実

行される。これを受け、Globus や UNICORE などのミドルウェアは、各拠点での異なるポリシを吸収し、ひとつの仮想的な計算機として提供する。

大量タスク実行を考慮に入れ、かつパラメータを変化させてシミュレーションを行う場合には、タスク管理ミドルウェアは一つにまとめたいが、現実問題として、タスクはグローバルなタスク管理ミドルウェアとローカルなタスク管理ミドルウェアの 2ヶ所で管理される。また、タスク管理ミドルウェアはタスク管理ミドルウェアは優先度のあるなしに関わらず待ち行列に基づいてタスクが管理される。タスク管理ミドルウェアの階層化はタスクの待ち行列が多段になることから、タスクがどの計算機で実行されているか確認することを難しくしている。このため、階層化されたタスク管理ミドルウェアの下では、タスク管理および実行されたタスクから出力される結果の一元化は困難である。

シミュレーション結果の管理について、シミュレーションの最終結果を得ることが重要であるとの認識から、結果が得られればよいという考え方に基づいて問題解決システムが構築される。そのため、タスク管理とは別の仕組みでシミュレーション結果が管理されている。また、タスクごとではなく、シミュレーション実施ごとに結果が管理されている。

したがって、シミュレーション実施者の意思決定に基づいてシミュレーションが実施する環境をバックエンドで支援するためには、以下の機能を追加しなければならない。

- 階層化されたタスク管理ミドルウェアに対してタスク管理を一元化する機能
  大量タスク実行であればあるパラメータの値をもつタスクが、また、依存関係のあるタスク実行において並行処理が可能なタスクが、どの拠点の計算機で実行されているか把握しやすくする。

- 一元化されたタスク管理のもとでタスク実行と実行中タスクの放棄を実現する機能
  シミュレーション実施者によって、ある特定のパラメータの値をもつタスクの実行や放棄を容易にする。

- 一元化されたタスク管理のもとでタスクの実行結果を管理する機能
  シミュレーション実施者の意思によってシミュレーションの途中から再実行されることを想定し、タスクとパラメータの値に関連づけてシミュレーション結果を管理する。

3.3.3 シミュレーション実施者の要求実現に向けた解決策

シミュレーション実施者はシミュレーションの実施中に改善点をみつけだし、即座に反映したい。つまり、パラメータを変化させたシミュレーションを実施すること、シミュレー

ション結果の管理と再利用を可能にすること、試行錯誤によってシミュレーション結果を改善できることがシミュレーション実施における対話的動作になる。

グリッド環境においてシミュレーションの対話的動作を実現するためには、タスクの実行制御が欠かせない。ところが、タスク実行制御方法の現状はグリッドミドルウェアにおけるフロントエンドとバックエンドにおけるそれぞれの問題点で示したように、グリッドミドルウェアの制約を受ける。シミュレーション実施者はグリッドミドルウェアの操作上の制約を受けながらタスク実行を行い、試行錯誤に基づいたシミュレーションを実施している。

試行錯誤によるシミュレーションを希望するシミュレーション実施者が既存のグリッドミドルウェアを意識することなくシミュレーションが実施できるためには、グリッド環境でのタスク実行操作を軽減させるべきだと考えた。これにより、シミュレーション実施者は問題解決の糸口を発見する行為により多くの時間を割くことができる。

本論文では、タスク実行制御に関わる操作をシステム側で担当し、シミュレーション実施者がなるべく少ないタスク実行制御の操作工数でシミュレーション結果の継続的改善を可能にすることに注目する。具体的な解決策を以下に示す。

- タスクフロー情報の修正と連動したシミュレーション実施
  複雑なシミュレーションになるほどタスクフローに基づいて実行される。既存のグリッドミドルウェアでは、タスクフロー情報を一旦タスク情報に分解し、タスクごとに実行される。そのため、タスクフロー情報を変更したい場合は、タスクフロー情報とタスクの実行状況を対照する作業が必要になる。また、シミュレーション実施中に、タスク情報の修正を実施するためには、実行中タスクの放棄操作、タスクの再実行操作をグリッド環境に対して行わなければならない。

  これらの負担を軽減するために、シミュレーション実施中にタスクフローの情報、その中でもパラメータの値変更といった情報を修正することで、タスク実行に反映する仕組みを設ける。

  これにより、シミュレーション実施者はタスクフロー情報の修正だけでよく、タスク実行や放棄の作業を軽減できる。

- 解候補として可能性の高いパラメータをもつタスクの優先実行
  高スループット計算のように大量のタスクを実行する場合、パラメータリストに基づいてタスクが実行される。タスクフローに基づくシミュレーションと同様、試行錯誤に基づくシミュレーションを行う場合、パラメータリストと実行済みのパラメータ

を対照する作業が必要になる。また、大量タスク実行ではグリッド環境に対してすべてのタスクを投入するかたちになる。グリッド環境のタスク管理ミドルウェアはタスクの実行を管理するだけであり、どのパラメータを有するタスクを実行したかを管理していない。したがって、ある特定のパラメータを設定したタスクを放棄するためには、シミュレーション実施者がパラメータの実行状況を把握しなければならない。

これらの負担を軽減するために、パラメータリストとタスク実行とが連携した仕組みを設ける。また、解探索にかかる時間を短縮するために、パラメータリストの上位に特定のパラメータを並べる仕組みを設ける。

これにより、シミュレーション実施者がパラメータリストの修正、パラメータとタスク実行の関連づけを把握する手間が軽減できる。そして、大量タスク実行による解探索にかかる時間が短縮できる。

3.4. まとめ

本章では、高スループット計算向けグリッドの現状と利用者の意思決定と高スループット計算向けグリッドの構築における課題を論じた。

シミュレーション実施者はパラメータを変化させてシミュレーションを実施したいこと、シミュレーション結果を再利用したいこと、試行錯誤によるシミュレーションの実施を要求していることを示した。そして、シミュレーション実施者によるシミュレーション実行制御の必要性と実現するための課題について、フロントエンドとバックエンドそれぞれの側面から示した。フロントエンドでは、シミュレーションの進捗状況の確認、パラメータの値変更と連動してシミュレーション実行に反映する機能、途中結果を再利用できる機能を、バックエンドでは、タスク管理の一元化、一元化されたタスクの実行と放棄の機能、一元化されたタスク管理のもとでの実行結果の管理機能が必要であることを示した。

最後に、シミュレーション実施者の要求実現に向けた解決策を示し、解決すべき課題を明確にした。既存のグリッドミドルウェアによるシミュレーション結果の継続的改善において、現状はシミュレーション実施者はタスク実行制御の操作で対応している。したがって、シミュレーション実施者によるタスク実行制御の操作工数を軽減し、シミュレーション実施時間のうち、より多くの時間を問題解決の糸口を発見することに利用されることを目指していること、また、その具体的な内容を示した。

第4章

高スループット計算向けグリッドにおけるタスク実行支援フレームワークの提案

4.1. ユーザの経験や知識に基づく操作環境の必要性

産業製品の設計および開発の現場において、複雑な物理現象や工学現象を解決することの重要性が増している。解析的な解法によって複雑な物理現象や工学現象を解決することは困難であり、コンピュータ・シミュレーションを援用した解法が主流である。また、自然科学および工学の分野で発展したコンピュータ・シミュレーションは金融産業、企業経営、経済および政治の現場、いわゆる社会科学の分野でも徐々に使われるようになってきた。

コンピュータ・シミュレーションによる問題解決が欠かせない一方で、シミュレーション実施者が計算機に対して専門的な知識を有しているとは限らない。そこで、シミュレーション実施者が計算機環境を意識することなくシミュレーションに専念するための環境として問題解決環境がある。問題解決環境はシミュレーション実施者が計算機環境と対話することを前提として、計算機に対する専門的知識を有する必要がなくてもシミュレーションに従事できることが可能になる。

従来における問題解決環境の構築は、シミュレーションの実施に必要な計算機の操作を情報技術で補い、シミュレーション実施時の負担を緩和させることが目的である。この考え方は現在でも変わらない。

高スループット計算によるシミュレーションを必要とする現場ではすべてのタスク実行が完了するまでには非常に長い時間を要するため、シミュレーション実施者は意思決定に基づいたシミュレーションを実施したい。ところが、既存技術では、以下のような問題を

抱えているために、シミュレーション実施者の意思決定に基づくシミュレーションが実現できない。

1. フロントエンドではシミュレーションの進捗状況を確認する機能とパラメータ変更に対してシミュレーション結果を再利用できる機能が提供されていないこと。
2. バックエンドでは複数のタスク管理ミドルウェアが動作しており、タスク管理の一元化ができていないこと。
3. 各タスク管理ミドルウェアはタスク実行をするのみである。どのパラメータを実行したかという管理は、シミュレーション実施者が行わなければならない。

そのため、従来の高スループット計算向けグリッドは、シミュレーション実施者の知識と経験と関係なく、最適解の探索のために無駄と考えられるようなタスクの実行も行わなければならなかった。

本章では、シミュレーション実施者の要求を満たすための方法として、高スループット計算向けグリッドにおけるタスク実行支援フレームワークを提案する。

4.2. タスク実行支援フレームワークの提案

4.2.1 タスク実行を支援する層の導入

既存技術によって構築される問題解決システムは、図 4.1 左側に示すように、グリッド基盤側はバックエンドが、シミュレーション実施者の操作はフロントエンドが担当する。

このモデルでは、シミュレーション途中でのパラメータ変更と実行中のタスクへの反映において、実行中タスクの停止、パラメータ変更作業、タスクの再実行作業といった作業をシミュレーション実施者が操作しなければならない。シミュレーション実施者は大量タスクを含むシミュレーションの繰り返しにおいて操作上の負担を強いられる。

大量タスク実行のように、タスク間で通信が殆んど発生しないような場合は大量の計算機を利用することが望ましく、かつ試行錯誤に基づいた大量タスク実行によるシミュレーションを実施する場合には、シミュレーション実施者から見てタスクひとつひとつのパラメータの変更が容易でなければならない。ところが、このモデルでは特定のグリッドミドルウェアに依存したかたちで問題解決システムを構築することが前提である。

計算機台数の規模拡大を考慮するためには、NAREGI Grid Middleware のようにグリッドミドルウェアの操作面での統合化を実現するか、中田らが提案した Jojo[51] と呼ばれる

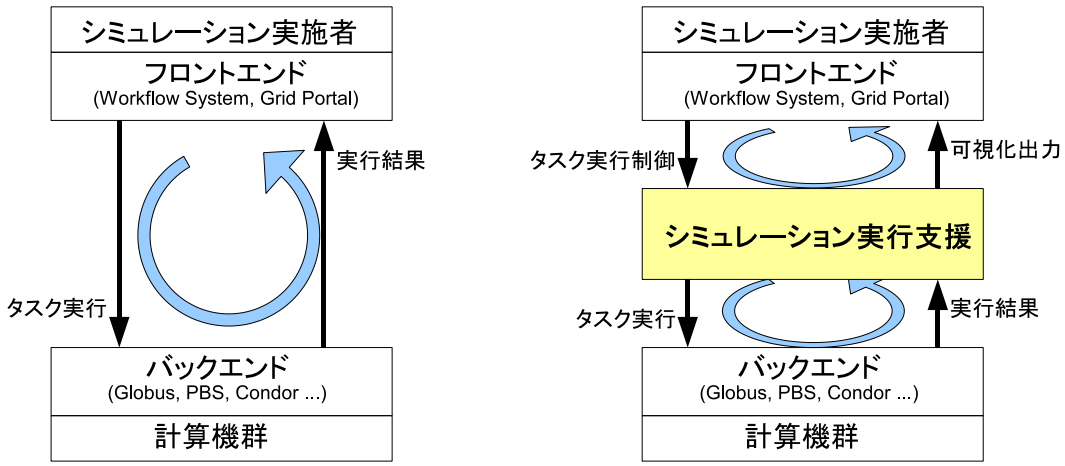


図 4.1 グリッド技術を用いた問題解決システムとシミュレーション実施者との関係 (左: 既存技術によるモデル, 右: 実施者の要求モデル)

ミドルウェアのようにシミュレーション実施者側が階層的なグリッド環境に合わせてタスク管理の統一化を実現する機構をアプリケーション側に追加する方法が考えられる。いずれもシミュレーション実施者が自主的にタスク管理をしなければならない問題を含んでいるので、シミュレーション実施者の負担を軽減するための解決策ではない。

本論文では、既存グリッドシステムを活用することを前提に、シミュレーション実施者の意思決定に基づく問題解決システムの構築を実現する。そのためには、フロントエンドとバックエンドとの間にタスク実行を支援する層を導入し (図 4.1 右側)、問題解決システム側でタスク管理の一元化を実現するとともに、タスクと計算結果の関連付けを容易にする。

タスク実行を支援する層を導入する理由は以下のとおりである。

- **バックエンド側のメリット**
  バックエンド側については、複数のタスク管理ミドルウェアの操作上の違いを吸収することが可能である。(タスク実行について例えば、Globus Toolkit であれば globus-job-run だが、Condor では condor_submit が使われる。) 統一的なタスク実行コマンドに置き換えることで、シミュレーション実施者は 1 つの実行コマンドで操作することができるようになる。

- **フロントエンド側のメリット**
  フロントエンド側については、タスク管理の統一化が可能になる。従来、タスク実行と計算結果の関連付け、およびタスク実行とパラメータの関連付けはシミュレーショ

ン実施者が手動で行わなければならなかった。支援層を導入することでタスク実行と計算結果、タスク実行とパラメータの関係づけが容易になる。また、シミュレーション実施者の操作がタスク情報の修正だけになる。

4.2.2 タスク実行支援フレームワーク

本論文では、図 4.1 右側に示すようなモデルを実現するために、タスク実行支援フレームワークを提案する。

まず、タスク実行支援フレームワークの全体像を図 4.2 に示す。タスク実行支援フレームワークは、グリッドミドルウェアで構成されたグリッド環境を 1 つのモジュールと考える。本論文では、グリッド環境側を下位、シミュレーション実施者側を上位とし、グリッド環境より上位に対して、プログラム実行モジュール、計算結果解析モジュール、システム状況監視モジュール、ユーザインターフェイスの 4 つのモジュールで構成する。これらのモジュールは ISO9000 において継続的改善を実現するための具体的なモデルである PDCA サイクルに基づいて配置した。

タスク実行支援フレームワークにおける動作要素とそれに対応するモジュールの関係を図 4.3 に示す。破線で囲まれた部分はシミュレーション実施者の動作と直接の関係はないが、統一したタスク管理の実現とシミュレーション結果の確認動作の促進に必要なため追加した。

次に、各モジュールの特徴について説明する。

- **ユーザインターフェイス**
  ユーザインターフェイスはシミュレーション実施者とグリッド環境との架け橋の役目をするモジュールである。PDCA サイクルにおける Action に相当する。

  ユーザインターフェイスでは、シミュレーション実施者の動作要素として、プログラムの選定、入力データの準備、入力パラメータの設定、プログラム実行順序の決定、結果確認、処理状況確認、を担当する。

  ユーザインターフェイスは主として、プログラム実行のための操作インターフェイス、シミュレーションの実行状況を把握するためのインターフェイス、計算結果を出力するためのインターフェイスで構成される。

- **プログラム実行モジュール**
  プログラム実行モジュールはシミュレーション実施者の操作に基づいてプログラムの

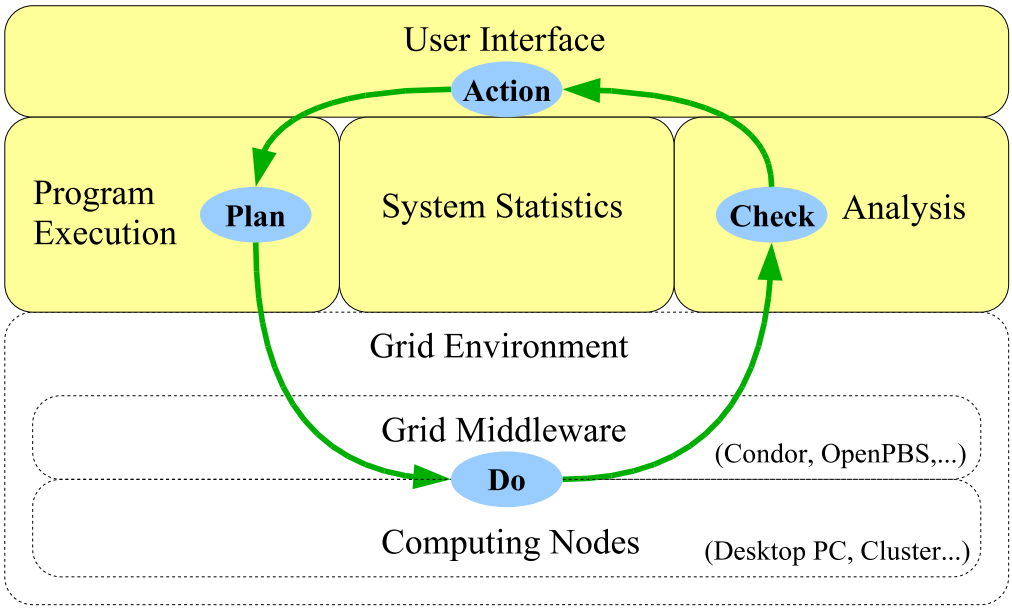


図 4.2 タスク実行支援フレームワークの全体像

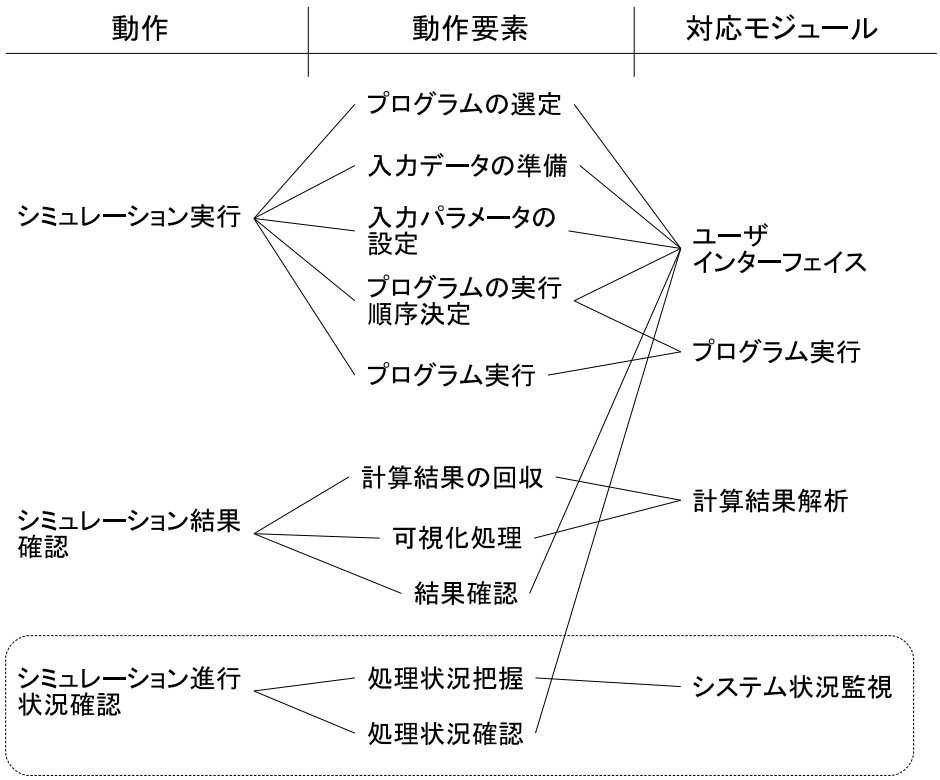


図 4.3 動作要素と対応モジュールとの関係

実行および停止を行うモジュールである。PDCA サイクルにおける Plan に相当する。

プログラム実行モジュールでは、シミュレーション実施者の動作要素として、シミュレーション実施者におけるプログラム実行順序の決定およびプログラム実行を担当する。

グリッドミドルウェアは基本的にコマンド入力による操作が必要であるが、コマンド入力による操作は複雑である。また、採用するグリッドミドルウェアによってコマンドの名前が異なる。そこで、プログラム実行モジュールでは、シミュレーション実施者が必要であろう操作とグリッドミドルウェアのコマンド体系との間を関連づけることで統一的な操作を可能にする。

- **計算結果解析モジュール**
  計算結果解析モジュールはプログラム実行結果として出力されるデータを回収し、シミュレーション実施者が意思決定をするために必要な形式へとデータを変換するモジュールである。PDCA サイクルにおける Check に相当する。

  計算結果解析モジュールでは、シミュレーション実施者の動作要素として、計算結果の回収および可視化処理を担当する。

  高スループット計算の場合、大量のプログラム実行が行われるため、プログラム実行結果が大量に出力される。プログラム実行結果を回収する際は、出力された結果と入力パラメータとの関連づけを行う。また、実行結果の可視化処理を前もって行い、シミュレーション実施者からの結果確認要求に応じて可視化情報を提示する。

- **システム状況監視モジュール**
  システム状況監視モジュールでは、グリッド環境の負荷情報およびプログラム実行状況を把握するモジュールである。PDCA サイクルと直接の関わりはないが、シミュレーション実施者の動作や他のモジュールとの連携のために設けた。

  計算結果解析モジュールでは、PDCA サイクルに直接関わらないが、シミュレーション実施者の動作要素に準じたものとして、処理状況の把握を担当する。

  まず、シミュレーションは長時間の実行を余儀なくされる。本フレームワークでは、シミュレーション実施者による意思決定が必要であるが、シミュレーション実施者は常に端末の前にいるわけではない。次に、高スループット計算では大量のプログラムを実行する。シミュレーション実施者の意思決定によって再実行をしたとき、その時点で実行中のプログラムを停止させる作業が必要になる。プログラムの停止作業に

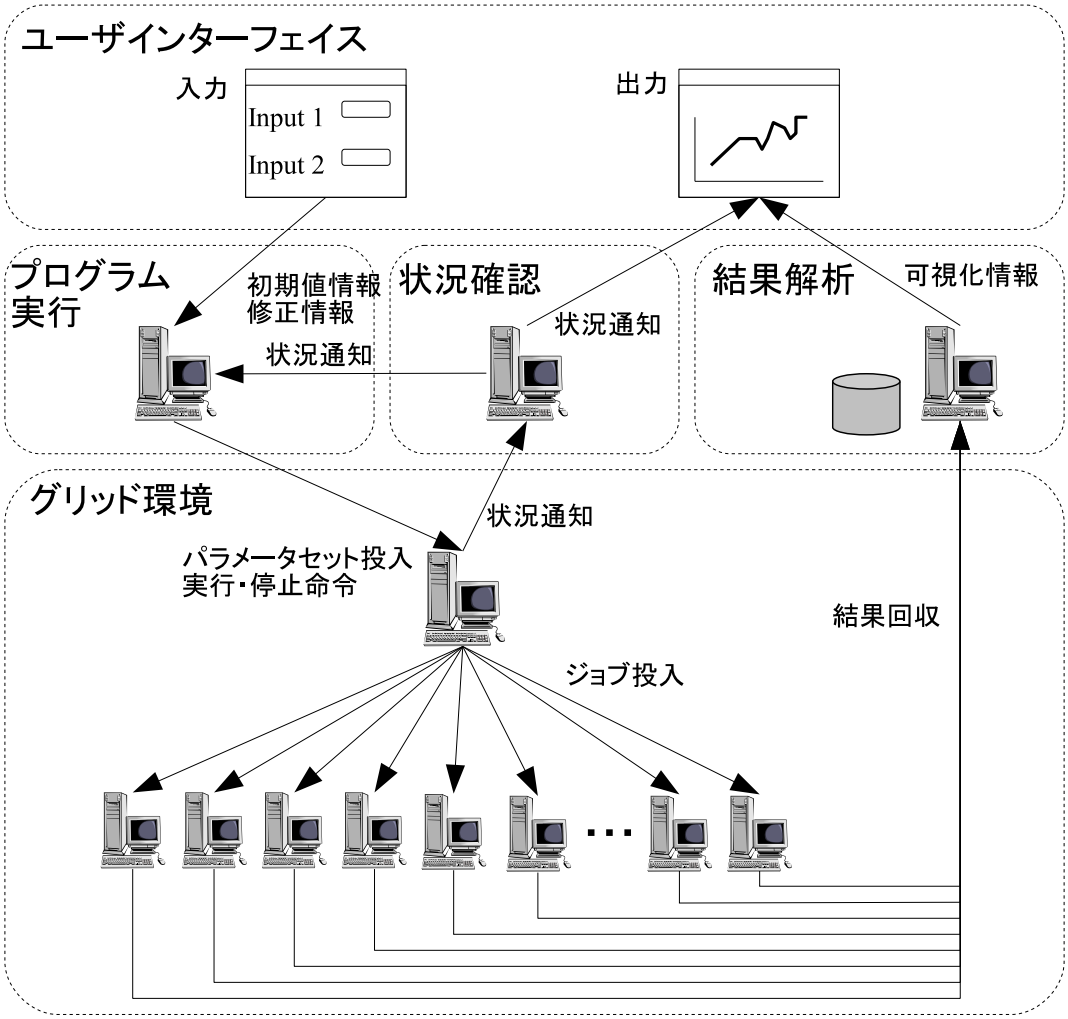


図 4.4 タスク実行支援フレームワークによるシステム構成

は、大量のプログラム実行を把握していることが要求される。このような理由から、シミュレーションの実行がどこまで処理が進行しているか確認できなければならない。よって、システム状況監視モジュールを導入する。

提案するタスク実行支援フレームワークに基づいて構築した問題解決システムの構成は図4.4のようになる。

表 4.1 タスク一覧

No.	プログラム	パラメータ
1	preprocess	-a 120 -b 512 -c 1024
2	mkparam	1024
3	main1	1 128
4	main2	-d 16
5	postprocess	output.dat

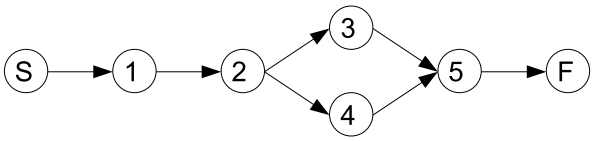


図 4.5 タスク実行順

4.2.3 タスク実行支援フレームワークにおけるタスクの管理

提案するタスク実行支援フレームワークでは、複数拠点のグリッド環境を利用するため、複数のタスク管理ミドルウェアを包含するようなタスク管理を実現する。ここでは、各拠点のタスク管理ミドルウェアにアクセスするための仕組みについて説明する。

タスク情報およびタスクフロー情報の定義

複数のグリッドミドルウェアが採用されている環境において、統一したタスク管理を行うためには、タスク実行支援フレームワークでタスクの管理を行う必要がある。ここでは、例として表 4.1 に示したタスクを図 4.5 に示す順番に実行する際のタスク管理について説明する。

まず、タスク実行支援フレームワークにおけるタスク情報とタスクフロー情報について以下に示す。

- **タスク情報**
  シミュレーション実施者は、図 4.7 に示すデータ構造をもつタスク情報を作成する。タスク情報は、{ **タスク番号, プログラム, パラメータ, 次タスク番号** } の 4 つ組の情報をもつ。具体的には、

  1 行目: タスク識別番号

  2 行目: プログラム (1 タスクに対してエントリーは 1 つのみ)

  3 行目: パラメータ (複数ある場合はスペースで区切る)

  4 行目: 次に実行するタスクの番号 (複数ある場合はスペースで区切る)

5 行目: (空行)

の順番にタスクに関する情報が記述される。

以下はタスク識別子 1 に関するタスク情報の記述例である。(なお、左側に示す数字は行番号を意味する。)

```
1: 1
2: preprocess
3: -a 120 -b 512 -c 1024
4: 2
5:
```

- **タスクフロー情報**
  プログラムの実行順序を示すタスクフロー情報について、タスク実行支援フレームワークにおけるタスクフローでは、各タスクにおける前後のタスクの依存関係が一意に定まる点に注目し、DAG 形式を採用した。したがって、タスク実行支援フレームワークにおけるタスクフロー情報は、次に実行すべきタスクの識別子を各タスク情報の 4 行目に記述する。また、タスクフロー全体は図 4.6 のように示される。

  タスクフロー情報の具体的な記述方法を以下に示す。

  – 次に実行するタスクが 1 つの場合
    タスク情報の 4 行目に次に実行するタスク識別子を記述する。タスク識別子 1 における例を以下に示す。

    ```
    1: 1
    2: preprocess
    3: -a 120 -b 512 -c 1024
    4: 2
    5:
    ```

  – 次に実行するタスクが複数ある場合
    図 4.5 においては、タスク識別子 2 がタスク識別子 3 および 4 を実行する場合に相当する。タスクフロー情報を意味する 4 行目には次に実行するタスク識別子の番号を記述する。各タスク識別子はスペースで区切られる。

表 4.2　データベース操作インターフェイス

	インターフェイス	操作結果
タスク情報登録	insert_new_id(タスクフロー番号, タスク番号, 値)	データベース登録
タスク情報修正	modify_id(タスクフロー番号, タスク番号, 値)	データベース修正
タスクフロー情報取得	get_record(タスクフロー番号)	タスクフロー情報
タスクフロー実行状況取得	check_id(タスクフロー番号)	タスクフロー情報

```
1: 2
2: mkparam
3: 1024
4: 3 4
5:
```

– 最終タスクの場合
図 4.5 においては、タスク識別子 5 が相当する。最終タスクの場合、タスクフロー情報を意味する 4 行目は空行にする。

```
1: 5
2: postprocess
3: output.dat
4:
5:
```

タスク情報とタスクフロー情報の管理

タスクの管理はシステム状況監視モジュールで行われる。シミュレーション実施者によって入力されたタスクフローデータはタスク毎にタスク情報格納データベースに登録される。各タスクの情報は図 4.7 に示すように、{**タスクフロー番号, タスク番号, プログラム, パラメータ, 次タスク番号, タスクの実行状態, タスク実行先, 実行先から発行されたタスク ID**} の 8 つ組の情報で構成される。

```
 1: 1
 2: preprocess
 3: -a 120 -b 512 -c 1024
 4: 2
 5:
 6: 2
 7: mkparam
 8: 1024
 9: 3 4
10:
11: 3
12: main1
13: 1 128
14: 5
15:
16: 4
17: main2
18: -d 16
19: 5
20:
21: 5
22: postprocess
23: output.dat
24:
25:
```

図 4.6 タスクフロー記述例 (左側は行番号を意味する。)

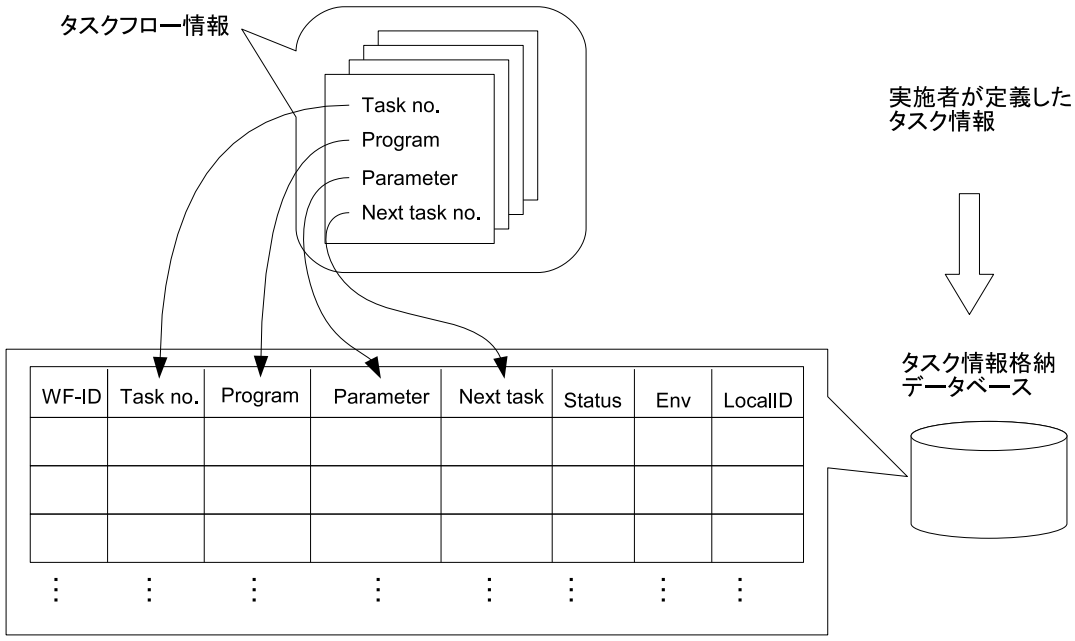


図 4.7　タスク情報格納データベースへの変換

このうち、タスク番号、プログラム、パラメータ、次タスク番号の 4 つの情報については、ユーザが定義したタスク情報を書き込む。また、タスクフロー番号、については、システム状況監視モジュールが適切な値を書き込む。最後に、タスクの実行状態、タスク実行先、実行先から発行されたタスク ID の 3 つの情報については、タスクが実行された段階でプログラム実行モジュールからそれぞれの値を取得し、システム状況監視モジュールが書き込む。

システム状況監視モジュールはタスクフローデータがタスク格納データベースに登録された時点で、タスクフロー番号をシミュレーション実施者に提示する。

タスク情報格納データベースは PostgreSQL といったリレーショナルデータベースシステムを用いて構築される。タスク情報格納データベースを操作するためのインターフェイスを表 4.2 に示す。データベース操作インターフェイスはタスク情報の登録、タスク情報の修正、タスクフロー情報の取得、タスクフロー実行状況の確認の 4 つのインターフェイスで構成される。

タスクの実行方針

タスク実行支援フレームワークは複数拠点の計算機を積極的に利用する方針であるため、図 4.2 に示したように、タスクの計算機への割り当ては既存のグリッドミドルウェアがもつ機能を利用する。したがって、各拠点のタスク管理ミドルウェアのタスク実行ポリシのもとでタスクが実行される。各拠点のタスク管理ミドルウェアはタスク実行支援フレームワークと独立して動作する。

ここで、タスク実行支援フレームワークにおけるタスク実行の振る舞いを考える。タスク実行支援フレームワークでは、各拠点のタスク管理ミドルウェアにタスクの実行を依頼していることになる。言い換えれば、タスク実行の権限を委譲しているといえる。このとき、タスクフローに基づくシミュレーション実施の場合、各タスク管理ミドルウェアに依頼する方法が問題になる。考えられる解決策としては、

- タスク実行支援フレームワークにおいて、タスクフロー情報をタスク情報に変換するとともに、その情報を保持しておき、適切な拠点にタスクを割り当てる。(逐次実行型)
- タスクフロー情報にあるタスクの実行をすべてをタスク管理ミドルウェアに丸投げする。(要求駆動型)

が挙げられる。いずれの場合も、タスクの放棄、再実行の作業をシミュレーション実施者が要求した段階で対応することが可能である。前者の場合はタスクごとに対応することに対して、後者の場合はタスクフローごとに対応することになる。

タスク実行支援フレームワークでは、シミュレーションの流れに注目しているためタスクフローに基づいた実施が必要になること、タスクの実行場所選択は既存のグリッドミドルウェアに任せたほうが適切な計算機を選択できることに注目し、後者の要求駆動型を採用する。

タスクの実行および停止

タスク実行支援フレームワークでは、複数のグリッドミドルウェアを利用することを想定している。そこで、シミュレーション実施者が統一的なタスク実行の操作を実現するために、表 4.3 に示す操作コマンドを設けた。タスク操作、タスクフロー操作を担当するユーザインターフェイスを実装する際、これらの操作コマンドを利用する。

操作コマンドを利用したタスク実行およびタスク停止の操作の流れを以下に示す (図 4.8)。

表 4.3 操作コマンド一覧

コマンド	タスクフロー識別子の有無	タスク識別子の有無	内容
接続および終了			
HELO	×	×	接続確立要求
QUIT	×	×	接続終了
タスク操作			
RUN	○	○	タスク実行
STOP	○	○	タスク強制終了
PAUSE	○	○	タスク一時停止
タスクフロー操作			
SUBMIT	○	×	タスクフロー情報の投入
DELETE	○	×	タスクフロー情報の破棄
FINJOB	○	×	終了タスク情報の提示
STATWK	○	×	全タスク実行状況の提示
STATFL	○	×	タスクフロー進行状況の提示
その他			
HELP	×	×	ヘルプメッセージ要求

1. シミュレーションを開始する際の準備

   (a) シミュレーション実施者はユーザインターフェイスを通じてタスクフローデータを投入する。このとき、タスクフローデータは ftp や scp などの転送ツールを利用する。

   (b) SUBMIT コマンドを用いてタスクフロー情報を投入する。

   (c) システム状況監視モジュールから出力されたタスクフロー番号を受けとる。

2. タスク実行操作が行われるとき

   (a) シミュレーション実施者は、RUN コマンドを用いて該当するタスクフロー番号の実行命令をプログラム実行モジュールに依頼する。

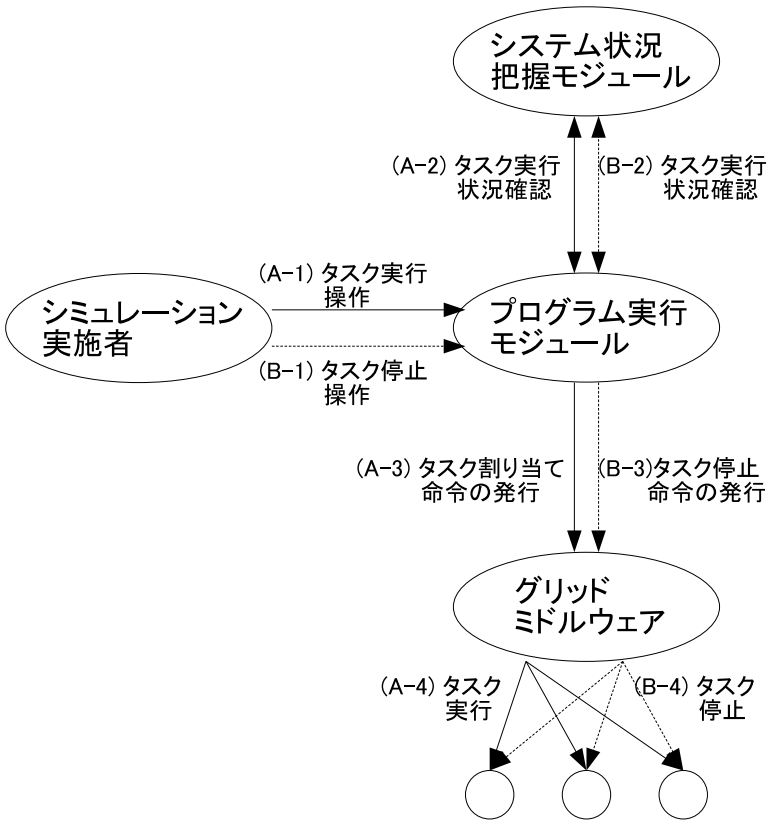


図 4.8 タスク実行およびタスク停止の動作の流れ

(b) プログラム実行モジュールは、システム状況把握モジュールに対して、該当するタスクフロー番号のタスク情報をすべて受けとる。

(c) プログラム実行モジュールはタスクの実行順序にしたがって実行される。タスクを実行した段階で得られるタスク実行先と実行先が発行した ID をシステム情報把握モジュールに送り、タスク情報格納データベースを更新する。

3. タスク停止操作が行われるとき

(a) シミュレーション実施者は、STOP コマンドを用いて該当するタスクフロー番号の停止操作をプログラム実行モジュールに依頼する。

(b) プログラム実行モジュールは、システム状況把握モジュールに対して、該当するタスクフロー番号のタスク情報をすべて受けとる。

(c) プログラム実行モジュールはタスク情報から、未実行のタスクと実行中のタスクを取り出し、該当するグリッド環境に対してタスク停止操作のコマンドを発

呼し、該当するタスクの停止処理を行う。停止処理ののち、システム情報把握モジュールに対してタスクの実行状態を未実行に修正する命令をなげる。

タスク投入スクリプトの作成

既存のタスク管理ミドルウェアはそれに適したタスク投入スクリプトを準備する必要がある。原則としてシミュレーション実施者がタスク投入スクリプトを記述し、それを投入することでタスクの実行が開始される。

タスク実行支援フレームワークでは、タスク情報から、既存のタスク管理ミドルウェアに適したタスク投入スクリプトを生成する。具体的な例として、PC クラスタのタスク管理ミドルウェアである OpenPBS の場合とデスクトップグリッドのタスク管理ミドルウェアである Condor の場合について示す。

- **OpenPBS の場合**

  PC クラスタ向けタスク管理ミドルウェアとして有名なものとしては、OpenPBS、PBS Pro、Sun Grid Engine とある。ここでは OpenPBS を例に説明するが、Sun Grid Engine でも同様のタスク投入スクリプトを実現できる。

  PC クラスタ向けタスク管理ミドルウェアでは、クラスタ内で閉じているため、分散した計算結果を回収する枠組みをあらかじめ準備しておく。例えば、Network File System(NFS) や GSI-SFS[52] などの共有ファイルシステムを用いることで、各クラスタノードからのアクセスが可能になる。

  タスク投入スクリプトを図 4.9 に示す。PC クラスタ向けタスク管理ミドルウェアはシェルスクリプトの書式に基づいて記述される。

  PC クラスタ向けタスク管理ミドルウェアには各タスク実行が終了したことを表現する機能がない。タスク実行支援フレームワークでは、どのタスクが完了したかを把握する必要があるため、`streport` と呼ばれるプログラムを用いて、明示的にタスク実行を確認する。図 4.9 に示すように、`streport` はシステム状況把握モジュールが動作しているサーバ名とタスク番号が引数であり、システム状況把握モジュールに対してタスク実行が完了したことを通知をする。

  タスクの実行はプログラム実行モジュールにおいて `qsub` というコマンドを、同様にタスクの停止は `qdel` を発呼する。

- **Condor の場合**

Condor の場合、特殊なタスク実行スクリプトが必要になる。各タスクに対して図 4.10 に示すようなタスク投入スクリプトを作成する。

Condor では OpenPBS とは異なりタスク管理サーバと計算ノードとの双方で必要なデータを転送できる機能をもっているため、計算結果を回収する枠組みを設ける必要はない。その代わり、should_transfer_files、when_to_transfer_output、transfer_input_files、transfer_output_files の 4 種類の設定語を使って、双方で転送するデータを記述する。(図 4.10 の例では、各タスクの実行が完了すると実行結果が戻ってくる設定になっている。)

Condor において各タスク実行が終了したことを確認する場合は、(1) 実行状況のログを確認する方法、(2) transfer_output_files に書かれているファイルが出力されているか確認する方法が考えられる。タスク実行支援フレームワークでは、ファイルが出力されているかを確認する後者の方法を用いる。

タスクの実行はプログラム実行モジュールにおいて condor_submit というコマンドを、同様にタスクの停止は condor_rm を発呼する。

大量タスク実行の単一タスク化

高スループット計算では大量のタスク実行が行われる。パラメータスウィープ型シミュレーションの場合、想定されるパラメータの組をひとつずつ実行するためには、タスク投入スクリプトをパラメータの数だけ作成することになる。シミュレーション実施者のタスク投入準備の手間が増える。大量タスク実行の場合、パラメータを記述したリスト、いわゆるパラメータリストを作成し、これに基づいてパラメータごとのタスク実行を行うことで、シミュレーション実施者のタスク投入準備の手間を軽減する。

タスク実行支援フレームワークでは、タスク実行の際に用いられるプログラムが 1 種類であることに着目し、大量タスクは特殊な単一タスクとして取り扱う。大量タスク実行を単一タスクとして取り扱うために、付録 A.1 に示す大量タスク実行スクリプトを用いる。

大量タスク実行スクリプトは、大量タスク実行をするためのプログラム名とパラメータリストを引数とする。付録 A.1 のスクリプトの場合、以下のように実行する。

```
submitter.pl <プログラム> <パラメータリストファイル>
```

大量タスク実行スクリプトをタスク情報に記述する例を以下に示す。例では、大量タス

```
#!/bin/sh

./calculateVersion4 250 750 -1 -1 2000 2500 2 2  4000 4000 1000 \\
4000 4000 1000 100  0.7  output_250-750_-1_-1_2000_2500_2_2_\\
4000_4000_1000_4000_4000_1000_100_0.7.dat

sleep 5

./streport statistics  3
```

図 4.9　OpenPBS におけるジョブ投入用スクリプト作成例

```
UNIVERSE = vanilla
Executable = calculateVersion4
Log=calcv4-62.log
Output = calcv4-62.out
Error = calcv4-62.err

Arguments = 250 750 -1 -1 2000 2500 2 2 4000 4000 1000 4000 4000 \\
1000   100  0.7  output_250-750_-1_-1_2000_2500_2_2_4000_4000_1000_\\
4000_4000_1000_100_0.7.dat

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

transfer_input_files = AGE.txt, C18.txt, ED.txt, HI.txt, parameter.txt, \\
WORK.txt
transfer_output_files = output_250-750_-1_-1_2000_2500_2_2_4000_4000_\\
1000_4000_4000_1000_100_0.7.dat

Queue
```

図 4.10　Condor におけるジョブ投入用スクリプト作成例

```
0 1000 1000 1000 -1 -1 2 2 1000 1000 1000 4000 4000 4000 100 0.7
0 2000 1000 1000 -1 -1 2 2 1000 1000 1000 4000 4000 1000 100 0.7
0 3000 1000 1000 -1 -1 2 2 1500 4000 1000 4000 4000 1000 100 0.7
0 4000 1000 1000 -1 -1 2 2 2500 2500 1000 4000 4000 1000 100 0.7
```

図 4.11　パラメータリスト例

ク実行を必要とするプログラムを match、パラメータリストファイルを taskparam.lst とした場合である。

```
1: 5
2: submitter.pl
3: match taskparam.lst
4: 6
5:
```

ここで、パラメータリストファイルである taskparam.lst の例を図 4.11 に示す。1 回のタスク実行に必要なパラメータの組は 1 行で記述する。したがって、定義されている行数分がタスク実行数となる。

4.2.4 タスクフロー差分実行機構による柔軟なシミュレーション実行制御

シミュレーションが複雑なほど複数の処理が行われる。そのため、タスク間に依存関係が発生する。タスク実行に依存関係をもつようなシミュレーションの場合、一般的にタスクフローに基づいて実行される。タスクフローに基づいて実行するために必要であるタスクフローシステムでは、タスクフローに記述されたとおりに実行する。

ここで、シミュレーション結果の継続的改善を実現することを考える。シミュレーション結果の継続的改善は、シミュレーション実施者がタスク実行によって得られた結果を検討し、パラメータの設定を変更するか否かを判断することである。シミュレーション実施者がパラメータ設定を変更すると判断した場合、パラメータ設定を変更したタスクの再実行を行う。同様に、タスクフローに基づいてひととおり終えたシミュレーション結果を検討し、あるタスクのパラメータのみ変更したシミュレーションを実施する例も考えられる。

このとき、既存のタスクフローシステムでは、タスクフローに記述されたとおりにプログラムが実行されるため、パラメータ設定を変更する判断をした場合、該当するタスクのパラメータを変更し、かつ、タスクフローの先頭から実行しなおすことになる。

例えば、図 4.5 に示す DAG 形式のタスクフローにおいて、タスク番号 4 の設定を変更したとする。既存のタスクフローシステムでは、タスク番号 4 の設定を変更することは可能であるが、

1. タスクフローに基づいたシミュレーション実行を停止する
2. タスク番号 4 の設定を変更したタスクフローを記述する

3. 記述し直したタスクフローをタスクフローシステムに投入し、タスクフローの先頭から実行する

という手順を踏む。この結果、既にシミュレーション結果が得られている分まで実行することになる。シミュレーション結果の継続的な改善を考えた場合、タスクフローの先頭からシミュレーションが実行されるのはタスク実行の柔軟性に欠ける。シミュレーションの再実行では、1 つのタスク実行が長時間になるほど、重複したタスク実行は無駄である。

そこで、シミュレーションの再実行において、タスクフローの適切な位置からタスク実行を可能にするために、タスク実行支援フレームワークでは、PDCA サイクルにおける Action と Plan の動作に着目し、タスクフローの情報修正をするだけでタスクフローによる再実行を可能にする機構を提案する。これをタスクフロー差分実行機構と呼ぶ。

タスクフロー差分実行機構は、シミュレーション実施者によって修正されたタスクフローの情報から、修正箇所からのタスクフローを作成する。そして、修正箇所からのタスクフローデータに基づいてシミュレーションの実行が再開される。このとき、修正前のタスクフローによるシミュレーションの停止、修正タスクフローデータによるシミュレーションの再開動作が発生するが、これらの動作はタスクフロー差分実行機構が担当する。タスクフロー差分実行機構によって、シミュレーション実施者はタスクフローの修正に注力できる。

図 4.5 に示すようなタスクフローを用いた先ほどの例の場合、シミュレーション実施者は

1. タスク番号 4 の設定を変更する

だけでよい。それ以外の操作、すなわちタスクフローに基づいたシミュレーション実行を停止する操作、記述しなおしたタスクフローをタスクフローシステムに投入する操作、適切な位置からタスクフローに基づいて実行する操作はタスクフロー差分実行機構が支援する。また、タスクフロー差分実行機構によって作成されるタスクフローは、一つ前の実行結果を利用するため、S → 4 → 5 → F となり、タスク実行数が低減できる。

タスクフロー差分実行機構の処理の流れを説明する (図 4.12)。タスクフロー差分実行機構は既にタスクフローによる実行がなされている状況で動作する。パラメータの修正が必要になったとき、シミュレーション実施者はユーザインターフェイスを利用してタスクフローデータを修正する。このデータを修正タスクフローデータと呼ぶ。

1. シミュレーション実施者は修正タスクフローデータをプログラム実行モジュールに送信する。

2. プログラム実行モジュールが修正タスクフローデータを受信した時点で、修正前のタスクフローの進捗状況をシステム状況監視モジュールに問い合わせる。

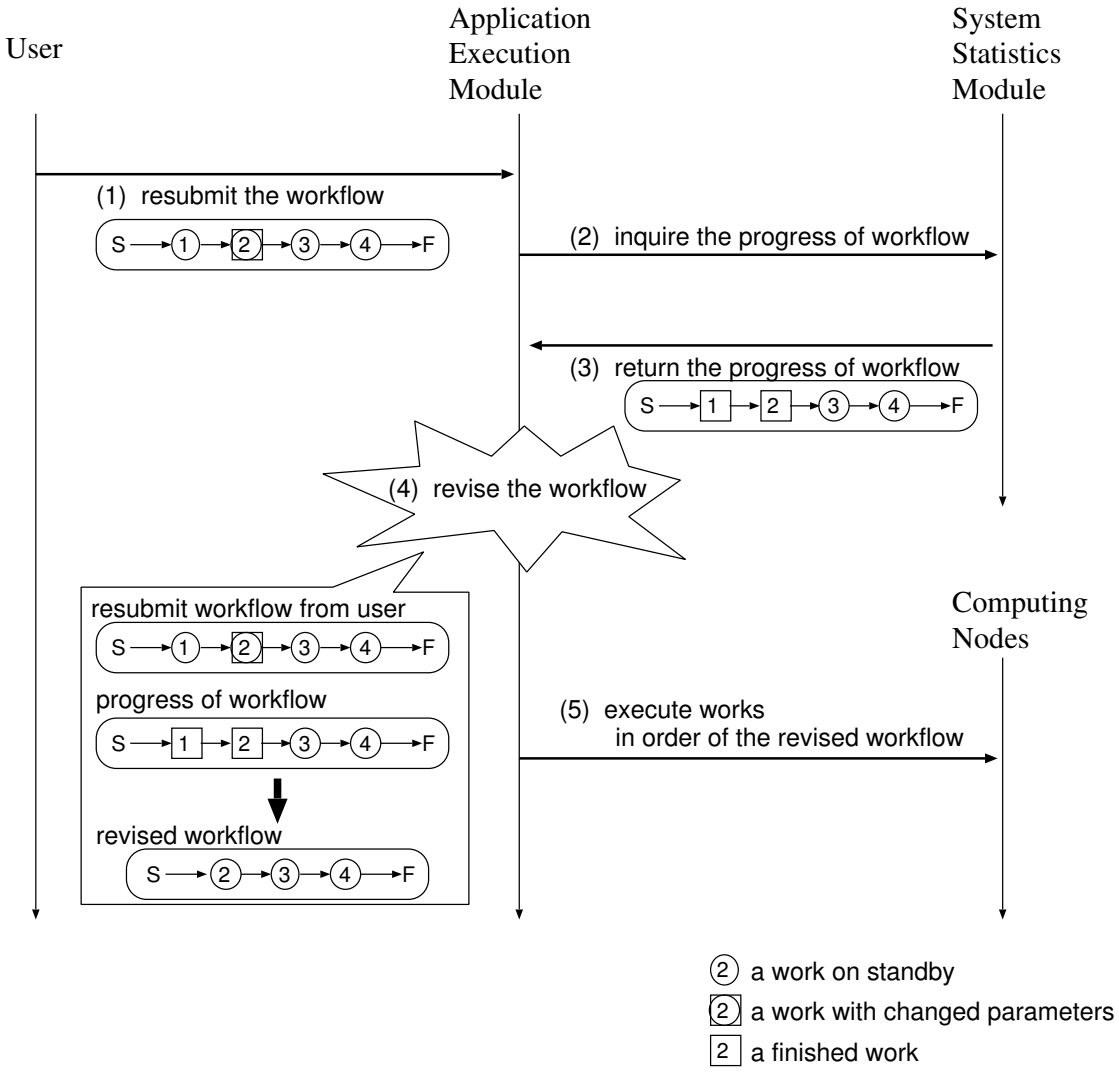


図 4.12 タスクフロー差分実行機構の処理手順

表 4.4　プログラム実行モジュール内のプログラムインターフェイス

	インターフェイス	返り値
タスクフロー受付	selector(タスクフロー番号)	再実行: 1 それ以外: 0
タスクフロー確認	diff_wf(タスクフロー番号)	再実行: 1 それ以外: 0
差分タスクフロー作成	make_wf(タスクフロー番号)	タスクフロー ファイル
タスクフロー放棄	task_abort(タスクフロー番号)	成功: 1 失敗: 0
タスクフロー割り当て	task_dispatch(タスクフロー番号)	成功: 1 失敗: 0
優先パラメータリスト作成	modify_plist(優先パラメータ情報)	優先パラメータ リストファイル

3. システム状況監視モジュールはプログラム実行モジュールからの問い合わせに対して、問い合わせ時点でのタスクフローの進捗状況 (進捗タスクフローデータ) を回答する。

4. プログラム実行モジュールでは、システム状況監視モジュールから受け取った進捗タスクフローデータと修正タスクフローデータとを対照し、新たに実行するタスクフローを作成する。このときできたタスクフローを差分タスクフローデータと呼ぶ。

5. 前のタスクフローに基づいて実行しているタスクを放棄するとともに、差分タスクフローデータに基づくタスクの実行を始める。

プログラム実行モジュール内でのタスクフロー差分実行機構の振る舞いについて説明する。図 4.13 はタスクフロー差分実行機構のプログラム実行モジュール内での流れをプログラムインターフェイスと対応づけたものである。また、タスクフロー差分実行機構で用いられるプログラムインターフェイスを表 4.4 に示す。

1. シミュレーション実施者はタスク実行支援モジュールに対してタスクフローを投入する。タスクフローがプログラム実行支援モジュールに送られると、selector() に処理が移る。

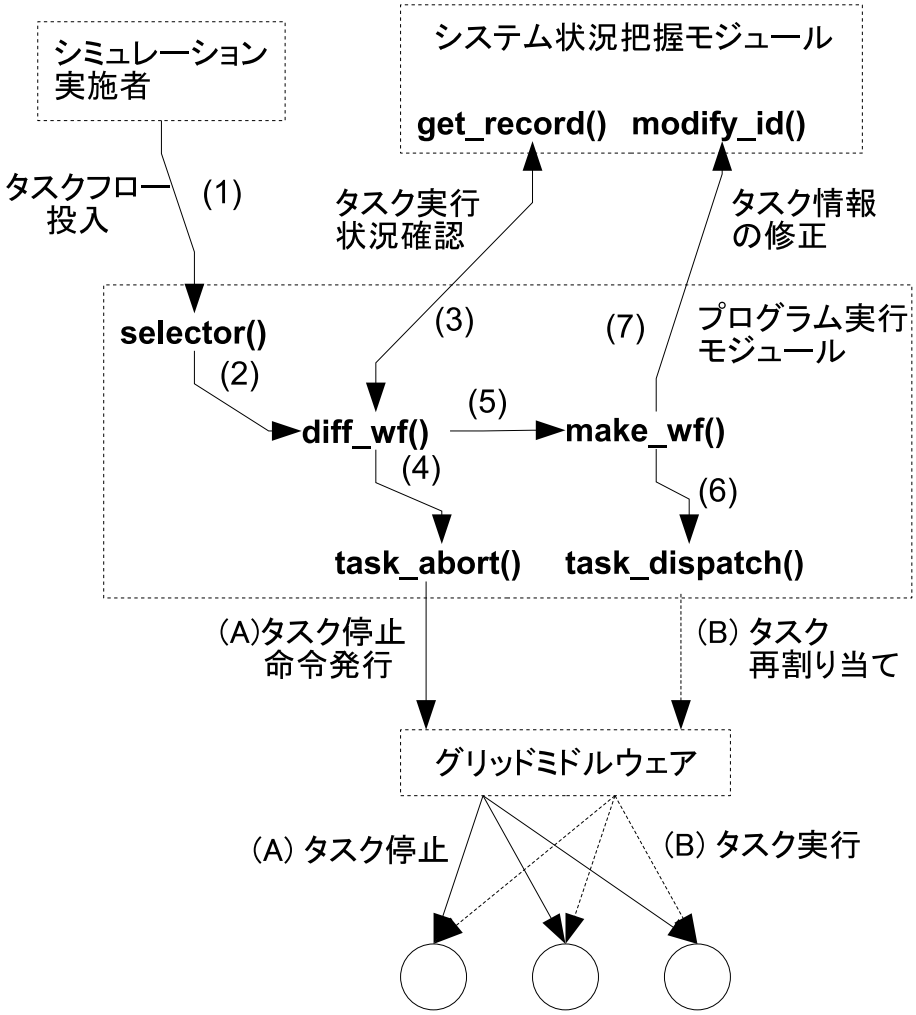


図 4.13 タスク差分実行機構のプログラム実行モジュール内での流れ

2. selector() ではタスクフロー番号から動作中のタスクフローであるかを確認する。タスクフローの再実行であることを確認すると、diff_wf() に処理が移る。

3. 実行中のタスクフロー情報を確認するために、diff_wf() はシステム状況把握モジュールの get_record() を呼びだし、タスク実行情報を取得する。

4. 取得したタスク実行情報を使って、diff_wf() の内部で task_abort() を呼びだし、グリッドミドルウェア側のタスク管理ミドルウェアに投入されている すべて のタスクを放棄する。(実行完了したタスクはタスク管理ミドルウェアの情報からは削除されているので、実質的には実行中と実行待ちのタスクがキャンセルされる。)

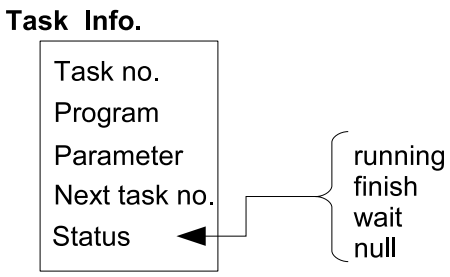


図 4.14　差分タスクフロー実行機構におけるタスク情報のデータ構造

5. make_wf() に処理が移り、差分タスクフロー情報を作成する。差分タスクフロー情報の作成手順を以下に示す。

   (a) シミュレーション実施者が修正したタスクフローデータのタスク情報を図 4.14 に示すデータ構造で保持する。シミュレーション実施者が修正したタスクフローデータは、タスクの実行把握情報は存在しないため、タスクの実行把握情報の欄は空欄にする。

   (b) システム状況把握モジュールが保持しているタスク情報格納データベースからタスクフロー番号にもとづくすべてのタスク情報を取得する。取得したタスク情報は修正したタスクフローデータのタスク情報と同様に、図 4.14 に示すデータ構造で保持する。

   (c) タスク情報格納データベースから取得したタスク情報をもとに実行中のタスク位置を検知する (図 4.15-a)。具体的には、タスク番号の小さい順に図 4.14 に示すデータ構造の status の項目で実行中の状態 (running の値) をもつタスクを探す。

   (d) タスク情報格納データベースから取得したタスク情報とシミュレーション実施者によって投入された修正タスクフローデータとの比較を行う (図 4.15-b)。タスク番号ごとにプログラム名およびパラメータの項目を比較し、修正されたタスク情報のタスク番号を検知する。

   (e) タスク情報格納データベースから取得したタスク情報にシミュレーション実施者が修正したタスク情報を上書きする (図 4.15-c)。

   (f) 実行中のタスク位置の情報 (5c. で得られた情報) と修正されたタスク位置の情報 (5d. で得られた情報) を利用して差分タスクフローデータを作成する (図 4.15-d)。

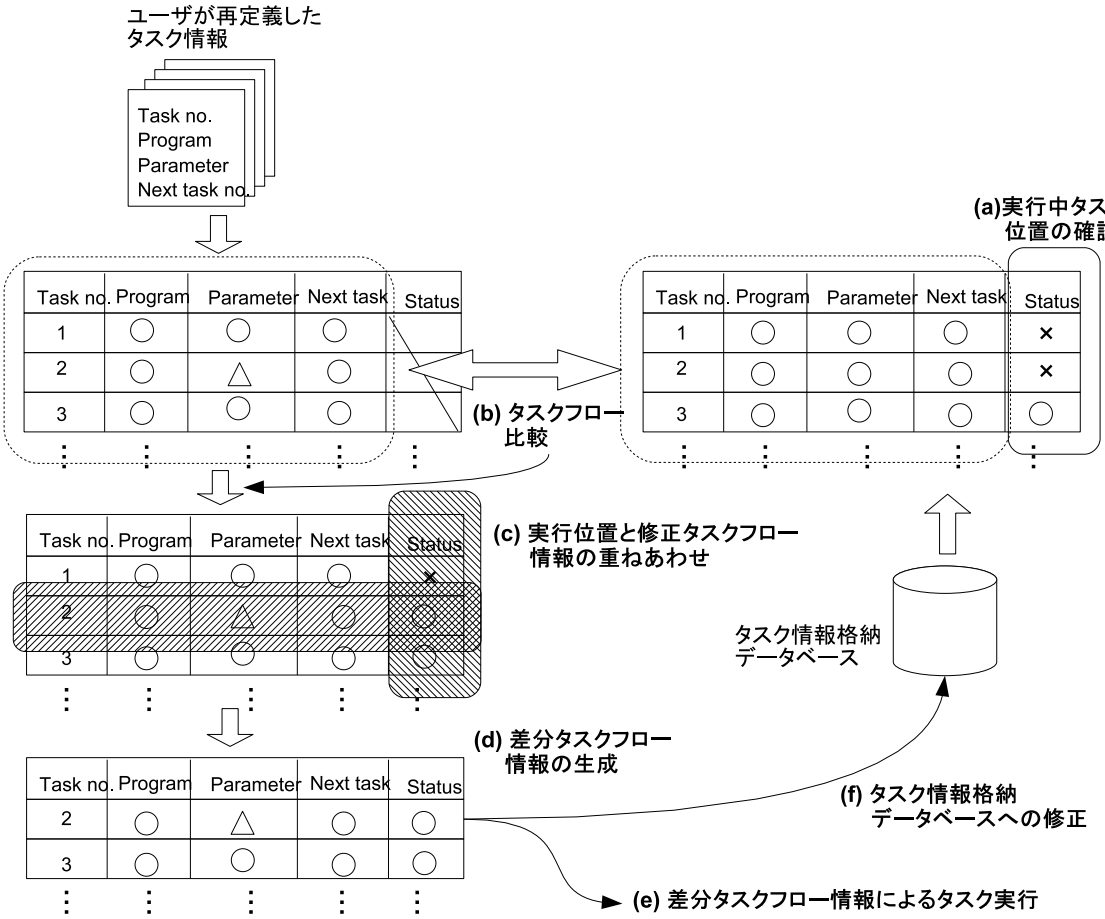


図 4.15 差分タスクフローの作成

i. 実行中のタスク位置が修正されたタスク位置よりも前にある場合、実行中のタスク位置からのタスクフローデータを作成する。

ii. 実行中のタスク位置が修正されたタスク位置よりも後ろにある場合、修正されたタスク位置からのタスクフローデータを作成する。

6. `make_wf()` の内部で `task_dispatch()` を呼びだし、ローカルのタスク管理ミドルウェアに対して差分タスクフロー情報に記述されている すべてのタスク情報 を投入する。

7. `make_wf()` の内部で `modify_id()` を呼びだし、タスク情報格納データベースに登録されているタスク情報を更新する。タスク情報の更新について、差分タスクフローとして出力されたタスク情報の実行状態は次のように取り扱う。先頭のタスク以外のタスクは wait 状態として更新する。先頭のタスクは running 状態として更新する。

タスクフロー差分実行機構の導入により、シミュレーション実施者はタスクの情報修正を施したタスクフロー情報を投入するだけでよい。シミュレーション結果の継続的な改善動作の面で、操作工数が低減される。

4.2.5 大量タスク実行におけるタスク優先実行制御

パラメータリストファイルに基づいた大量タスク実行の場合、記述されている順に実行される。タスクの実行順序が固定化されてしまうため、シミュレーション結果の出力順はリストの記述順に影響を受ける。それゆえに大量タスク実行においては次のような問題が生じる。

- 高スループット計算のうち、データ並列計算やパラメータスウィープ型計算のように全探索が必要な計算の場合、パラメータリストファイルがシーケンシャルに定義されていると、他のパラメータの計算結果を見ることができない。
- 分子シミュレーションの例ではパラメータを変更して複数のシミュレーションを実施し、様子を見ることが頻繁に行われる。これを満たすためには大量タスク実行の放棄とパラメータ変更による再実行の繰り返しを頻繁に行うことになる。このとき、重複したパラメータでタスク実行してしまうので、計算機資源を無駄に利用する。

そこで、既に得られた計算結果を利用したシミュレーション実施者の意思決定に基づくシミュレーション実施を行うために、パラメータリストの順序を変更し、特定のパラメータの組を優先実行する機能をプログラム実行モジュールに追加する。

この機能は、パラメータリストファイルに記述されている順序を変更し、ある特定の値をもつパラメータの組を優先して実行するものである。

パラメータリストファイルの実行順序を変更する機能の動作の流れを具体例を示しながら説明する。具体例として、図 4.17 に示すパラメータリストを用いる。これを初期パラメータリストとする。

1. まず、シミュレーション実施者は随時出力されるシミュレーション結果を確認し、優先実行させたいパラメータの値を決定してもらう。優先実行させたいパラメータの値が決定したら、その値を操作インターフェイスに入力する。

   入力された値は操作インターフェイスによって図 4.19 に示すデータを出力する。優先実行させたいパラメータを示すデータは CSV の書式で、第 1 番目がパラメータの位

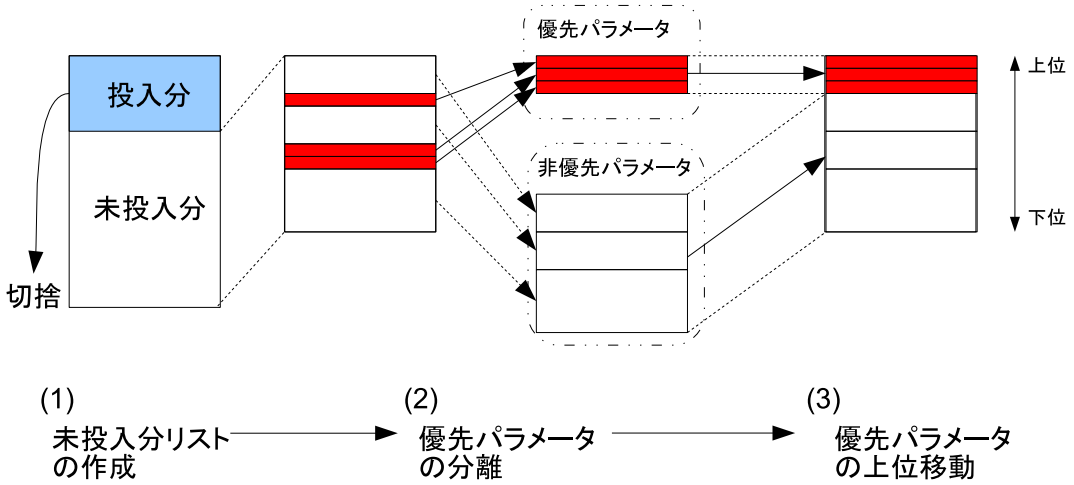


図 4.16 パラメータリストの更新手順

置を、第 2 番目がその値を示す。アスタリスクで記述されているパラメータの位置はどの値でも構わないことを意味する。図 4.19 の例では、7 番目と 8 番目の引数が 4000 の値をもつリストを優先する意味である。

2. プログラム実行モジュールは、優先実行させたいパラメータの値を受けとると、次の操作を行う。

   (a) 実行し終えたパラメータリストを除いたパラメータリストを作成する (図 4.16-(1))。これを差分パラメータリストと呼ぶ。この操作は、同一パラメータリストの重複実行を避けるために行う。

   付録 A.1 に掲載する大量タスク実行用スクリプトでは、submitdat.bat というファイルに実行し終えたパラメータリストが記述される。このファイルに書かれているパラメータをパラメータリストから抜き出す。

   (b) 差分パラメータリストから、優先実行させたいパラメータの値をもつパラメータリストとそうでないパラメータリストと分離する (図 4.16-(2))。前者を優先パラメータリスト、後者を非優先パラメータリストと呼ぶ。

   差分パラメータリストの先頭から順に、優先実行させたいパラメータの値を持っているか比較する。比較した結果、優先パラメータリストであれば、優先パラメータリストファイルとして一時的に出力する。また、非優先パラメータリストであれば、非優先パラメータリストとして一時的に出力する。

```
 1: 1000 1000 -1   -1   2500 3000 2500 2500
 2: 3000 2500 -1   -1   2500 2000 2500 2500
 3: 4000 1000 4000 1000 2500 2000 4000 4000
 4: -1   -1   1000 1000 4000 2500 2500 2500
 5: 1000 1000 -1   -1   1500 4000 2500 2500
 6: -1   -1   4000 4000 1500 4000 4000 1000
 7: 4000 4000 -1   -1   2500 4000 2500 2500
 8: 1000 1000 -1   -1   4000 1000 2500 2500
 9: 4000 4000 -1   -1   2500 4000 4000 1000
10: 1500 4000 1500 4000 2000 2500 4000 4000
11: 4000 4000 4000 4000 2500 4000 4000 4000
12: 2500 2500 -1   -1   1000 1000 4000 4000
13: -1   -1   3000 2500 2500 3000 4000 1000
14: 1500 4000 -1   -1   1000 1000 2500 2500
15: -1   -1   2500 2500 1000 1000 4000 1000
16: -1   -1   3000 2500 2500 4000 4000 1000
17: -1   -1   4000 1000 1000 1000 2000 4000
18: -1   -1   2000 2500 4000 2500 2500 2500
19: 4000 1000 -1   -1   3000 2500 2000 4000
20: 4000 1000 -1   -1   1000 1000 2000 4000
```

図 4.17　優先タスクリスト作成前 (上位 20 行, 左端は行番号)

```
 1: 1500 4000 1500 4000 2000 2500 4000 4000
 2: 4000 4000 4000 4000 2500 4000 4000 4000
 3: 2500 2500 -1   -1   1000 1000 4000 4000
 4: 2000 2500 -1   -1   2500 2000 4000 4000
 5: 1000 1000 -1   -1   3000 2500 4000 4000
 6: 1000 1000 1000 1000 2500 4000 4000 4000
 7: 2500 2500 2500 2500 4000 4000 4000 4000
 8: 1500 4000 -1   -1   3000 2500 4000 4000
 9: 1500 4000 1500 4000 2500 2000 4000 4000
10: 4000 1000 -1   -1   2500 1000 4000 4000
11: 2000 2500 -1   -1   3000 2500 4000 4000
12: 2000 2500 -1   -1   2500 2500 4000 4000
13: 2000 2500 2000 2500 4000 1000 4000 4000
14: 3000 2500 3000 2500 1500 4000 4000 4000
15: 2000 2500 -1   -1   4000 1000 4000 4000
16: 2500 2500 2500 2500 2500 2000 4000 4000
17: 3000 2500 -1   -1   2000 2500 4000 4000
18: 1500 4000 1500 4000 1000 1000 4000 4000
19: 3000 2500 -1   -1   2500 4000 4000 4000
20: 2500 2500 2500 2500 1000 1000 4000 4000
```

図 4.18　優先タスクリスト作成後 (上位 20 行, 左端は行番号)

```
param01, *
param02, *
param03, *
param04, *
param05, *
param06, *
param07, 4000
param08, 4000
```

図 4.19 優先実行するパラメータの情報

(c) 優先パラメータリストファイルに非優先パラメータリストファイルの内容を追記する (図 4.16-(3))。この結果得られたものを修正パラメータリストと呼ぶ。

修正パラメータリストの例を図 4.18 に示す。7 番目と 8 番目の引数が 4000 になっている。また、初期パラメータリストの 10 行目、11 行目、12 行目が修正パラメータリストの 1 行目、2 行目、3 行目になっている。

3. プログラム実行モジュールで作成した修正パラメータリストファイルを用いて、タスク実行を再開する。

タスク実行支援モジュールでは、大量タスク実行スクリプトによるタスク実行を行うので、現在実行中の大量タスク実行スクリプトを放棄する。そして、修正パラメータリストファイルを大量タスク実行スクリプトの第 2 引数に設定し、タスク実行を再開する。

タスク優先実行制御のプログラム実行モジュール内での振る舞いを説明する。タスク実行支援フレームワークでは、大量タスク実行スクリプトを用いて大量タスク実行を 1 つの単一タスクとして取り扱っている。プログラム実行モジュール内での基本的な振る舞いは図 4.20 に示すように、タスクフロー差分実行機構に拡張したかたちになっている。拡張した点を以下に示す。

- タスクフロー差分実行機構では `selector()` に対して修正したタスクフローを直接投入した。一方、タスク優先制御機構では `modify_plist()` から `selector()` を内部で呼び出し、タスクフロー情報が修正されたかのように振る舞う。
- `modify_plist()` では優先するパラメータの位置と値を入力とし、新しいパラメータリストと新しいタスクフロー情報を生成する。

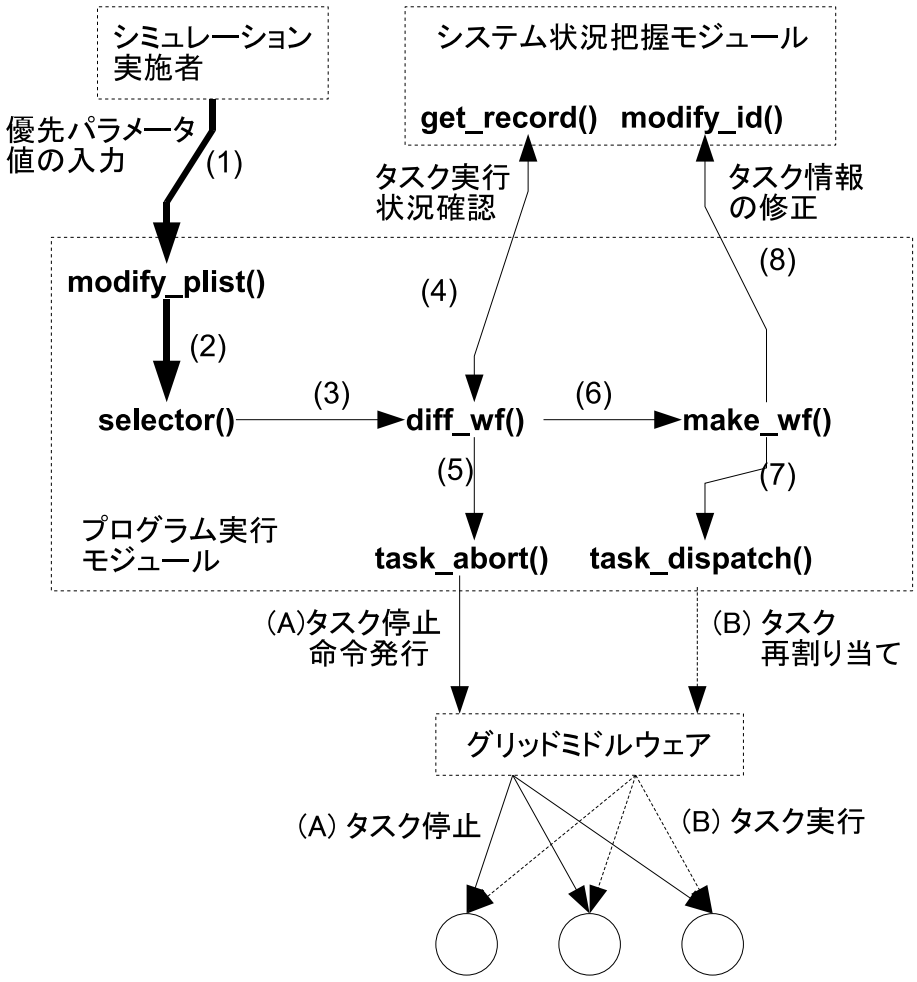


図 4.20　パラメータ優先実行制御のプログラム実行モジュール内での流れ

新しいタスクフロー情報は、大量タスク実行スクリプトの引数であるファイル名を変更する。具体的にはファイル名にバージョン番号を付加する。(例えば、初期のファイル名が paramlist であれば、paramlist-v02 という感じに、異なるファイル名にする。)

この仕組みを導入することで、シミュレーション実施者の継続的な改善により目的関数が最大になるであろうパラメータリストが優先的に実行される。また、既に実行されたパラメータをもつタスクについては、過去の結果を利用するので、パラメータリストを用いた場合のタスク再実行において 1 つのパラメータを重複して実行するといった問題が解決される。

表 4.5 タスク実行支援フレームワーク導入による効果

比較項目	従来	提案
方針	すべてのタスクの実行を完了させること	希望する解が得られるまで改善する動作を行うこと
シミュレーション実施者との対話	一方向 (実施者 → システム)	双方向
パラメータ誤入力への対処	シミュレーション終了後	シミュレーション実施中に対応できること
大量タスクの実行順序	シーケンシャル実行が原則	シーケンシャル実行とランダム実行の併用が必要
大量タスクの停止処理	残りすべてのタスクに対して手動で操作	1 回の操作ですべてのタスクが停止できること

4.3. まとめ

本章では、高スループット計算向けグリッドにおけるタスク実行支援フレームワークについて提案した。

問題解決環境の構築において、ユーザの知識や経験に基づいた操作環境が必要である。本章では、シミュレーション実施者による意思決定動作を分析した。その結果、シミュレーションの実行と結果確認の動作が継続的に行われることが、シミュレーション結果の改善につながることを示した。この分析から、PDCA サイクルにもとづいたタスク実行支援フレームワークを提案した。

タスク実行支援フレームワークはユーザインターフェイス、プログラム実行モジュール、計算結果解析モジュール、システム状況監視モジュールの 4 モジュールで構成されることを述べた。

シミュレーション実施者の操作面での負担を考慮し、タスク実行支援フレームワークでは、タスクフロー差分実行機構および大量タスクにおけるタスク優先実行機構を提案した。タスクフロー差分実行機構は複雑なシミュレーションの実施において、タスクフロー情報の修正位置からタスク実行を可能にする。また、タスク優先実行機構は既に実行されたパラメータを除外するとともに、パラメータリストの中から優先するパラメータの実行順位を上位にする。本章では、タスクフロー差分実行機構および大量タスクにおけるタスク優先実行機構の振る舞いについて説明した。

タスク実行支援フレームワークを導入することによる定性的な効果を表 4.5 に示す。タ

スク実行支援フレームワークを導入することで、シミュレーションの途中で結果を確認しながらタスクのパラメータ修正と実行への反映が可能になる。また、分子シミュレーションなどの例にみられる、パラメータを変更して複数のシミュレーションを実施して出力される結果の様子を見ることに対しても柔軟に対応できる。

第5章

高スループット計算におけるデスクトップ計算機の利用

5.1. デスクトップグリッド

5.1.1 デスクトップグリッドとは

高スループット計算においてシミュレーション時間の短縮を実現するためには、計算機の台数を確保することが必要になる。一般的には、Globus Toolkit といったミドルウェアを用いて複数拠点の PC クラスタを確保することが行われるが、拠点ごとに管理ポリシが異なるため、複数拠点の計算機を常に利用できるとは限らない。

一方で、事務処理で用いられている計算機、大学などの演習室に設置されている計算機は必要に応じて使われているものの、1 日という期間で考えたとき、各計算機は常に稼働している状態とは限らない。平日夜間、休日は利用者がいないこと、平日昼間でも昼食時や会議などで端末の前から離れた場合、授業の合間などは遊休状態である。

このように、デスクトップ計算機の状態に着目し、余剰となっている計算処理能力をシミュレーションなどに用いる枠組みとしてデスクトップグリッドがある。特に、費用面で PC クラスタの導入が難しい企業、計算機資源の有効利用に注目している企業や大学および高専でデスクトップグリッドが積極的に取り入れられている。

5.1.2 デスクトップグリッドと高スループット計算

デスクトップグリッドの構成要素であるノードは、企業であれば事務処理用の計算機が大学であれば演習室や研究室で利用されている計算機が対象となる。PC クラスタと同様、デスクトップグリッドでもコモディティな計算機を利用するが、

- すべてのノードが同一性能であるとは限らないこと、
- ノードには所有者が存在すること、

が PC クラスタとの相違点である。デスクトップグリッド環境全体として考えたとき、各計算機の所有者の振舞いが特定できないため、シミュレーションに利用できるノード数が時々刻々と変化する。そのため、シミュレーション全体としてスループットを一定にすることは難しいところが PC クラスタとは異なる。

5.1.3 デスクトップグリッドにおけるタスク割り当て方針決定の難しさ

デスクトップグリッドでは各ノードのハードウェア制約と著しい状態変化があるため、PC クラスタを用いたグリッドと異なり、タスク割り当て方針を決定することが容易ではない。以下でタスク割り当て方針決定の難しさについて論じる。

Greedy なタスク実行の限界

一般的に、高スループット計算では大量のタスク実行が行われる。1 つのプログラムに対して大量のパラメータを準備し、各ノードでパラメータを実行する形態である。大量のタスク実行を行う場合、PC クラスタで広く採用されている Greedy なタスク実行を考えることができる。Greedy なタスク実行では 1 台でも多くのノードを利用することが可能であるが、各ノードによって処理能力が異なる、また、ノード利用可能状況が時々刻々と変化するデスクトップグリッドでは処理能力の高いノードに割り当てられるか、もしくは処理能力の低いノードに割り当てられるかは予測できない。シミュレーション実施者はシミュレーション全体のスループットを高めたいという要求を常にもっているが、遊休状態だからといって Greedy なタスクを割り当てを選択すると却ってスループット低下を招く。

タスク制御にかかるコスト

デスクトップグリッドを構成するすべてのノードには所有者が存在する。デスクトップグリッドでは各ノードの遊休状態を用いることが原則であるため、各ノードでのタスク実行における優先順位は、

1. 所有者によって実行されるタスク (以後、所有者タスクと略す)
2. デスクトップグリッド側から依頼されるタスク (以後、グリッドタスクと略す)

になる。デスクトップグリッド構築ミドルウェアで有名な Condor と BOINC はこの優先順位に基づいてタスク実行が行われている。

ここで、ノードの状態が遊休状態から解除された場合を考える。所有者が計算機利用を再開した場合である。

Condor では独自にタスクのサスペンド機能、タスクマイグレーション機能を提供しており、グリッドタスクの制御はこれらの機能を使い分けている。

具体的には、遊休状態が解除された時点で実行中だったグリッドタスクはサスペンド機能が働き、一時停止状態になる。数分経ってもグリッドタスクのサスペンド機能が解除されない場合は、タスクマイグレーション機能が働き、Condor Manager の指示によって別なノードで実行を再開する。

但し、すべてのタスクにサスペンド機能やタスクマイグレーション機能が有効ではない。Condor が提供する condor_compile のコマンドで作成されたタスクのみに限られる。したがって、タスク間でメッセージパッシングを行うプログラムでは通信が途切れた場合の保証ができないためこれらの機能は使えない。タスクの実行が継続されるという意味でこれらの機能は有用であるが、タスクマイグレーション機能が作動するとタスク移動の時間的コスト、再割当先の決定にかかるコストが発生することに注意しなければならない。また、タスクマイグレーション機能自体を有効にすることで、タスク本来の処理内容にマイグレーションに関係する機能 (チェックポインティング機能による演算状態の断片記憶) が追加されるため、タスクを純粋に実行するよりも遅くなる問題がある。特に、Condor の場合はチェックポインティング機能がアプリケーションレベルでの実装であるため、チェックポインティング機能にかかるコストは無視できない。

一方、BOINC ではノードは決められた処理をこなすだけなので、Condor のような機能は提供していない。サーバ側が高スループット計算全体のスループット維持のために各ノードの処理状態を考慮してタスク投入量の制御を行うこと、結果の正しさを確認するために

複数のノードに同一の問題を解かせる工夫を施している。このため、BOINC では解くべき問題が限られる。

まとめ

以上のように、デスクトップグリッドではノードの所有者の振る舞いに左右されることから、タスク割り当て方針の決定が難しい。

まず、Greedy なタスク実行はデスクトップグリッド向きではないことがいえる。

次に、Condor のようにタスクマイグレーション機能を有効にして、タスク実行の確実さを選択するかについては、長時間のタスク実行であればタスクマイグレーション機能のコストは小さいが、短時間で終了するようなタスクの場合にはタスクマイグレーション機能を有効にするコストが大きくなり、タスクマイグレーション機能が適切かは疑問である。

5.2. 負荷情報に基づいたノード選抜機構の提案

デスクトップグリッドにおけるタスク割り当てでは、グリッドを構成するノードのなかから所有者が利用していないノードを探すことが必要になる。そこで、各ノードにおける過去の負荷状態の情報を利用することで、遊休状態が確実とされるノードを選抜する方法を提案する。

5.2.1 ノード選抜機構の構成

本論文で提案するノード選抜機構は、プログラムを実行する計算機 (Node)、プログラムを割り当てるノードであるタスク割り当てサーバ (Job Allocator)、タスク割り当てサーバの依頼を受けノード選抜を行うノード選抜サーバ (Node Selection Server) の 3 つで構成される (図 5.1)。以下では、ノード選抜サーバを構成する機能であるノード情報収集モジュール、ノード情報蓄積モジュール、ノード選抜モジュールについて説明する。

- ノード情報収集モジュール
  ノード情報収集モジュールは、デスクトップグリッドを構成する各ノードのプロセッサの状態およびネットワークの状態を収集する。ここで、プロセッサおよびネットワークの状態は負荷という形で示す。本システムにおけるプロセッサ負荷は load average を利用する。また、ネットワーク負荷はノードに対して入出力されたトラフィック量

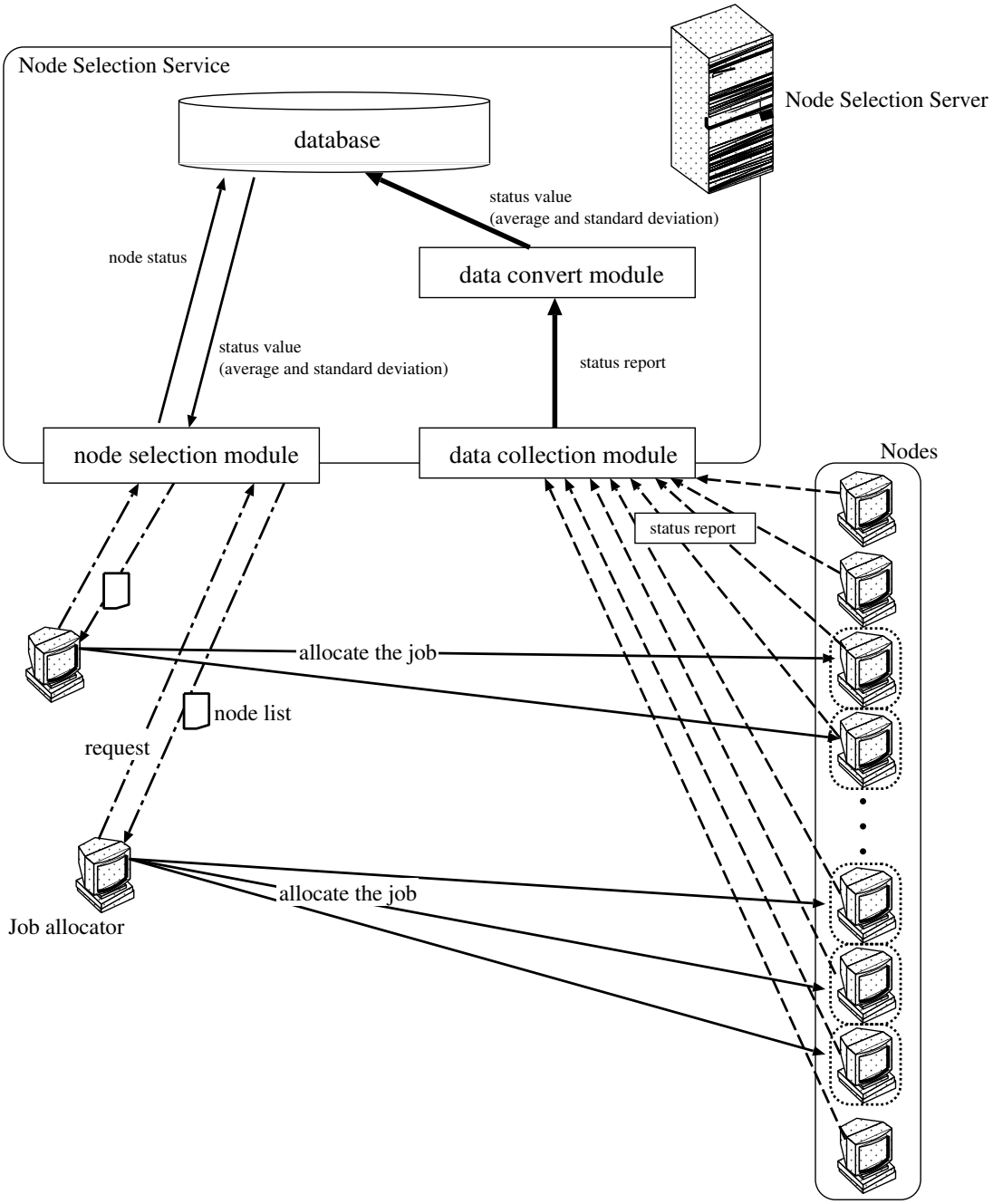


図 5.1 ノード選抜機構の構成

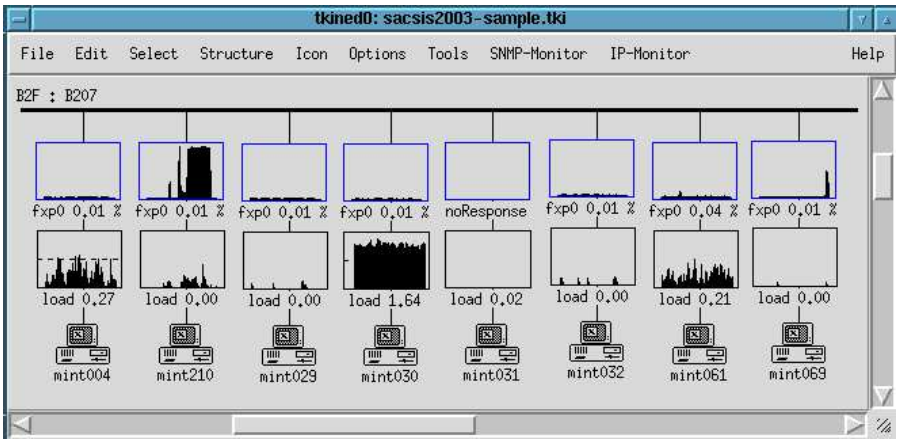


図 5.2　キャンパスにある計算機の振舞い

を利用する。ノード情報収集モジュールは各ノードに対して一定の時間間隔で問い合わせを行い、プロセッサ負荷およびネットワーク負荷の値を取得する。一定の時間間隔で各ノードに対して負荷情報を問い合わせ、ステータスチェックの役目を兼ねる。したがって、データが取得できないノードについては、何らかの障害が発生したものとみなす。

- ノード情報蓄積モジュール
  ノード情報蓄積モジュールは各ノードの振舞いについて統計処理を行う。具体的には収集した負荷情報をもとに 1 時間毎の平均と標準偏差を出力する。負荷の平均および標準偏差は過去の履歴として蓄積する。

- ノード選抜モジュール
  ノード選抜モジュールは、Job Allocator のタスク投入のリクエストを受けると、蓄積された各ノードの負荷変動の値を用いてタスク投入時に適切なノードを選抜する。そして、選抜したノードのリストをユーザに通知する。

5.2.2　負荷の短期傾向を利用したノード選抜

ノード負荷の分類

デスクトップグリッドにおけるノードの負荷は各ノードの所有者が利用するタイミングに影響を受ける。図 5.2 はある日の昼間における奈良先端科学技術大学院大学 情報科学研

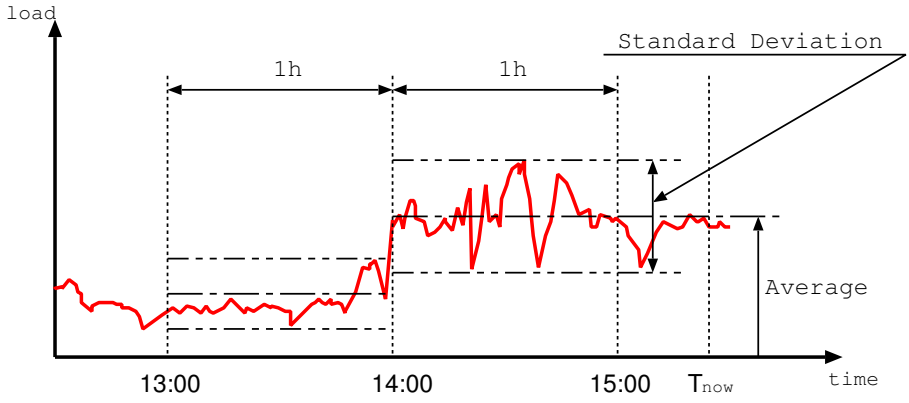


図 5.3 計算ノードにおける負荷の振舞いと負荷変動との関係

究科内に設置されている計算機のプロセッサ負荷とネットワーク負荷を Scotty と呼ばれるツールで監視し、描画したものである。図中各計算機の上部のグラフがネットワーク負荷を、下部のグラフがプロセッサ負荷を示している。負荷の高い計算機と低い計算機がいり交じった状態であることが図 5.2 から読みとれる。

一般的に、会社や研究室に設置されているデスクトップコンピュータは原則として執務時間に負荷が発生すると考えられる。また、大学の演習室にある演習用端末は演習のあるときに限り負荷が発生する。これらの計算機は 1 日という単位で考えたとき、常に高負荷であるとは限らない。夜間や休日は利用されていないため、ほぼ無負荷である。このことから、ノードの負荷は時間の経過と関係がある。

そこで、ノードの時間的経過による負荷に着目し、負荷変動というパラメータを定義する。負荷変動は、以下の 2 種類に分類する。

- 定常的な事象による負荷変動
  長時間実行されるプログラムが及ぼす負荷。シミュレーションを行うためのプログラムのように、プロセッサ能力を占有するプログラムによって生じるもの。

- 突発的な事象による負荷変動
  計算機所有者の操作によって発生する負荷。Web ブラウザ、ワードプロセッサなど、対話的な操作を必要とするアプリケーションによって生じるもの。

これら 2 つの負荷変動を定量的に示すためのパラメータとして、それぞれ 1 時間毎の負荷の平均および標準偏差を用いる (図 5.3)。

負荷の短期傾向を利用したノード選抜の方針

デスクトップグリッドとして存在するノードの中からプログラム処理時間の短縮を図るために適切とされるノードの選抜を行うためにはなるべく負荷の小さいノードを選抜することが望ましい。また、プログラム処理時間のゆらぎを小さくするために適切とされるノードの選抜を行うためには、なるべく負荷の変動が小さいノードを選抜することが望ましい。そこで、デスクトップグリッドとして登録されているノードの中から以下の方針に従ってノード選抜を行う。

- 過去の負荷変動に対して、以下の方針でノードを選抜する。
  1. 負荷の平均が小さいノードを、
  2. 負荷の平均が同じノードに対しては負荷の標準偏差が小さいノードを優先する。
- 過去の負荷変動の履歴に対して、過去の負荷変動が小さいものを優先する
- 現在の負荷に対して、負荷の値が小さいものを優先する

短期傾向を利用したノード選抜アルゴリズム

負荷の短期傾向を利用したノード選抜は以下の流れで行われる。

1. 予めノード選抜システムに登録されているノードの数と同じ大きさのテーブル X を準備する。テーブルの各要素とノード選抜システムに登録されているノードリストとは 1 対 1 対応が取られているものとする。また、テーブル X のすべての要素は 0 で初期化されているものとする。
2. ノード選抜システムに問い合わせたいノードの数 m を定義する。
3. 現時点における各ノードのプロセッサ負荷およびネットワーク負荷の低い順に m 台選び出す。選び出されたノードに該当するテーブル X の要素に 1 を加える。
4. 1 時間前の各ノードのプロセッサ負荷の平均の低い順に m 台選び出す。選び出されたノードに該当するテーブル X の要素に 1 を加える。
5. 同様に、1 時間前の各ノードのネットワーク負荷の平均の低い順に m 台選び出す。選び出されたノードに該当するテーブル X の要素に 1 を加える。

表 5.1 計算ノードの諸元

CPU	Pentium III 800MHz
Memory	512MByte
OS	FreeBSD 4.2-RELEASE
Network	100Base-TX Ethernet
Compiler	gcc version 2.95.2
MPI	MPICH version 1.2.2.3

6. 1 時間前の各ノードのプロセッサ負荷の平均と標準偏差の和をとり、その値の低い順に m 台選び出す。選び出されたノードに該当するテーブル X の要素に 1 を加える。

7. 同様に、1 時間前の各ノードのネットワーク負荷の平均と標準偏差の和をとり、その値の低い順に m 台選び出す。選び出されたノードに該当するテーブル X の要素に 1 を加える。

8. (4) から (7) と同じ処理を n 時間前のデータに対しても同様に行う。(但し、n はノード選抜システム側で予め設定された値とする。)

9. (3) から (8) の処理で得られたテーブル X の各要素の値の高い順に m 台選び出す。この結果から選び出されたノードが本システムを用いて選出されたノードとなる。

5.3. 評価

提案するノード選抜機構によって適切にノードが選抜されているかについて評価する。

5.3.1 実験環境

奈良先端科学技術大学院大学の情報環境基盤である曼陀羅システムに接続されている計算機を用いた。曼陀羅システムには様々な計算機等が接続されているが、実験では情報科学研究科棟内に配置されている個人常用ワークステーションに絞った。計算ノードの諸元を表 5.1 に示す。性能評価で使用した計算機は 88 台である。

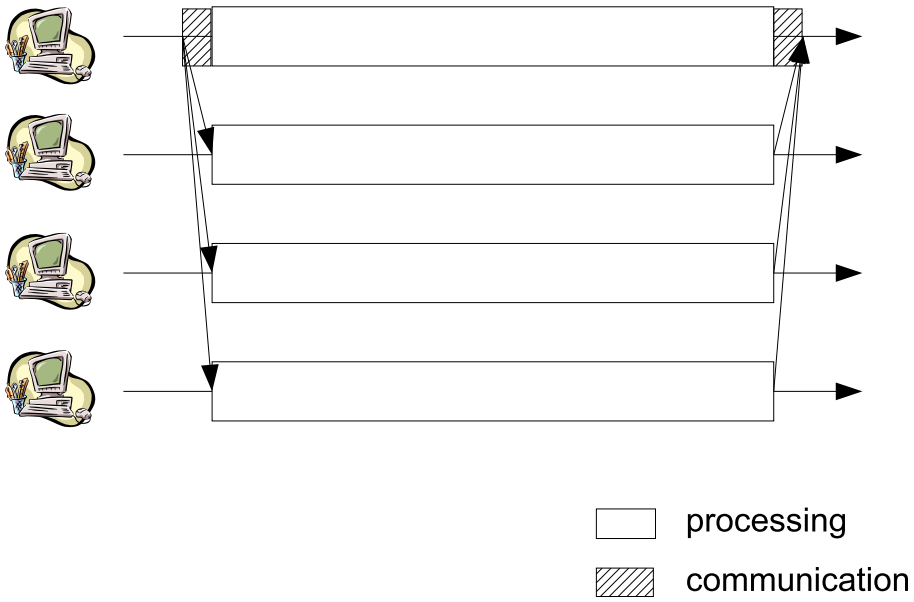


図 5.4　NPB-EP ベンチマークの動作フロー

5.3.2　実験方法

ノード選抜機構で適切にノードが選抜できたこと示すべく、本論文では、並列処理用のベンチマークツール用いて選抜したノードでのスループットを測定した。

ベンチマークツールについては、並列計算機の実行性能を知る上で最も使われている NAS Parallel Benchmarks[1]　(NPB) version 2.3 の中から EP を用いた。問題サイズは CLASS A を用いた。

NPB-EP ベンチマークは典型的なモンテカルロシミュレーションを行うことを想定したベンチマークである。プログラムはほぼ完全な並列化がされているため、計算機間でのデータ転送はほとんどない (図 5.4)。このベンチマークがもつ処理フローのイメージとしてはモンテカルロシミュレーションのほか、データ並列アプリケーション、パラメータスイープアプリケーションであり、高スループット計算向けのアプリケーションとして考えることができる。

ベンチマークツール実行時に選抜されるノードの数は 4、8、16、32 台とし、提案するノード選抜アルゴリズムを用いてノードを選抜した場合、およびランダムにノードを選抜した場合の NPB-EP ベンチマークから出力されるスコアの平均値で評価を行う。スコアは 1 秒間あたりにプロセッサがどれだけ実行されたかを表す Operations per second (op/s) で

[1] http://www.nas.nasa.gov/Software/NPB/

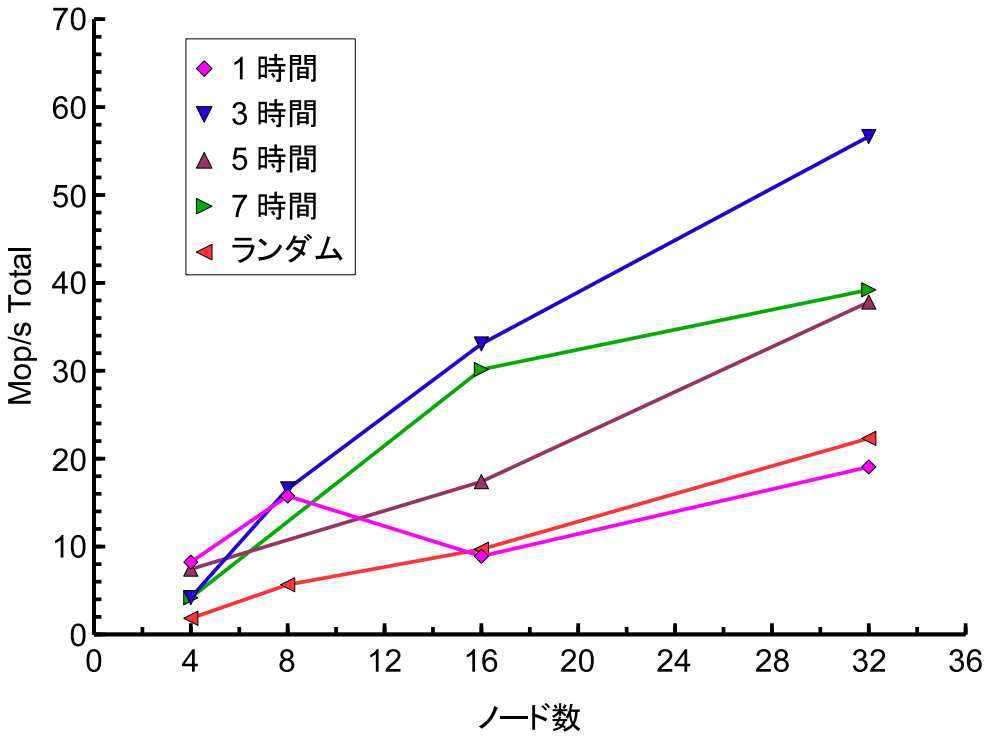


図 5.5 NPB-EP によるベンチマーク結果

出力される。なお、ノード選抜アルゴリズムを用いた場合の過去の履歴については過去 1 時間、過去 3 時間、過去 5 時間、過去 7 時間の 4 種類を用いる。但し、過去 5 時間の履歴、および過去 7 時間の履歴を用いて 8 台のノードを選抜したときのベンチマークツールによる結果については、実験時におけるパラメータ設定のミスにより結果が得られなかったので除外する。

5.3.3 実験結果

図 5.5 は 2003 年 1 月 30 日 16:00 から開始した実験における NPB-EP のランダムにノードを選抜した場合および、提案するノード選抜アルゴリズムを用いてノードを選抜した場合のベンチマーク結果を示したものである。

NPB-EP ベンチマークでは計算処理のほとんどが各ノードで行われ、通信によるオーバヘッドが少ないため、負荷の小さいノードを選抜することで処理能力を向上させることが可能である。実験結果からランダムにノード選抜したときに比べて、最大で 4.5 倍 (過去 1

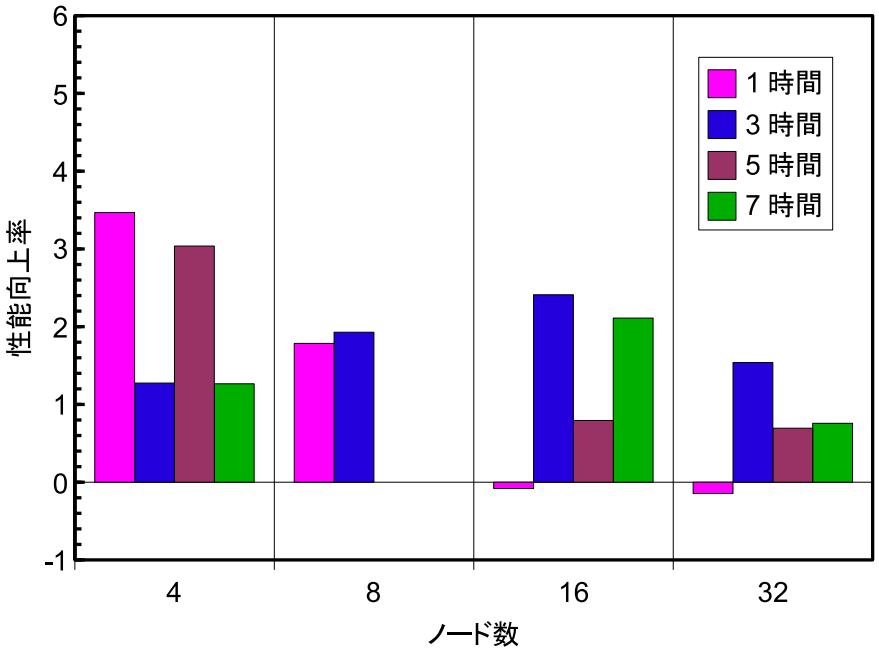


図 5.6　NPB-EP による性能向上率

時間分利用で 4 ノードを選抜した場合) の結果改善が可能になった。

5.3.4　選抜するノード数と負荷履歴との関係

図 5.6 は NPB-EP においてランダムにノードを選抜した場合のベンチマーク結果を基準としたときの、提案するノード選抜アルゴリズムを用いてノードを選抜した場合の性能向上率を示したものである。ここで、性能向上率を定義する。性能向上率 $P_{speedup}$ はノード選抜アルゴリズムを利用した場合の結果を $S_{selection}$、ランダムにノードを選抜した場合の結果を S_{random} としたとき、以下の式で表す。

$$P_{speedup} = \frac{S_{selection} - S_{random}}{S_{random}} \tag{5.1}$$

負荷の短期傾向を利用したノード選抜では、過去の負荷履歴からプログラムを実行する時点において最も遊休状態であろうノードを選抜する。したがって、グリッドを構成するノードの数に比べてプログラムの実行に必要となるノード数が多くなるほど遊休状態でないノードを選抜する可能性が高くなる。

NPB-EP ベンチマークにおいては、過去 1 時間分の結果を利用して 16 ノードおよび 32

ノードを選抜した場合において、性能向上が達成できなかった。ノードの数が増えるにつれて性能向上率が低下していることが読みとれる。

なお、過去1時間分の結果を利用した場合の性能低下に関係する内容として、Derrick らは実証実験によるアプローチからデスクトップグリッドの特徴について報告している [53]。Derrick らの調査によると、San Diego Supercomputer Center における Entropia's DCGrid において計算ノードに投入したプログラムがサスペンドしている時間は平日においては平均 2.8 時間であると報告している。デスクトップグリッドによってプログラムのサスペンド時間は異なるものの、サスペンド時間から必要最低限の負荷履歴を定義することで適切なノード選抜が可能になると考える。

5.4. まとめ

本章では、デスクトップグリッドを構成する計算機の負荷を考慮に入れたノード選抜機構について提案した。

デスクトップグリッドのノードには所有者が存在するため、所有者の利用しない時間帯、いわゆる遊休状態を利用することが必要になる。この点を考慮に入れたタスク割り当て方針決定について論じた。その結果、PC クラスタで用いられている Greedy なタスク割り当てではデスクトップグリッドのタスク割り当てに適していないことを示した。また、Condor のようにタスク実行支援策を盛り込んだものについて、1 タスクあたりの実行時間が短時間なものについては支援策のコストがかかってしまうため適さないことを示した。

本論文では、各計算ノードが生じる負荷は計算ノードの所有者に影響を受けている点に着目し、負荷の傾向を利用したノード選抜機構を提案した。提案するノード選抜機構によって適切なノードが選択されたかを確認するためにベンチマークツールを用いて評価を行った。その結果、概ねシミュレーション時間の改善が可能であることを示した。

第6章

タンパク質立体構造予測シミュレーションへの適用

6.1. タンパク質立体構造予測シミュレーション

6.1.1 タンパク質立体構造予測とは

タンパク質は約20種類のアミノ酸がペプチド結合によってつながった高分子であり、三次元立体構造に折りたたまって生化学的機能を発現する。タンパク質がはたらくためには、通常タンパク質ごとに決まった形を保持することが必須である。生物種のゲノム、全DNA塩基配列が明らかになった今日、その塩基配列から本当に有益な情報を引き出すためには、その配列がコードしているタンパク質の立体構造、機能を知ることが不可欠である。しかし、アミノ酸配列が与えられたときに、そのタンパク質の立体構造を予測することは非常に難しい。

タンパク質立体構造予測とは、X線解析などにより立体構造が同定されていないアミノ酸配列を用いて、その配列がとりうる立体構造をモデリングすること (図6.1) であり、タンパク質立体構造構築原理や機能解析の理解に極めて重要なアプローチである [54]。

立体構造未知のアミノ酸配列 (以後、予測配列と呼ぶ) から立体構造を予測する方法は以下に分類される (図6.2)。

- 比較モデリング
  アミノ酸配列の配列類似度が30%以上の2つのタンパク質は進化的に相同な近縁関係にあり、類似の立体構造をとる。予測配列と近縁で立体構造既知のタンパク質を検

図 6.1　タンパク質立体構造予測

索し、その結果に対して立体構造テンプレートに予測配列をはめ込むことができる。このような方法を比較モデリング (**Comparative Modeling, CM**) と呼ぶ。

- フォールド認識
  立体構造テンプレートが CM の配列相同検索によって発見できない場合、フォールド認識 (**Fold Recognition, FR**) 法を用いて遠縁の構造既知タンパク質を検出し、CM を行う。

- 新規フォールド
  近縁および遠縁の立体構造テンプレートが存在しない予測配列に対して、第一原理に基づき物理化学的エネルギー関数による立体構造を予測する方法 (de novo 予測法) を新規フォールド (**New Fold, NF**) と呼ぶ。

1 つの予測配列が複数のドメインから構成されている場合がある。一般的に、このような予測配列はドメインに分割されたあと、それぞれのドメインに適した予測手法がとられる。本論文では配列相同検索と FR によってテンプレートに基づく立体構造予測が可能な配列をそれぞれ CM ターゲット、FR ターゲットと呼び、それ以外を NF ターゲットと呼ぶ。

6.1.2　全自動立体構造予測システム ROKKY

ROKKY[1] [55] は、CM ターゲット、FR ターゲットおよび NF ターゲットを自動的に判別してタンパク質の立体構造を予測するシステムである。このシステムは国際的立体構造予測コンテスト CASP6(6th Critical Assessment of Techniques for Protein Structure Prediction)

[1] http://www.proteinsilico.org/rokky/

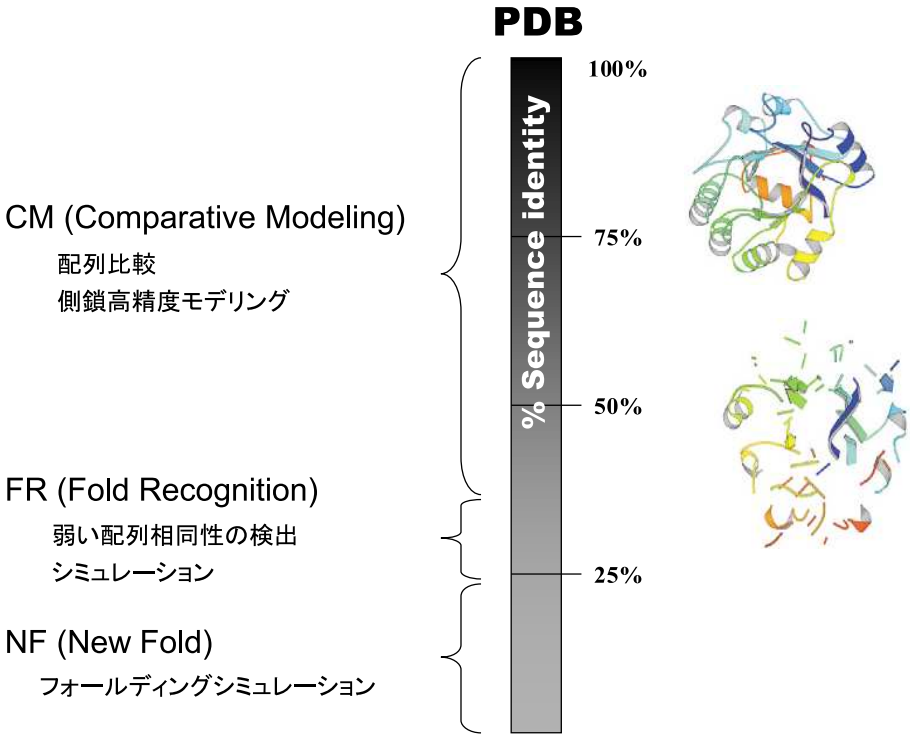


図 6.2 ターゲットの分類

において FR および NF ターゲットの予測精度がサーバとして世界 2 位の実力をもつ高性能な全自動予測システムである。

ROKKY は、PSI-BLAST[56] と FUGUE[57] によるテンプレート検索のツール群と de novo 予測の Fragment Assembly Simulated Annealing(FASA) を用いた SimFold[58, 59] から構成されている。ROKKY における処理の流れを図 6.3 に示す。ROKKY の立体構造予測は PSI-BLAST による予測配列の相同検索を行うことから始まる。統計的有意な近縁のテンプレートが検出できなかった場合は FUGUE によるフォールド認識を行って遠縁のテンプレートを検索する。見つかった近縁・遠縁のテンプレートを用いて比較モデリングを行う。テンプレートが存在しない短い領域に対しては構造既知データベースを用いたループモデリングを行う。テンプレートが存在しない予測配列は NF ターゲットとして FASA を行う。まず、テンプレート検索結果を用いてドメイン分割を行う。そして、それぞれのドメインに対して FASA を複数回実行し、各試行の予測結果をクラスタ分析することによって最終的な予測構造を構築する。

ROKKY によるタンパク質立体構造予測では、NF ターゲットでドメイン分割を終えた

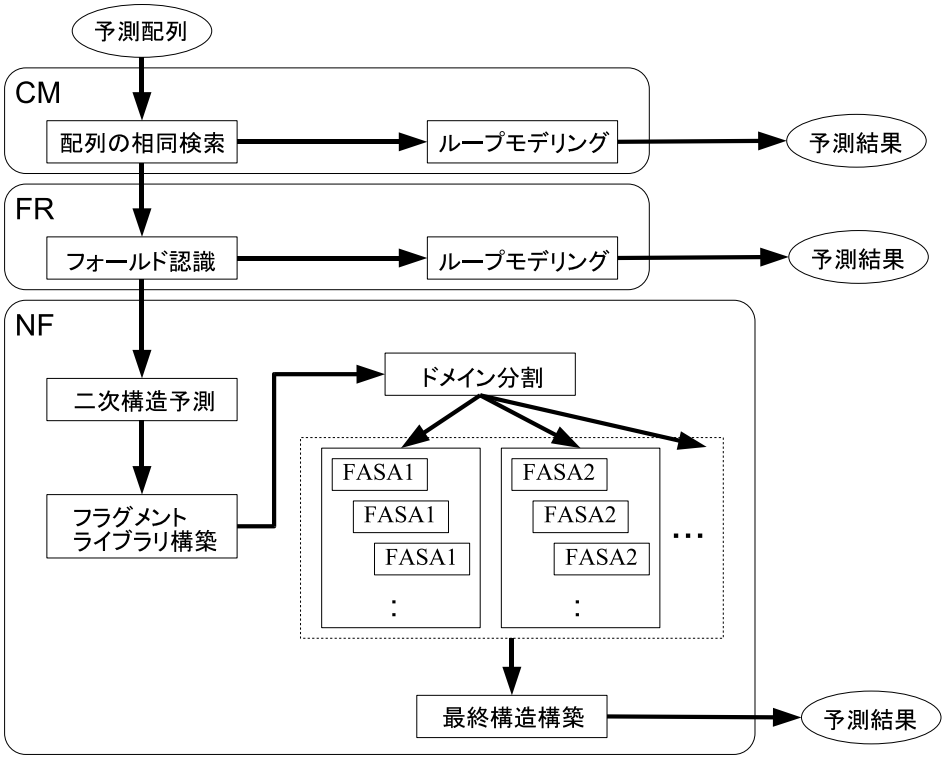


図 6.3　ROKKY における処理の概略

あとの FASA の処理は大量のパラメータセットを準備してプログラムを実行する。この部分が高スループット計算による処理部分である。ターゲットが複雑になるほど、大量のシミュレーションを実行して結果を得ることになる。

6.1.3　タンパク質立体構造予測における対話的動作の必要性

現在、タンパク質立体構造予測手法を確立するための研究が盛んに行われている。立体構造の予測手法を確立することは人間が判断することと同じことができる意味を含んでいる。研究者は予測手法の確立に向けて、より天然構造に似た結果を得るための方法を模索をしている。

図 6.4 は CASP6 において出題された予測配列のうち 21 個の予測配列について、ROKKY を使って得た予測結果 (全自動による予測) と研究者が計算結果を見ながら手動で操作して得た予測結果 (手動による予測) との関係を示したものである。前者は予測配列に対してプログラムを単純に実行して得た結果であり、後者はプログラムの実行結果を見ながら研究

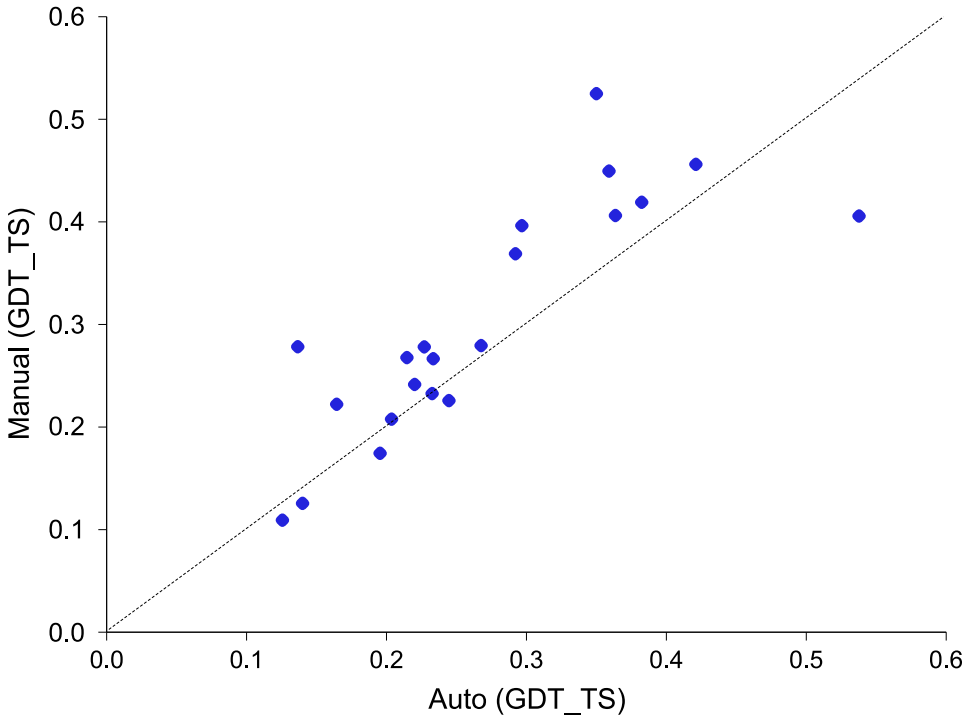


図 6.4 全自動予測結果と手動予測結果の関係

者らがもっている知識や経験をもとにプログラムに与えるパラメータを修正して得た結果である。横軸と縦軸はそれぞれ全自動予測結果と手動予測結果の GDT_TS(Global Distance Test Total Score)[60] である。GDT_TS は誤差 xÅ(x $= 1.0, 2.0, 4.0, 8.0$) 以下で天然立体構造と重ねられる予測立体構造の残基の割合 GDT_P_x によって下式のように定義される。

$$GDT\_TS = \frac{GDT\_P_1 + GDT\_P_2 + GDT\_P_4 + GDT\_P_8}{4} \tag{6.1}$$

図 6.4 では 7 割の予測配列が破線よりも上側に分布している。この結果から、予測配列の多くは依然として手動による予測が有効である。

しかし、シミュレーションを常に手動で行うのは研究者に大きな負担がかかる。自動化できる部分は自動化し、必要に応じてシミュレーション実施者の操作を必要とするシミュレーション環境が望ましい。タンパク質立体構造予測においてより天然構造に近い結果を得るためには、継続的な改善に基づいてパラメータの修正を行うことに専念することが必要である。

6.1.4 タンパク質立体構造予測における継続的改善操作の困難さ

タンパク質立体構造予測では、複数の解析プログラムを用いて天然構造に近い立体構造を導く。このような場合、既存のタスクフローシステムを用いることで、シミュレーション実施者の意思に基づく複数プログラムの実行順序の定義が可能になる。

しかしながら、既存のタスクフローシステムではタスクフローに基づいて実行するのみであり、シミュレーション途中でのタスクフロー情報の変更、タスクに与えるパラメータの変更ができない。すなわち、シミュレーション実施者によるより天然構造に近い立体構造を導くための継続的改善操作ができない。

より天然構造に近い立体構造を導くためには、シミュレーション実施者の意思決定に基づいた各解析プログラムのパラメータの変更を可能にするタスクフローシステムが必要である。

6.2. タスク実行支援フレームワークに基づいたタンパク質立体構造予測システム

タンパク質立体構造予測における目標は立体構造を同定することである。タンパク質立体構造予測に関わる研究者は継続的な改善によって立体構造の同定をおこなっている。また、継続的な改善によって得られた知見はタンパク質立体構造予測の自動化への有益な情報となる。この方針を踏まえ、タスク実行支援フレームワークに基づいたタンパク質立体構造予測システムを構築した。

6.2.1 タスク実行支援フレームワークへの適用とシミュレーション実施者への効果

タスクフローによるシミュレーション実行制御

タンパク質立体構造予測システムにおける具体的目標は、アミノ酸配列からふさわしいと思われる立体構造を検索することである。予測システムに対して、入力される主のデータはアミノ酸配列である。そして、出力されるデータは立体構造の結果である。

ROKKY によるタンパク質立体構造予測では、図 6.3 に示されるように、CM、FR、NF といった 3 種類の解析手法を用いているが、これらは逐次実行される。具体的には、図 6.5 に示すタンパク質立体構造予測用プログラムが用いられている。これらのプログラムを逐

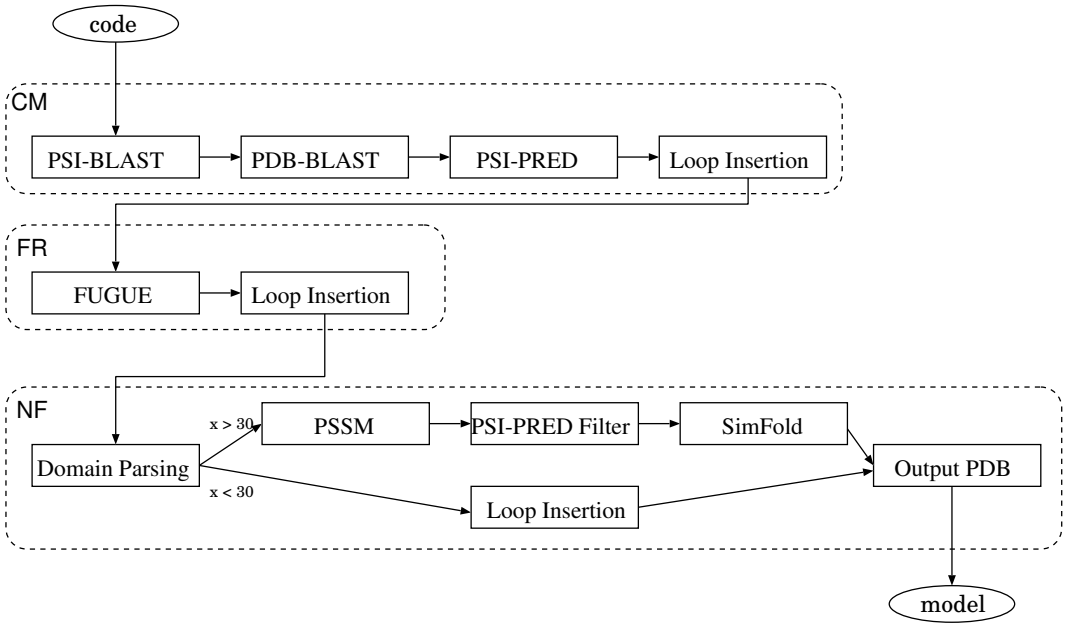


図 6.5 シミュレーションで用いるタスクフロー

次実行するために、ROKKY では 1 つの Perl スクリプトで記述していた。このため、パラメータはシミュレーション実施者の知識と経験により固定されており、パラメータ変更ができない。また、改善的動作のために、ひとつ前のプログラムから再実行することができない。

そこで、タスク実行支援フレームワークを適用するために以下の修正を施した。

- 図 6.5 に示したように、CM、FR、NF それぞれに所属するプログラムを選別した。そして、Perl スクリプトを用いて、CM タスク、FR タスク、NF タスクを実行するスクリプトを作成した。これにより、例えば NF タスクのみの実行が可能になる。
- 各タスクのスクリプトはパラメータ入力を可能にした。これにより、パラメータ変更を可能にした。
- CM タスク、FR タスク、NF タスクの順序で実行するためのタスクフローをタスクフロー情報のフォーマットに基づいて記述した。実際は、図 6.6 に示すタスクフロー記述支援ツールを用いてタスク情報とタスク間の依存関係を入力する。そして、タスクフロー記述支援ツールからタスクフロー情報を出力する。

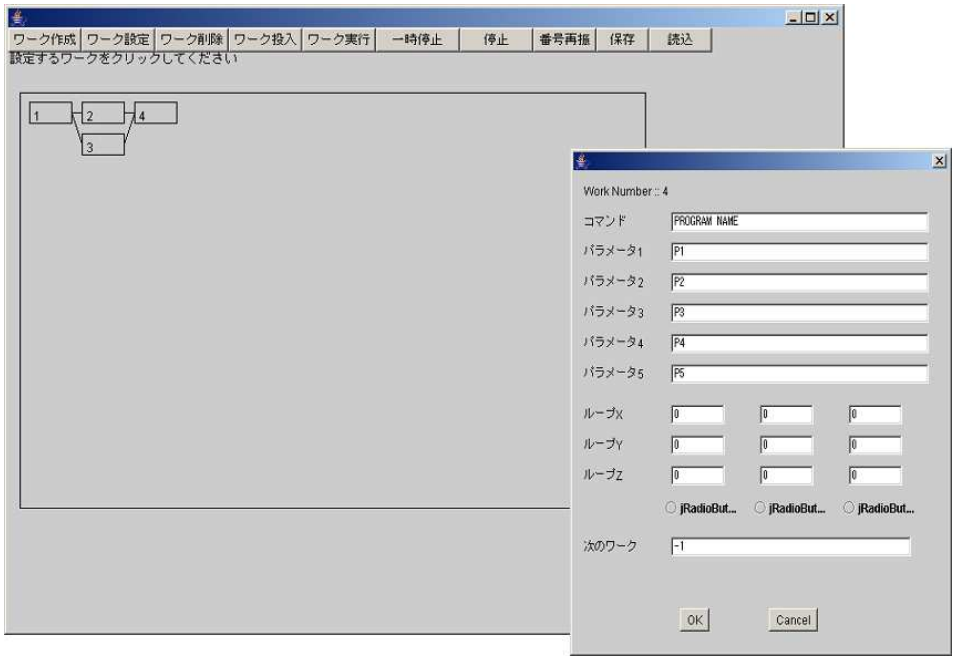


図 6.6　タスクフロー記述支援ツール

ユーザインターフェイス

シミュレーション実施者がタスク実行制御のために必要なユーザインターフェイスはタスク実行制御用インターフェイスと結果確認用インターフェイスの 2 点である。

- タスク実行制御用インターフェイスについては、図 6.6 に示すタスクフロー記述支援ツールを利用する。

  タスクフロー記述支援ツールでは、実行するプログラムの名前、プログラムを実行するために必要なパラメータ、プログラムの実行順序を入力する。タスクフロー記述支援ツールはタスク実行支援フレームワークと連動させており、タスク実行支援フレームワークへのタスクフローの投入が可能になっている。したがって、タスク情報、タスクフロー情報の修正はタスクフロー記述支援ツールを用いて情報修正をするだけでよい。

- 結果出力用インターフェイスについては、Web ブラウザを利用した結果表示ツールが ROKKY のツールとして提供されている。したがって、ROKKY が提供する結果表示ツールを結果出力用インターフェイスとして利用する。

  なお、結果出力用インターフェイスとして用いるための修正はない。

タスク実行支援フレームワーク適用によるシミュレーション実施者への効果

タンパク質立体構造予測システムの構築においてタスク実行支援フレームワークを適用することで、シミュレーション実施者は以下の効果が得られる。

- シミュレーション実施者が操作することで、実施者の希望に基づいたタスク位置からのタスクフロー実行が可能になる。これにより、各タスクにおけるパラメータの設定ミスに対して早期発見と対策が可能になる。
- タスクフローデータを部分的にかつ簡単に更新できるため、より複雑な依存関係をもつタスク実行であっても、シミュレーション実施者はタスクフローの修正動作だけで異なるパラメータをもつタスクを実行できるようになる。

6.2.2 対話的動作を考慮することによるシミュレーション結果の改善効果

シミュレーション実施者の意図的な動作によってタンパク質立体構造予測の結果がどのように変化するかを確認するために、シミュレーション実施者の意図的な操作が必要な予測配列を入力として効果を確認する実験を行った。

計算機環境

タンパク質立体構造予測システムで使用した計算機の性能を示す。

タスクフロー管理サーバおよびメタタスクディスパッチャのスペックはそれぞれ Intel 社 1.0GHz Pentium III プロセッサ 1 基を搭載した PC、Intel 社 2.53GHz Pentium 4 プロセッサ 1 基を搭載した PC を用いた。計算ノードは、独立した PC 1 台と PC クラスタ 1 台を用いた。計算機のスペックは、それぞれ Intel 社 2.53GHz Pentium 4 プロセッサ 1 基を搭載した PC、Intel 社 2.53GHz Pentium 4 プロセッサ 1 基を搭載した PC を 1 台と Intel 社 1.0GHz Pentium III プロセッサ 1 基を搭載した PC を 6 台で構成されたヘテロジニアスクラスタである。PC クラスタのタスクディスパッチャは Pentium 4 プロセッサ搭載の PC が担当する。計算結果解析サーバはタスクフロー管理サーバとの兼用とした。

ROKKY では 1 台でも多くの計算ノードを必要とするため、タスクディスパッチャとなる Pentium 4 プロセッサ搭載の PC も計算ノードとしても利用できるように設定をした。すべての計算ノードは 100Mbps の Ethernet で接続されている。

シミュレーション実施者からはタスクが単独で実行できる 1 台の計算ノードと PC クラスタ 1 台、合計 2 台の計算ノードが登録されているように見える。

予測配列

タンパク質立体構造予測は本来、タンパク質の構造が未知なものに対してコンピュータを利用することで、構造を決定するものである。本論文では、シミュレーション実施者の意図的な動作によってタンパク質立体構造予測の結果がどのように変化するかを確認することが目的であるため、ある予測配列の天然構造は未知であると仮定して、予測配列から立体構造を求めることにした。

入力となる予測配列は CASP6 において出題された問題の中から、Target T0198 および Target T0215 の 2 種類を用いた。T0198 および T0215 の特徴を以下に示す。

- **T0198**
  235 残基の予測配列をもつ問題である。ROKKY による予測計算において、2 つのドメインに分割し、それぞれのドメインを SimFold を用いて立体構造予測を行ってしまうことが過去の実験で明らかになっている。2 つのドメインで計算しても希望に沿った予測結果が得られないことがわかっているため、希望どおりの予測結果を得るためには、1 つのドメインで SimFold を実行するようにタスクの情報を修正する作業が必要になる。

- **T0215**
  76 残基の予測配列をもつ問題である。ROKKY による予測計算において、SimFold の実行時に用いられるパラメータの 1 つである、フラグメントライブラリと呼ばれるデータが初期状態の場合ではよい結果が得られないことが過去の実験で明らかになっている。希望どおりの予測結果を得るためには、修正したフラグメントライブラリを使って実行するようにタスクの情報を修正する作業が必要になる。

タスクフロー情報修正の動作シナリオ

提案するタンパク質立体構造予測では、タスクフロー情報の修正がシミュレーション実施者の対話的動作を意味する。シミュレーション実施者によるタスクフロー修正の操作があった場合とない場合によって立体構造がどのように変化するか確認するために、それぞれのシナリオを示す。

まず、シミュレーション実施者によるタスクフロー修正の操作を行った場合のシナリオを以下に示す。

1. 図 6.5 に示した、CM、FR、NF の順に実行するタスクフローデータを作成する。

2. タスクフローデータに基づいたシミュレーションを行う。

3. 適当な時間間隔で出力結果を確認する。

4. 出力結果がふさわしくないと判断した時点でタスクフロー記述支援ツール (図 6.6) を用いてタスクの情報修正をする。

   - T0198 については、1 つのドメインで修正するように NF タスクの情報を修正する。具体的には、以下に示すように NF タスク 3 番目の引数を 0 にする (デフォルトは 1)。

     ```
     T0198 -d 0 fragment.dat
     ```

   - T0215 については、修正されたフラグメントライブラリを用いて実行するように NF タスクの情報を修正する。具体的には、以下に示すように NF タスク 4 番目の引数を fragment2.dat にする (デフォルトは fragment.dat)。

     ```
     T0215 -d 1 fragment2.dat
     ```

5. 修正したタスクフローで再実行する。

一方、シミュレーション実施者による操作を行わなかった場合の動作のシナリオは 1、2、3 の操作だけである。

シミュレーション実施者によるタスク実行操作の比較

シミュレーション実施者がシミュレーション結果を逐次確認した後、タスクフローによるシミュレーションの再実行が必要な場合の動作を比較する。シミュレーションの再実行が必要と判断し、問題解決システムにシミュレーションの再実行を反映させるには、

1. 実行中のタスクフローを放棄して、シミュレーションの実施を中断する

2. 実行途中のタスクを確認し、タスクフローの位置と対応させる

3. タスクフロー情報を修正し、新たなタスクフロー情報を生成する

4. 新たなタスクフロー情報を問題解決環境に投入する

5. 新しいタスクフロー情報に基づいて、シミュレーションの実行を再開する

表 6.1 シミュレーション実施者によるタスク実行制御動作の比較

動作項目	既存ミドルウェアのみ	フレームワーク適用後
1. 実行中タスクフローの放棄	実施者が操作	フレームワークが吸収
2. 実行途中位置の確認	実施者が確認	フレームワークが吸収
3. タスクフロー情報の修正	実施者が修正	実施者が修正
4. タスクフローの投入	実施者が操作	実施者が操作
5. タスクフローの再実行命令	実施者が操作	フレームワークが吸収

の手順が必要になる。

表 6.1 はシミュレーション実施者が結果の継続的改善を行う際に生じるタスク実行動作を既存グリッドミドルウェアのみを利用した場合と提案するフレームワークを適用した場合で比較した結果である。タスク実行支援フレームワークを適用することで、シミュレーション実施者は 5 つの動作から、3. タスクフローの情報修正と 4. タスクフローの投入の 2 つの動作に軽減できる。

また、3. においては、タスクフローの差分実行を行うためには、従来はタスク情報とタスクフロー情報の 2 種類を変更しなければならない。タスク実行支援フレームワークを適用することでタスク情報を修正するのみになるため、タスクフロー情報の修正作業を省くことができる。

タスク実行支援フレームワークを適用した場合の動作の流れは以下のようになる。

1. シミュレーション実施者の作業

   (a) シミュレーション結果を確認する。
   タンパク質立体構造予測システムでは ROKKY が提供している結果確認ツールを用いているため、これを用いて結果を確認する。

   (b) タスクフロー情報を修正し、新たなタスクフロー情報を生成する。
   図 6.6 に示したタスクフロー記述支援ツールを用いて該当するタスクのパラメータ情報を変更する。例えば、T0198 の場合はパラメータ 3 のボックスに 0 を入力し、タスクフロー生成のボタンを押すことで差分タスクフローが生成される。

   (c) 新たなタスクフロー情報を問題解決システムに投入する。
   タスクフロー記述支援ツールでは、タスクフローの実行支援も行っている。タ

スクフロー実行ボタンを押すことでタスクフローが問題解決システムに投入される。

2. タスク実行支援フレームワークによる作業

   (a) 実行中のタスクフローを放棄して、シミュレーションの実施を中断する。
   この作業はプログラム実行モジュールの `diff_wf()` と `task_abort()` の動作に相当する (図 4.13)。

   (b) 新しいタスクフロー情報に基づいて、シミュレーションの実行を再開する。
   この作業はプログラム実行モジュールの `make_wf()` と `task_dispatch()` の動作に相当する (図 4.13)。

以上の結果から、タスク実行支援フレームワークを適用することでシミュレーション実施者には下線部の作業が増えるものの、シミュレーション結果の改善に必要な作業が軽減できる。タスク実行制御を行わない場合と比較してもシミュレーション実施者の作業のみに抑えられるため、操作上の負担を抑えたシミュレーション実施が可能である。

シミュレーション結果の改善に関する考察

構築したタンパク質立体構造予測システムによってタンパク質立体構造予測のシミュレーション結果がどのように改善されたかについて評価する。

図 6.7 および図 6.9 は T0198 および T0215 のそれぞれにおいて、シミュレーション実施者によるタスクフローデータの修正を行った場合と行わなかった場合でタンパク質立体構造予測を行ったときの正解との類似度の時間的変化を示したものである。横軸は時間を縦軸は最小二乗誤差 (Root Mean Squared Distance; RMSD) を示す。RMSD は天然立体構造 (正解) の i 番目原子と予測立体構造の j 番目原子との距離 d_{ij} を用いて下式のように定義される。

$$RMSD = \sqrt{\frac{1}{n}\sum_{i,j=1}^{n} d_{ij}^2} \tag{6.2}$$

このとき、n は原子数を表す。

T0198 および T0215 のシミュレーションにおける立体構造をそれぞれ図 6.8 および図 6.10 に示す。図中 Auto はタスクフローの修正を行わなかった場合、図中 Proposed はタスクフローの修正を行った場合の結果である。なお、タスクフローの修正を行わなかった場合の

結果は、ROKKY をそのまま利用した場合に相当する。一方、タスクフローの修正を行った場合の結果はシミュレーション実施者の操作に基づいた場合に相当する。

T0198 では 2 つのドメインから 1 つのドメインで実行するようにタスクフローの修正を行ったため、図 6.8 P12 のように 1 つのドメインになり、正解の構造に近づいている。切り替えた時点では RMSD の値が 35.98 とよくないが、計算を続けていくうちに切り替えてから 55 時間後 10 以下になっている。一方、タスクフローの修正を行わなかった場合、2 つのドメインのまま計算を継続しているため、図 6.8 A12 のように α と β の部分に分離している。計算を継続しても RMSD の値が 17 を推移したままであることが図 6.7 Auto のグラフから読みとれる。よって、これ以上計算を継続しても良い結果が得られないことがわかる。

また、T0215 では修正したフラグメントライブラリを用いるようにタスクフローの修正を行ったため、図 6.10 P22 の構造が正解の構造に似た構造になった。タスクフローデータの修正を行った時点 (図 6.9 P21 のあたり) では RMSD の値が 7.89 とよくないが、切り替えてから 14.5 時間後の 34 時間後には 5 以下になっている。一方、タスクフローの修正を行わなかった場合はフラグメントライブラリが初期状態のままなので、RMSD の値が 6.6 を推移しており、それ以下になる兆しが見られない。よって、図 6.9 から正解に似た構造が得られるとは考えにくい。

T0198 および T0215 の実験では 1 回の修正のみだったが、修正を行わなかった場合、すなわち全自動で立体構造予測シミュレーションする場合の結果の改善は緩やかに推移している。一方、修正を行った場合、すなわち対話的動作によって立体構造予測シミュレーションする場合の結果の改善は対話的動作を行ったあとに急激に推移している。これらの結果から、タンパク質立体構造予測シミュレーションに提案するタスク実行支援フレームワークを用いることで、立体構造予測結果を改善できる効果がある。

1 タスクあたりの処理時間

T0198 および T0215 を予測配列として入力し、立体構造予測シミュレーションを行ったときの CM と FR の平均実行時間を表 6.2 に示す。また、T0198 および T0215 における SimFold 単体の平均実行時間をそれぞれ表 6.3、表 6.4 に示す。

表 6.3 は残基数が 111 残基、124 残基、235 残基の平均実行時間を示している。T0198 の場合、残基数は 235 残基であるが、シミュレーション実施者による修正操作を行わない場合、NF ターゲットの計算において 2 つのドメインに分割して SimFold を実行する判断を下す。ここでは、残基数とプログラムの実行時間との関係を示したいため、参考までに 111 残基と 124 残基の結果を示した。

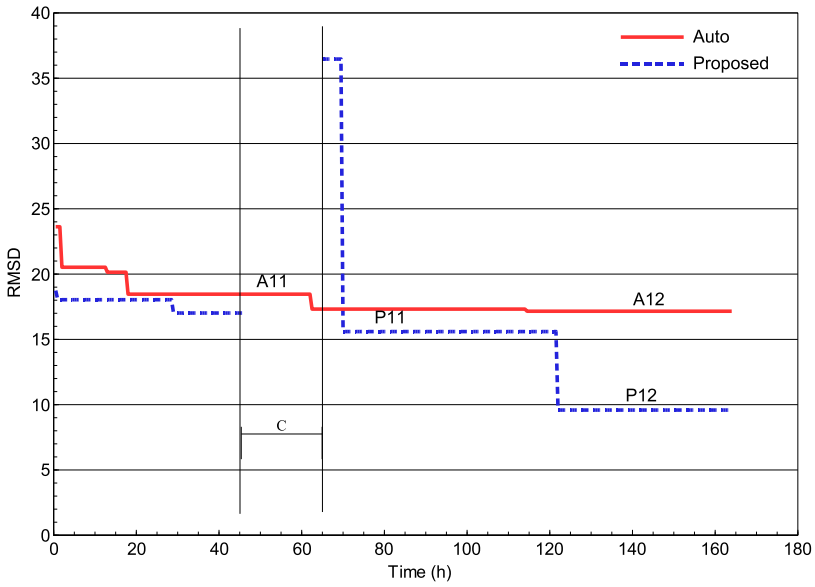


図 6.7 RMSD 最小値の時間的変化 (T0198)

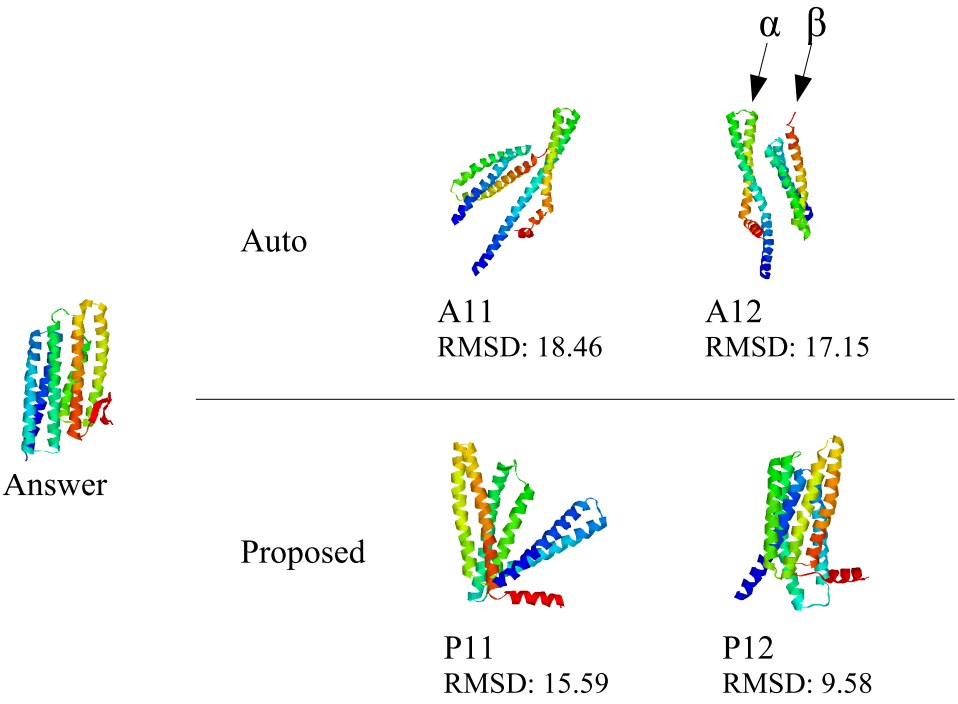


図 6.8 正解との立体構造の比較 (T0198)

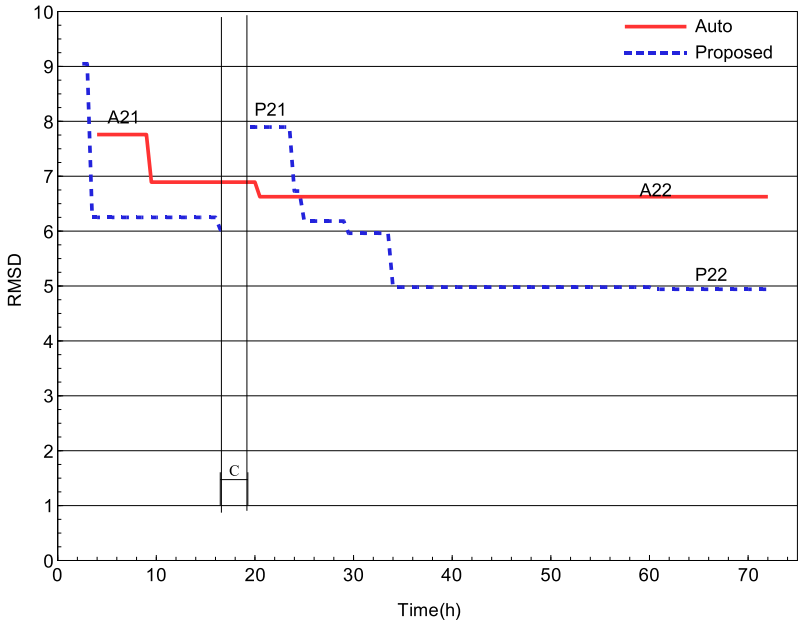


図 6.9　RMSD 最小値の時間的変化 (T0215)

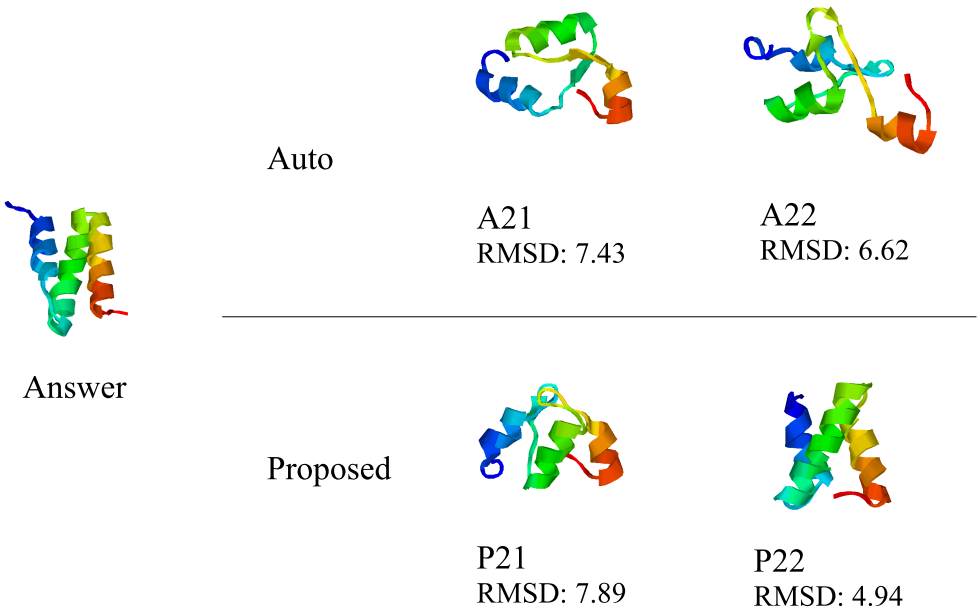


図 6.10　正解との立体構造の比較 (T0215)

表 6.2 CM および FR の平均実行時間 (T0198, T0215)

	T0198	T0215
CM	3'26"	1'14"
FR	9'05"	5'32"

表 6.3 SimFold プログラム単体の平均実行時間 (T0198)

残基数	111	124	235
Pentium III 1GHz	2h32'00"	2h58'00"	9h44'00"
Pentium 4 2.53GHz	1h20'00"	-	4h55'00"

表 6.4 SimFold プログラム単体の平均実行時間 (T0215)

残基数	76
Pentium III 1GHz	1h10'00"
Pentium 4 2.53GHz	0h35'00"

これらの結果から、CM、FR、NF 各タスクの実行時間は残基数に影響を受ける。特に、NF は長時間のタスク実行が余儀なくされることがこれらの結果からわかる。

タスクフロー差分実行機構の適用によるシミュレーション実施への影響

タスクフロー差分実行機構の導入によって、シミュレーション実施者によるタスクフローの修正が容易になったが、それに伴いシミュレーション実行時間に影響を及ぼす。

シミュレーション実施者による操作を行った際、T0198 においては 45 時間から約 19 時間分の結果が、T0215 においては 17 時間から約 2 時間分の結果が出力されていないことが図 6.7 および図 6.9 からわかる。それぞれの図中 C が結果が出力されていない時間帯に該当する。

これは、ROKKY の結果出力モジュールの処理上の制限によるものであることがわかっている。ROKKY の結果出力プログラムにおいて、出力結果の統計処理としてクラスタ分析を行う。統計処理の精度を維持するために SimFold から出力される計算結果が最低 10 個必要であるとの経験則が得られている。実験では、ヘテロジニアスなクラスタではあるが 7 台の PC で構成されたクラスタを用いて SimFold を実行しているため、少なくとも各計算ノードで 2 タスク以上実行しなければ統計処理ができない。T0215 では修正前も修正後も同じ残基数のシミュレーションを行っているため、表 6.4 で示した Pentium III プロセッサ搭載の計算ノードでの SimFold プログラム実行時間の 2 倍とほぼ一致する。また、T0198 では 2 つのドメインから 1 つのドメインで実行するように切り替えたため、SimFold で計算すべきドメインの残基数が大きくなる。表 6.3 で示した 235 残基の結果から Pentium III プロセッサ搭載の計算ノードでの SimFold プログラム実行時間の 2 倍とほぼ一致している。これらの結果から、提案システムによる結果に一定時間の出力がないのは、1 タスクあたりの実行時間に影響を受けていることがわかる。

なお、タスクフロー差分実行機構が実行したときの実行時間のオーバヘッドについて、タスクフロー管理サーバで測定した。シミュレーション実施者によって修正されたタスクフローを入力し、それに切り替わるまでの時間は 20 秒であった。したがって、タンパク質立体構造予測の実施において、タスクフロー差分実行機構のオーバヘッドによる影響は受けない。

シミュレーション実施者が頻繁にタスクフローの修正を行い、シミュレーション結果の改善動作を行うと想定した場合、結果出力の待ち時間が累積されるので、却って希望とする結果が得られない場合も考えられる。特に、残基数が大きい場合には結果出力にかかる待ち時間がオーバヘッドとなることがわかった。

システム構築における改善策としては、1 タスクあたりの実行時間短縮と統計処理に最低限必要とされる計算機台数の確保が考えられる。前者については、SimFold の実行時間が残基数に比例することが表 6.3 と表 6.4 から明らかななので、SimFold の実行時間を短縮するだけの計算機能力を確保する。後者については、最低でも 10 タスクが並行して実行できるだけの計算機台数を確保する。

また、頻繁に改善動作を行うことを想定した場合には、複数パラメータの実行を可能になるだけの計算機台数を確保することでよい結果を得るための時間が短縮できる。

6.3. まとめ

本章ではタスク実行支援フレームワークを適用した事例としてタンパク質立体構造予測システムの構築について述べた。

タンパク質立体構造予測の現場では、立体構造予測の自動化に向けて研究開発が進んでおり、継続的改善によって立体構造予測に関する知見を得ることが重要であることに触れた。

本章では、ROKKY と呼ばれるタンパク質立体構造予測システムをベースとして、タスク実行支援フレームワークを適用したシステムを構築した。タンパク質立体構造予測では、複数の解析手法を用いて結果を得ることから、タスクフロー技術を導入した。タスクフロー技術の導入に際し、従来のタスクフローシステムだけではシミュレーション実施者の意思決定による継続的改善操作が困難なため、タスクフロー差分実行機構を提案、実装し、シミュレーション実施者の操作の容易化と意思決定支援を可能にした。

最後に、タスク実行支援フレームワークによるタンパク質立体構造予測にもたらす効果について述べた。シミュレーション処理時間の短縮については、残基数が増えることによるシミュレーション処理時間の影響を受けるため、残基数に応じた計算機の処理能力が必要であることがわかった。シミュレーション結果の改善については、全自動による結果に比べて正解と思われる結果に近づくことから、対話的動作による効果があることを示した。

第7章

マルチエージェントシミュレーションに基づく託児所配置問題への適用

7.1. マルチエージェントシミュレーションと託児所配置問題

7.1.1 マルチエージェントシミュレーション

マルチエージェントシミュレーションは、従来人工知能分野で研究が行われてきた人間の知能モデルをシミュレーションに展開した研究分野である。コンピュータならびにネットワーク技術の発展を背景として、1990 年以降急速に発展した研究分野である。一般的には、エージェントが相互影響を受けることで対象となる場の振舞いを定量的に示す方法と考えることができる。ここで指すエージェントとは、知覚、認識、記憶、学習、判断、行動といった人間がもっている情報処理機能をモデル化したものである。

マルチエージェントシミュレーションの適用範囲は広い。特に、社会科学分野では、エージェントの振舞いを人間の振舞いとして置き換えることにより、社会における発見や定式化を支援することができると考えられている。マルチエージェントシミュレーションの社会科学の応用として、文献 [61] では、サッカーシミュレーション、交通流・人流シミュレーション、金融市場シミュレーション、災害・防災シミュレーションの例を挙げている。また、既婚女性の労働供給に関する適用例 [62] や、国民年金納付問題への適用例 [63] などがある。

社会科学分野においてマルチエージェントシミュレーションが注目されている理由は、社会現象が実験により検証することが困難なことにある。自然科学分野では現象を実験室な

どで再現できるが、社会科学分野では人間の思考、動作、決断を対象としているため、実験室で現象を再現することができない。そのため、マルチエージェントシミュレーションに期待が寄せられている。

7.1.2 託児所配置問題

日本が直面している課題のひとつに少子化問題がある。厚生労働省が発表した平成 17 年人口動態統計の年間推計[1] において、平成 16 年の合計特殊出生率[2] は 1.29 である。この数値は将来における日本の総人口減少を意味している。労働力の低下、経済活動の停滞、高齢化社会への変化をもたらし、様々な社会問題を引き起こす。

厚生労働省が発行した平成 17 年度版 労働経済白書 [64] において、女性の意識と就業促進に向けた課題がまとめられている。その中で、女性の就業行動は出産および育児によって大きく影響を与えていること、すなわち、就業と育児の両立が困難であることが示されている。また、財務省財務総合政策研究所が発行した「少子化の要因と少子化社会に関する研究会」報告書 [65] において、就業による出産抑制効果は育児休暇制度や保育サービスの充実により緩和されることが提言されている。これらの報告から、少子化対策のためには女性の就業行動を促進すること、具体的には、保育サービスをどう充実させるかが解決の糸口であるといえる。そこで考えられるのが、託児所設置による非就業者の就職行動促進に関する検討である。本論文では、託児所配置問題と呼ぶ。

託児所配置問題は、託児所の配置によって子供をもつ女性が就業にどのような影響をもたらすかを知ることである。具体的には、職場との位置関係によって女性が就業するか、性格の異なる住民間の相互作用によって就業するかどうかを検討することである。この問題を解決することは、地方公共団体においては住民サービスの提供につながる。また、日本政府においては少子化の解消とともに、税収向上も期待できる。

7.1.3 マルチエージェントシミュレーションに基づく託児所配置問題

託児所配置問題は、職場との位置関係によって女性が就業するか、性格の異なる住民間の相互作用によって就業するかどうかを検討することであると述べた。すなわち、ある地域の女性は、会社、託児所、近隣の女性からの影響を受ける。これらの相互影響により就

[1] http://www.mhlw.go.jp/toukei/saikin/hw/jinkou/suikei05/index.html

[2] その年次の 15 歳から 49 歳までの女子の年齢別出生率を合計したもの。一人の女子が仮にその年次の年齢別出生率で一生の間に生むとしたときの子どもの数に相当する。

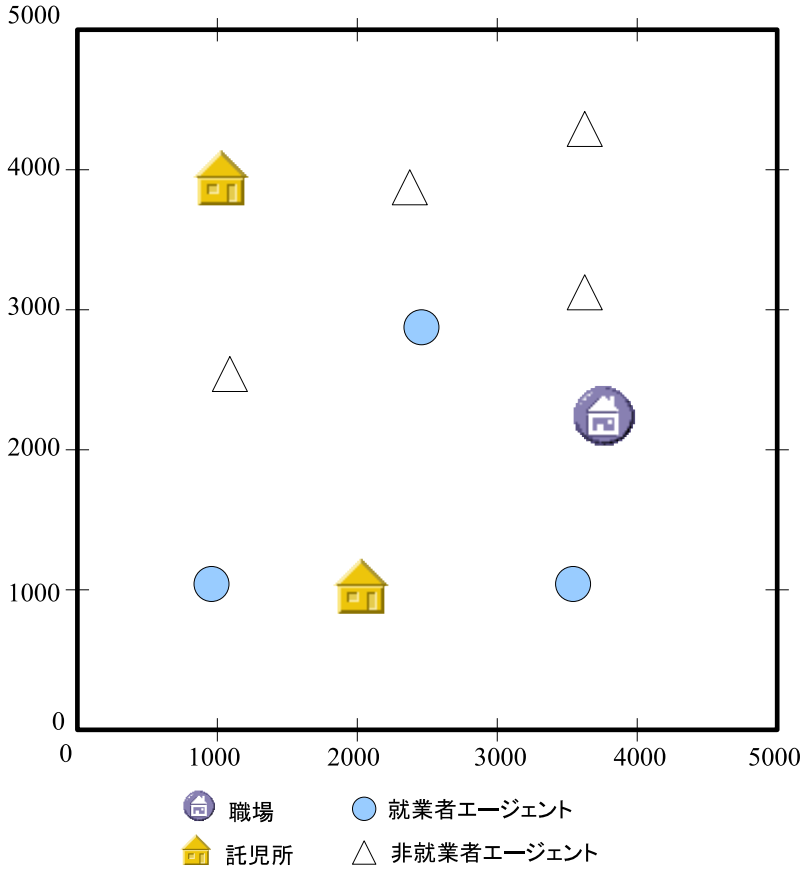


図 7.1 託児所とエージェントの位置関係

業するかを判断する問題と置き換えることができる。このように、何らかの相互影響を受ける問題と解く場合には、マルチエージェントシミュレーションが得意とする。本論文では、マルチエージェントシミュレーションによる託児所配置問題として、村田らが提案する方法 [66] を取り上げる。

本論文で取り上げる託児所配置問題は、図 7.1 に示すように、非トーラス状の 2 次元空間内に職場、託児所、エージェントが存在する。

託児所配置問題においてエージェントは女性とする。各エージェントは就業行動を行うか否かを決定する。エージェントが就業するかを判断する基準は、効用差関数、エージェントの知覚範囲、周囲のエージェントとの相互影響である。効用差関数とは、年齢、教育年数、配偶者の収入、子供の数を変数とした関数である。エージェントの知覚範囲とは、エージェントが就業するか否かを判断する際に情報を収集する範囲である。知覚範囲に託児所がある場合は就業行動をとりやすいと仮定する。周囲のエージェントとの相互影響は、

知覚範囲内に存在する就業中のエージェントが多いほど就業選択を行うと仮定する。本論文で取り扱う問題におけるエージェントは、周囲のエージェントが就業していなくても就業行動を率先して行うリーダエージェントと、周囲の行動に左右されるフォロワエージェントの２種類が存在するものとする。

7.1.4 託児所配置問題における対話的動作の必要性

マルチエージェントシミュレーションによる託児所配置問題は２つの特徴を有する。

- マルチエージェントシミュレーションは確率論的手法である。
  入力データの一部にある確率分布もしくは乱数を使うため、統計的に十分な数の計算を繰り返す。

- パラメータの個数が増えるとパラメータの組み合わせ数が指数的に増大する。
  託児所配置問題は託児所の適切な配置を検討することが目的である。託児所の適切な配置をみつけるために、例えば、託児所の位置、職場の位置、エージェントの初期条件を変更する。その結果、シミュレーションを実行する数が指数的に増える。

村田らの実験では、各実験条件において場合分けをできるだけ少なくなるように、職場の位置、エージェントの人数比 (リーダとフォロアの比率)、託児所の位置、エージェントが託児所を選択するうえで許容できる距離がそれぞれ 4,9,7,3 通りとし、かつエージェントの分布中心は職場の位置が中央にある場合は 12 通り、それ以外は８通りとしている。これらの値から算出されるシミュレーションの実行数は 6804 通りになる。なお、6804 通りのシミュレーションを Pentium4 2.8GHz の計算機で１台で実行した実験を行っており、その結果、317 時間 54 分 46 秒かかったと報告している。

より詳細なシミュレーションを行う場合、例えば、職場の位置、託児所の位置を増やした場合の就業選択行動に関する効果を確認する場合、シミュレーションの実行数の増加が容易に想像できる。考えられる限りでのパラメータの組み合わせを作成し、パラメータの組み合わせすべてを総当たりで計算する従来の考え方では、パラメータの組み合わせ数が増えれば、シミュレーションに要する時間も増加する。

ここで、託児所配置問題に取り組む政策立案者の立場を考える。政策立案者はシミュレーションから得られる結果に政策的な意味をもつため、具体的な条件を設定したシミュレーションを希望する。また、最適な託児所配置の条件を知るために、特定のパラメータを与えてはタスク実行をするといった動作が行われる。

7.1.5 託児所配置問題解決システム構築における対話的動作実現の困難さ

託児所配置問題では、パラメータの組み合わせによって作成される解集合から評価値が最大となるパラメータの組み合わせを探索することが要求される。既存の技術を用いて託児所配置問題解決システムを構築する場合、以下の方法が考えられる。

- 想定されるパラメータの組み合わせすべてを試す方法 (自動的に実行する場合)

  長所 タスク実行が自動化可能であること。
  短所 解集合の数によって問題解決にかかる時間が増大すること。

- パラメータの組み合わせを一つずつ試す方法 (対話的に実行する場合)

  長所 シミュレーション実施者の結果を確認しながら解探索ができること。
  短所 問題解決にかかる時間が増大すること。シミュレーション実施回数が大幅に増えること。

シミュレーション実施者は、パラメータ群の集合からサンプルとして一部の解候補を取り出して実行し、その結果を確認したのち再度別の解候補を実行する、という繰り返し操作が望ましい。したがって、これらの中間的な方式がシミュレーション実施者にとって望ましいといえる。

7.2. タスク実行支援フレームワークに基づく託児所配置問題解決システム

マルチエージェントシミュレーションによる託児所配置問題に求められる目標は、大量の解集合から、託児所配置で最大の効果を得ることができる解を探索することである。このことを踏まえ、タスク実行支援フレームワークを用いて対話的動作に基づく託児所配置問題解決システムを構築する。

```
0 1000 1000 1000 -1 -1 2 2 1000 1000 1000 4000 4000 4000 100 0.7 →
output_0-1000_1000_1000_-1_-1_2_2_1000_1000_1000_4000_4000_4000_100_0.7.dat
0 1000 1000 1000 -1 -1 2 2 1000 1000 1000 4000 4000 1000 100 0.7 →
output_0-1000_1000_1000_-1_-1_2_2_1000_1000_1000_4000_4000_1000_100_0.7.dat
0 1000 1000 1000 -1 -1 2 2 1500 4000 1000 4000 4000 1000 100 0.7 →
output_0-1000_1000_1000_-1_-1_2_2_1500_4000_1000_4000_4000_1000_100_0.7.dat
0 1000 1000 1000 -1 -1 2 2 2500 2500 1000 4000 4000 1000 100 0.7 →
output_0-1000_1000_1000_-1_-1_2_2_2500_2500_1000_4000_4000_1000_100_0.7.dat
```

(以下続く。紙面の都合上、→は行の続きを意味する。)

図 7.2　パラメータリスト例

7.2.1　タスク実行支援フレームワークへの適用とシミュレーション実施者への効果

パラメータリストファイルの準備と大量タスク実行スクリプトの適用

託児所配置問題解決システムでは、各パラメータの値の組み合わせによって作成される大量のパラメータの組から託児所配置で最大の効果を得る各パラメータの値を探す。いわゆるパラメータスイープ型のシミュレーションである。

本論文で提案する託児所配置問題解決システムでは、まず、シミュレーション実施者が図 7.2 に示すようなフォーマットでパラメータリストファイルを準備する。パラメータリストファイルの 1 行は 1 つのタスクとして割り当てる引数を意味する。また、行数はタスク実行数を意味する。

次に、大量タスク実行スクリプトを用いて、大量タスク実行を可能にする。具体的には、図 7.3 に示すタスクフロー情報を生成し、タスク実行支援フレームワークに投入する。本論文で適用した託児所配置問題の場合、大量タスク実行を行うのみなので、タスクフロー情報に記載されるタスク情報はひとつだけである。

ユーザインターフェイス

シミュレーション実施者が優先タスク実行を可能にするために必要なユーザインターフェイスはパラメータ変更用操作インターフェイスと結果確認用インターフェイスの 2 種類である。

```
1: 1
2: submitter.pl
3: calculateVersion4 renew.bat
4:
5:
```

図 7.3 託児所配置問題におけるタスクフロー (左側は行番号を意味する。)

- パラメータ変更用操作インターフェイスは、図 7.4 に示すように、Web ブラウザをインターフェイスとして採用する。

  パラメータ変更用操作インターフェイスでは、初期パラメータリストの入力、優先実行させるパラメータ値の入力が可能である。

- 結果出力インターフェイスは、パラメータ変更用操作インターフェイスと同様、図 7.5 に示すように、Web ブラウザをインターフェイスとして採用する。

  託児所配置問題解決システムでは、パラメータごとに結果が出力される。シミュレーション実施者が結果を確認する場合、パラメータごとに結果を確認する場合と複数の結果を比較しながら確認する場合が考えられる。そこで、結果確認用インターフェイスでは、タスク実行が完了したパラメータリストの一覧から、シミュレーション実施者が興味をもつパラメータリストを選択する (図 7.5 左側)。そして、選択されたパラメータに対してグラフ出力するように施した (図 7.5 右側)。これにより、パラメータの違いによるシミュレーション結果の違いを比較できる。

タスク実行支援フレームワーク適用によるシミュレーション実施者への効果

託児所配置問題解決システムをタスク実行支援フレームワークに適用することによって、シミュレーション実施者は以下の効果が得られる。

- シミュレーション実施者の操作により実施者の希望に基づくパラメータの組を優先的に実行できる。これにより、パラメータの組をひとつずつ入力しながら結果を得るのと同じ効果が得られる。

- パラメータの組み合わせすべてを実行する必要がなくなる。これにより、最適解探索おける解候補の早期発見が可能になる。また、託児所配置問題では様々な条件下での

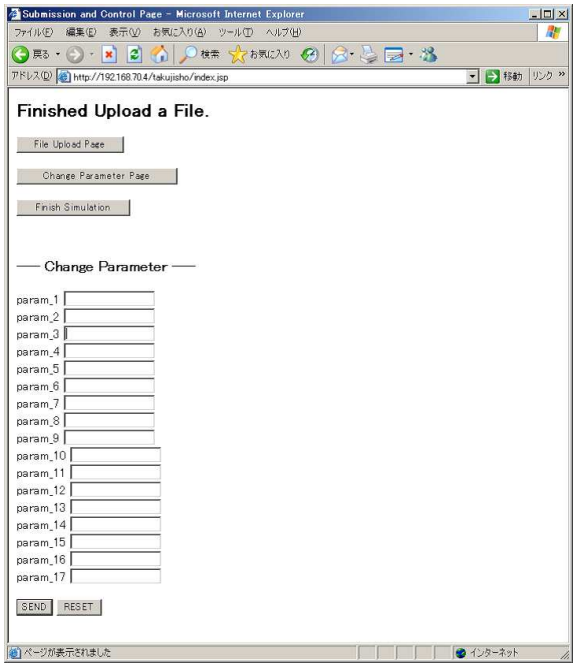


図 7.4　パラメータ変更操作ページ

シミュレーションを短期間で行うことが可能になる。

7.2.2　対話的動作を考慮することによるシミュレーション結果の改善効果

シミュレーション実施者の意図的な動作が継続的に行われることで、託児所配置問題の結果にどのような改善効果をもたらすかについて述べる。

計算機環境

託児所配置問題解決システムで使用した計算機の性能を示す。プロトタイプシステムでは PC クラスタを利用した。PC クラスタは Intel 社 3.0EGHz Pentium 4 プロセッサを搭載した PC を 10 台で構成したホモジニアスクラスタである。すべての PC は 1Gbps の Ethernet ポートを有しており、プロセッサ間の通信速度は 1Gbps である。

PC クラスタでのジョブ管理は Condor を使用した。使用した Condor のバージョンは 6.6.7 である。Condor では実行するプログラムを各計算ノードにコピーする機能を有しているので、プロトタイプシステムではこの機能を利用して、各パラメータの計算結果を回収する。

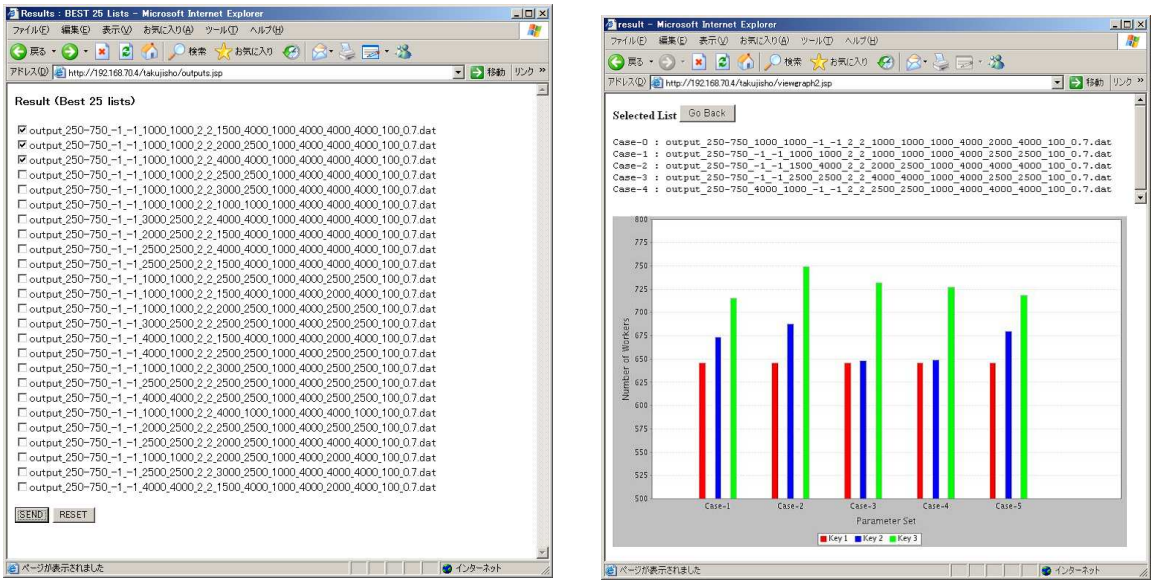


図 7.5 出力結果確認ページ (左:選択画面, 右:グラフ描画)

PC クラスタを構成する計算機のうち 1 台は、プログラム実行モジュール、状況確認モジュール、結果解析モジュールを担当する。このノードをサーバノードと呼ぶ。残り 9 台はすべて計算ノードとした。なお、サーバノードは計算ノードの作業も行うため、シミュレーション実施者からは 10 台の計算ノードが登録されているように見える。

サーバノードではパラメータ入力および変更、シミュレーション結果の出力を可能にするために、Java Server Pages(JSP) を利用した。プロトタイプシステムでは Tomcat 4.1.30 を使用した。シミュレーション実施者は Web ブラウザを利用してパラメータの入力と変更、シミュレーション結果の確認を行う。

シミュレーション実施者によるタスク実行操作の比較

シミュレーション実施者がシミュレーション結果を逐次確認した後、パラメータリストの順序変更によるシミュレーションの再実行が必要な場合の動作を比較する。託児所配置問題はパラメータスウィープ型のシミュレーションであるから、大量のタスク実行が発生する。シミュレーションの再実行が必要と判断し、問題解決システムにシミュレーションの再実行を反映させるには、

1. タスク管理システムに投入されているタスクをすべて放棄して、シミュレーションの実施を中断する

表 7.1 シミュレーション実施者によるタスク実行制御動作の比較

動作項目	既存ミドルウェアのみ	フレームワーク適用後
1. 投入されたすべてのタスク放棄	実施者による操作	フレームワークが吸収
2. パラメータリストから実行完了したパラメータを分離	実施者による操作	フレームワークが吸収
3. パラメータの特徴抽出	実施者が確認	実施者が確認
4. 未実行パラメータリストの作成	実施者が作成	フレームワークが吸収
5. 新たなパラメータリストによる再実行命令	実施者による操作	フレームワークが吸収

2. 実行が完了したパラメータのタスクと実行されていないパラメータのタスクを整理する

3. 実行完了したタスクの結果から、評価値が最大になると思われるパラメータの特徴を抽出する

4. 特徴抽出結果から優先実行されるパラメータを上位にした未実行パラメータリストを作成する

5. 新しいパラメータリストに基づいて、シミュレーションの実行を再開する

の手順が必要になる。

表7.1はシミュレーション実施者がパラメータリストの変更を行う際に生じる動作を既存グリッドミドルウェアのみを利用した場合と提案するフレームワークを適用した場合での比較した結果である。タスク実行支援フレームワークを適用することで、シミュレーション実施者は5つの動作から、3. パラメータの特徴抽出に軽減できる。

タスク実行支援フレームワークを適用した場合の動作の流れは以下のようになる。

1. シミュレーション実施者の作業

   (a) シミュレーション結果を確認する。
   図7.5に示した出力結果確認ページを用いて、評価値が最大となるパラメータを確認する。出力結果確認ページでは、複数のパラメータリストを選択して評価値を比較するグラフが出力できる。また、パラメータ一覧を出力する際に、評

価値の高い順にパラメータを並び替えて出力することで、評価値を比較するためのグラフを確認する手間が減る。

(b) パラメータの特徴を抽出する。
出力結果確認ページで評価値を確認した結果、評価値が高くなると思われる引数の位置や値を確認する。

(c) 特徴抽出した結果をタスク実行支援フレームワークに指示する。
託児所配置問題解決システムではタスクフローの修正はないため、パラメータ変更操作のみである。図7.4に示したパラメータ変更操作ページを使う。パラメータ変更操作ページはタスクの引数位置と関連づけているので、該当するパラメータの位置に優先したい値を入力する。入力し終えたら、送信ボタンを押す。

2. タスク実行支援フレームワークによる作業

(a) 投入されているすべてのタスクを放棄する。
この作業は図4.20におけるプログラム実行モジュールの `task_abort()` の動作に相当する。

(b) パラメータリストから実行完了したパラメータを分離し、優先実行するパラメータを上位にした未実行パラメータリストを作成する。
この作業は図4.20におけるプログラム実行モジュールの `modify_plist()` の動作に相当する。

(c) 新しいパラメータリストに基づいてシミュレーションの実行を再開する。
この作業は図4.20におけるプログラム実行モジュールの `task_dispatch()` の動作に相当する。

この結果から、タスク実行支援フレームワークを適用することで、シミュレーション実施者には下線部の作業が増えるものの、解探索時間の短縮に必要な作業が軽減できる。また、パラメータリストを修正せずすべてのパラメータを実行する場合と比較した場合、結果確認とパラメータ修正依頼の動作に抑えることができるため有効であるといえる。

シミュレーション実施者の操作による解探索結果および解探索時間の変化

パラメータスウィープ型のシミュレーションは、一般的に各パラメータの値の集合から、すべての組み合わせを作成し、組み合わせに基づいてシミュレーションを行う。すべての

組み合わせによるシミュレーション結果が出力されたのち、目的を満足する各パラメータの値を探しだす。

ところで、パラメータの組み合わせの多くは目的を満足するか判断しにくい組み合わせである。パラメータを与えてシミュレーションを行うことではじめて目的を満足するか否かが判別できる。従来は、すべての組み合わせをシミュレーションし終えてから、良い結果だけを選別する。目的を満足するパラメータの組み合わせを確実にみつけだすことができるが、シミュレーションに要する時間も長時間必要になる。

本論文で提案する託児所配置問題解決システムでは、シミュレーション実施者の知識と経験による操作を必要とするが、すべての組み合わせのシミュレーションを行わなくとも目的を満足するパラメータの組み合わせをみつけることができるようになる。そこで、すべての組み合わせのシミュレーションを行わなくとも目的を満足するパラメータの組み合わせをみつけだせるか確認するための実験を行った。実験における条件は以下の通りである。

- 託児所配置シミュレーションの環境条件
  本実験で変更可能なパラメータは職場位置、エージェントの種類、エージェントの中心分布、託児所設置位置とする。各パラメータの数はそれぞれ 4 通り、2 通り、7 通り、7 通りとする。これら以外の条件および具体的な値は村田らの報告と同様とする。変更可能なパラメータから作られるパラメータの組み合わせは 394 通りとなる。

- 初期パラメータリストファイルの条件
  本実験で用いたパラメータリストは一様乱数によってパラメータの順序を変更する。パラメータの順序を変更するために使った乱数は松本らによって開発された Mersenne Twister と呼ばれる疑似乱数生成アルゴリズム [67] である。

- 託児所配置問題解決システムを操作する被験者の振る舞い
  被験者は常に問題解決システムの前に座ったまま実験するのではなく、被験者の都合の良いタイミングでシミュレーション結果の確認をしてもらうようにした。シミュレーション結果を確認する時点で、希望する結果と異なると判断した場合、パラメータ変更用ユーザインターフェイスを用いてタスクの優先実行をしてもらった。そして、託児所配置の評価値が最大になるパラメータの組み合わせが、以後現れないと思うまで被験者の意思決定に基づくパラメータ変更の操作を行ってもらう。

  なお、パラメータ変更の操作はすべてのパラメータの組み合わせが終わるまで可能とする。

表 7.2 被験者による結果の改善効果 (ソートなし)

	上位 5 位内	上位 10 位内	タスク数	変更回数
被験者 1	×	×	193	7
被験者 2	○	○	173	10
被験者 2	○	○	296	4

表 7.3 被験者による結果の改善効果 (ソートあり)

	上位 5 位内	上位 10 位内	タスク数	変更回数
被験者 1	○	○	202	8
被験者 2	○	○	172	8
被験者 3	○	○	283	9

- 結果出力の提示方法
  本実験では、被験者に対して 2 種類のシミュレーション結果を提示する。ひとつは、シミュレーション結果の良い順に並べたもの、もうひとつはシミュレーション結果が出力された順に並べたものである。差をつけることで、シミュレーション実施者の結果出力確認時の負担が及ぼす影響を確認する。

シミュレーション実施者の操作による結果の改善効果を表 7.2、表 7.3 に示す。表 7.2 は被験者に対して計算結果をそのまま提示した場合 (ソートなし) の結果、表 7.3 は被験者に対して上位 25 位までの計算結果を提示した場合 (ソートあり) の結果である。この実験では、3 人の被験者に 2 種類の実験をしてもらった。

プロトタイプシステムでは 10 台のホモジニアス PC クラスタを用いているため、最適解決定までのタスク数を解探索時間とみなした。最大タスク数は 394 個であるから、それよりも小さい値であれば解探索時間が短縮できたとみなせる。また、プロトタイプシステムで用いたマルチエージェントシミュレーションプログラムは内部で乱数を用いているため、パラメータリスト 1 つあたり 100 回の演算を行う。1 つのパラメータリストから出力される評価値は 100 回の演算結果の平均値とする。100 回の演算結果の平均値を評価値として採用しているため、シミュレーション全体としては一定の傾向を出力する。試行実験を繰り

返すことで、小さな範囲での順位変動があり得る。そこで、被験者が最適解と選択したパラメータの組み合わせが上位５位内もしくは上位 10 位内に収まっているかを確認した。

実験の結果、最適解を選択するまでのパラメータ変更回数、最適解を選択するまでのタスク数に大きな差は見られなかった。但し、ソートした計算結果を被験者に提示することで改善の兆しが得られた。被験者１においては上位５位内、上位 10 位内の結果を出力できるようになった。また、被験者２は変更回数が低減した。逆に、被験者３はソートしない結果を提示した場合のほうが変更回数が少なかった。被験者３はソートしない場合を提示された場合は最適解の選択作業に時間を費やしたが、ソートする場合を提示されると選択作業にかかる負担が軽減されるため、より多くのパラメータ変更を行う意欲が起きたと考えることができる。

最適解を選択するまでの解探索時間については、プロトタイプシステムを用いることで大幅に改善されたといえる。ソートした計算結果を被験者に提示した場合、被験者１については 48.8%、被験者２については 56.4%、被験者３については 27.8%、にあたるタスクの実行を削減できた。

組み合わせ数増加によるシミュレーション実施者の継続的改善の効果

託児所配置問題解決システムに対話的動作を考慮することで、パラメータに与える値の種類が増加しても、必要な計算結果を優先的に確認することが可能になる。

マルチエージェントシミュレーションでは、評価値を最大にするパラメータの値を探すために、各パラメータに対して想定される値の集合を準備することになる。シミュレーションプログラムを実行する回数、言い替えると組み合わせの数は、パラメータの数および各パラメータとして与える値の集合から指数的に増加する。

従来、マルチエージェントシミュレーションでは、指数的に増加する組み合わせの数を少しでも減らすために、マルチエージェントシミュレーションの定性的特徴と２次元空間の対称性といった問題特有の特徴を考慮してパラメータリストを作成していた。そのため、評価値を最大にする各パラメータの値の範囲を特定するにとどまっていた。

ここで、政策立案のような実務的な観点で使うことを想定する。実務的な観点では、パラメータの値は定性的なものではなく、定量的なものが要求される。従来は、予備実験として定性的なシミュレーションを行い、ある程度の絞り込みを行ってから具体的な値の特定を行うことになる。すなわち、同じシミュレーションを２回実施することになる。

これに対し、提案する託児所配置問題解決システムでは、単純に各パラメータに与える値の数を増やしても、シミュレーション実施者の継続的な改善によって、パラメータ値の

表 7.4 パラメータ数増加による結果の改善効果

	上位 5 位内	上位 10 位内	タスク数	変更回数
被験者 1	○	○	374	14
被験者 2	×	×	160	13

特定を 1 回の実施でよい。そこで、パラメータの組み合わせの数を増やし、かつシミュレーション実施時間を制限した状況下で、目的を満足する各パラメータの値をみつけだせるかの実験を行った。この実験では、シミュレーション実施者の継続的な改善による解探索の効果を確認する。

実験における条件は先の実験と同様の条件とするが、異なる点について示す。託児所配置シミュレーションの環境条件について、パラメータリストの数を増やすために、託児所設置位置を 7 通りから 12 通りに増やす。この変更によって作られるパラメータの組み合わせは 672 通りになる。

託児所配置問題解決システムを操作する被験者について、被験者の都合の良いタイミングでシミュレーション結果の確認をしてもらう。本実験では、シミュレーション実施時間を制限した状況を想定しているため、被験者の意思決定が反映できるのは、シミュレーション開始から 3 時間以内とする。これは、672 通りの半数にあたる 386 通りの組み合わせが完了する時間が 2.8 時間になることから決定した。

被験者 2 名の操作による結果の改善効果を表 7.4 に示す。先ほどの実験同様、被験者に対して上位 25 位までの計算結果を提示した。被験者 1 は制限時間直前までパラメータの変更を行い、最適解の可能性が高いと思われるパラメータの組み合わせを実行した結果を反映している。一方、被験者 2 は短時間での解決を試みたため、被験者 2 が選択した最適解は上位 10 位内にも含まれていない。パラメータの組み合わせ数を増やしてのシミュレーションでは被験者の判断ミスが発生することに注意しなければならない。

7.2.3 政策立案の現場へ適用するための課題

提案する託児所配置問題解決システムは、将来的には政策を立案する現場での活用を目指している。本論文では、託児所配置問題解決システムのプロトタイプを示したが、政策立案の現場に適用するには課題が残されている。本論文では、システム構築の観点から、

政策立案の現場で用いるための課題について述べる。

パラメータリスト作成における課題

プロトタイプシステムでは初期パラメータリストファイルを作成し、初期パラメータリストファイルの記述順にタスク実行が行われる。政策立案者が、政策上の条件をもとに定義されるパラメータの値を初期パラメータリストファイルに記述する必要がある。また、プロトタイプを用いた実験において、処理されたパラメータの組み合わせを一意に確認する仕組みを提供しなかったため、被験者の判断ミスが発生した。

対策としては、グラフィカルな操作インターフェイスを提供する方法が考えられる。グラフィカルな操作インターフェイスを介して、初期パラメータリストファイルの作成、優先実行するタスクの選択操作、計算結果を見ながら不必要と思われるパラメータリストの排除操作をできるようにする。

複数のグリッド環境を利用するための課題

政策立案者はマルチエージェントシミュレーションの定性的特徴を理解しているとは限らない。そのため、初期パラメータリストは政策立案上のあらゆる場合を想定して作成される。作成されたパラメータリストは大量のタスク実行を伴うことから、大量タスク実行に見合った計算機能力が必要になる。プロトタイプシステムでは、タスク管理システムとして Condor を採用した PC クラスタに適用した。シミュレーション時間の短縮を実現するには、必要に応じて複数の組織にある計算機を利用することが必要になる。

対策としては、パラメータ変更部において複数のタスク管理システムに対応したタスク投入モジュールを実装することが挙げられる。現状では、様々なタスク管理システムが用いられているが、操作体系そのものは Condor の場合と変わらない。プロトタイプで実装したコードをもとに、Condor 以外のジョブ管理システムに対応した改良を行う。

7.3. まとめ

本章では、マルチエージェントシミュレーションによる託児所配置問題について、タスク実行支援フレームワークに基づく託児所配置問題解決システムの設計とプロトタイプシステムの構築について述べた。

本章で取り上げたマルチエージェントシミュレーションによる託児所配置問題では、適切

な託児所の配置条件を導き出すために、パラメータスウィープによるアルゴリズムを採用している。パラメータスウィープによるアルゴリズムでは、想定される範囲でのパラメータリストを作成し、すべてのパラメータリストを実行する。このとき、パラメータの種類、パラメータに与える値の数が増えると、パラメータの組み合わせ数が指数的に増加する。

本章では、シミュレーション実施者の意思決定によって指数的に増加するパラメータの組み合わせによることなく最適解を探索する方法として、対話的動作に基づく託児所配置問題解決システムを提案した。提案する託児所配置問題解決システムでは、随時出力されるタスクの結果を確認しながら、最適解をもつパラメータリストを優先的に実行する特徴を有する。

最後に、対話的動作に基づく託児所配置問題解決システムのプロトタイプシステムを構築し、対話的動作による解探索時間と最適解探索の効果と政策立案の現場へ適用するための課題について述べた。最適解探索の実験では、適切な計算結果の提示方法を採用し、かつ被験者の対話的動作を可能にすることで、最適解の探索が短期間でできるようになることがわかった。プロトタイプシステムでは対話的動作による最適解探索の効果を確認することが主眼であった。政策立案の現場に適用するためには、グラフィカルなユーザインターフェイスによるパラメータリスト作成支援ツールと複数のタスク管理システムに対応したタスク投入モジュールの実装が必要であることを示した。

第8章

本研究で得られた知見

問題解決システムを用いたシミュレーションはいまや学術研究や製品開発に携わる方々の重要な解析ツールである。解くべき課題が複雑になるほどシミュレーションによる問題解決の重要性が増す。問題解決システムの構築において、解くべき課題の複雑さによらず、かつ、シミュレーション実施者の立場に立った環境を実現することが必要であるといっても過言ではない。

本研究では、高スループット計算向けグリッドにおける問題解決システムの構築に焦点をあて、シミュレーション実施者の意思決定に着目したシステム構築の考え方として、タスク実行支援フレームタスクを提案した。そして、提案するフレームタスクをタンパク質立体構造予測問題およびマルチエージェントシミュレーションによる託児所配置問題に適用した。本章では、タスク実行支援フレームワークにおける知見をまとめる。

8.1. フレームワークの有用性に関する知見

本論文では、シミュレーションの現場で行われている問題解決のプロセスに注目し、シミュレーション実施者の意思決定による操作を取り入れることで、シミュレーション結果が改善するとの仮説を立てた。この仮説に基づいたタスク実行支援フレームワークを提案するとともに、タンパク質立体構造予測問題およびマルチエージェントシミュレーションによる託児所配置問題に適用した。本節では、タンパク質立体構造予測およびマルチエージェントシミュレーションによる託児所配置問題におけるタスク実行支援フレームワークの有用性について述べる。

8.1.1 パラメータ設定ミスに対する早期対応

タンパク質立体構造予測の現場では、天然構造に近いタンパク質の立体構造を導くことが目標であり、研究者らは立体構造予測手法の改善に力を注いでいる。

タンパク質立体構造予測では、複数の予測手法を組み合わせて１つの立体構造を導くこと、計算結果を確認しながら各予測手法での再計算の判断、各手法でのパラメータ変更の判断が改善動作として必要になる。ところが現状は、複数の計算手法を有機的に実行することは可能であっても、シミュレーションの再実行やパラメータ変更の判断とその改善動作までは不可能であった。

そこで、タスク実行支援フレームワークを適用し、再計算やパラメータ変更の判断および改善動作を可能にした。タンパク質立体構造予測のように、タスク間に依存関係をもつようなシミュレーションの場合、タスクフロー差分実行機構によって任意のタスク位置からシミュレーションの再実行を実現できるため、シミュレーション実施者によるシミュレーション結果の継続的な改善動作を可能にした。

これにより、シミュレーション実施者がシミュレーションの再実行の際に負担であった修正タスクフローデータの作成動作、タスクフロー再実行依頼動作が、タスクのパラメータ修正のみとなり、シミュレーション実施者の作業量を減らした。また、シミュレーション再実行前に得たタスクの実行結果が再利用できるようになった。

8.1.2 最適解探索における解候補の早期発見

マルチエージェントシミュレーションによる託児所配置問題では、職場の位置、エージェントの初期条件から託児所の最適な位置を見つけることが目標である。これに対して、従来は託児所の位置を毎回変化させたシミュレーションを行うか、想定されるすべての託児所位置におけるシミュレーションを行うかのいずれかであった。マルチエージェントシミュレーションでは、パラメータの組が指数的に増加することから、適切な託児所の位置を見つけ出すためには、シミュレーション実施者のシミュレーション動作の負担を強いる、もしくはすべてのシミュレーションが終るまで待たなければならない、という問題があった。

そこで、提案したタスク実行支援フレームワークを用いて託児所配置問題解決システムを構築した。随時出力される計算結果を確認しながら、解候補をもつであろうパラメータの組を優先実行できるようにした。その結果、評価値が最大となるパラメータの組を短期間で得られるようになった。

8.2. フレームワークの適用範囲に関する知見

8.2.1 高スループット計算の場合

本論文で提案したタスク実行支援フレームワークは高スループット計算の例として、タンパク質立体構造予測およびマルチエージェントシミュレーションによる託児所配置問題に適用した。

タンパク質立体構造予測では NF ターゲットのシミュレーションが大量タスクの実行を必要とする。また、マルチエージェントシミュレーションによる託児所配置問題の場合、託児所位置による評価値の算出に大量タスクの実行を必要とする。いずれの場合も、大量のタスクを実行するシミュレーションであるため、大量タスクの実行を必要とするシミュレーションには適用可能である。

勿論、問題解決内容にあわせてカスタマイズが必要である。タンパク質立体構造予測の場合、複数の解析手法を有機的に組み合わせることが必要であるため、タスクフローによるシミュレーションプログラムの制御が必要であった。また、マルチエージェントシミュレーションによる託児所配置問題の場合、プログラムが必要とするパラメータの数に合わせてパラメータ変更用操作インターフェイスを実装したこと、計算結果のよい順に並べて表示する機能を結果出力インターフェイスに実装したことが挙げられる。

8.2.2 高スループット計算以外の場合

シミュレーション技法は高スループット計算以外にもある。例えば、分子の振舞いを特定することで分子場の問題を解く多体問題、航空機等の物体まわりの流体解析で用いられる有限差分法のように、複数のプロセッサで並列実行可能ではあるが計算結果を互いに同期をとらなければならないシミュレーション技法が考えられる。このようなシミュレーション技法では、計算が完了することを前提として計算プログラムを実行する。長時間かけて計算したのにも関わらず、計算結果を確認している段階でパラメータの設定ミスに気づき、計算結果が台無しになってしまうケースが少なくない。

本研究では適用事例として取り扱わなかったが、このようなシミュレーション技法の場合でも、提案するタスク実行支援フレームワークを用いることで、計算結果を台無しにする回数を低減できる。この手のシミュレーション技法では、アニメーションによって計算結果の可視化が積極的に行われている。アニメーション化のために計算結果はタイムステップ毎に出力される点に着目し、タイムステップ毎に得られる結果を途中結果として利用す

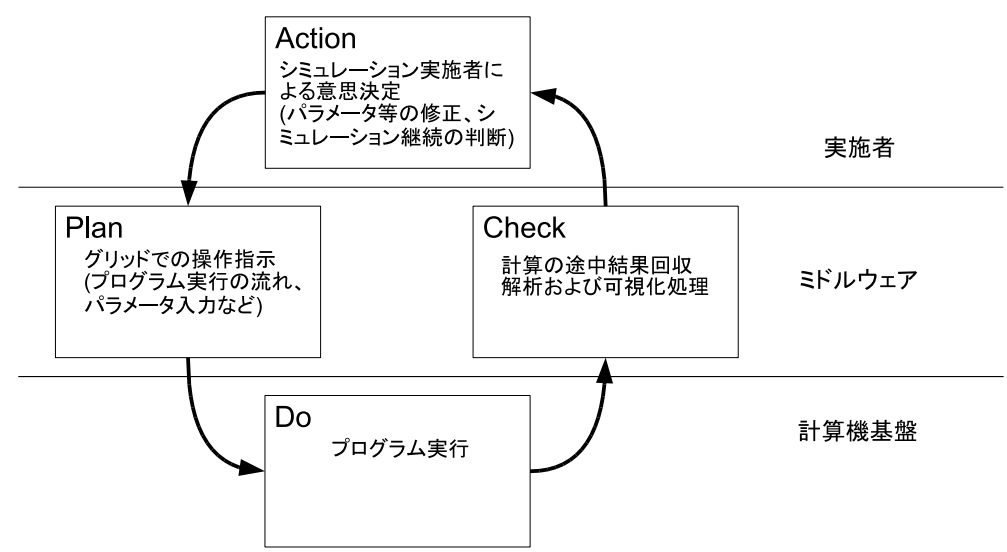


図 8.1　PDCA サイクルとシミュレーション実施者との関係

る。そして、シミュレーション実施者に途中結果とパラメータ一覧を提示するとともに、シミュレーションの実行制御を行う仕組みを提供することが必要になる。

また、シミュレーションプログラムにチェックポイント処理を組み込み、計算結果の保護およびシミュレーション実施者へのシグナルとして用いるとより効果的である。

8.3. 問題解決システム構築に関する知見

8.3.1 PDCA サイクルに基づく問題解決環境の実現

シミュレーション実施者は、例えば、最適解を探索するといった、ある目標に対して継続的な動作を行うことが要求される。そのためには、シミュレーション実施者の意思決定に基づいてシミュレーションを実施することが必要になる。

本論文では、品質管理に関する国際規格である ISO9000 において継続的改善のモデルとされている PDCA サイクルに基づいてシステム構築を行った。この結果、シミュレーション実施者と問題解決環境との関係は図 8.1 のようになり、「設定された目標に向かっているかを監視すること」、「必要に応じて是正措置をとること」が繰り返しできるようになった。

8.3.2 既存ミドルウェアにおける操作コマンドの隠蔽化

既存グリッドミドルウェアは数多く存在する。グリッドミドルウェアそれぞれが特徴をもっている。また、それぞれ異なる操作体系であり、操作体系が統一されていない。

グリッド技術による問題解決システムの構築において、現状は、シミュレーション実施者にミドルウェアの操作体系を理解してもらうことが先決だった。

それに対し、本論文で提案したタスク実行支援フレームワークは、パラメータ変更と大量タスク実行の支援に注目し、プログラム実行モジュールにおいて既存ミドルウェアの操作コマンドを隠蔽した。これにより、シミュレーション実施者はシミュレーションに専念できるようになった。

8.3.3 シミュレーション実施者の意思決定行動とユーザインターフェイスの実装

タスク実行支援フレームワークでは、シミュレーション実施者の意思決定行動はユーザインターフェイスを通じて行われる。シミュレーション実施者の意思決定によるシミュレーションの実行は、ヒューリスティックスなシミュレーション実行と読みとることができる。

ヒューリスティックスなシミュレーションの実行では、シミュレーション実施者の能力を短期間利用することで最適な結果を得る方針である。シミュレーション実施者が判断を誤ると最適な結果が得られる保証が失われる。よって、計算結果を出力するインターフェイスは解くべき問題の特徴を汲むことが大切である。

タスクの依存関係があるシミュレーションのパラメータ変更

タンパク質立体構造予測 (6 章) のような依存関係をもつタスク実行のシミュレーションでは、各タスクに設定されるパラメータの値を変更してシミュレーション結果を改善する。

このとき、ユーザインターフェイスで各タスクのパラメータ変更を可能にするだけではなく、グリッドミドルウェアにおいて依存関係を考慮したタスク実行をサポートすることで、シミュレーション実施者の問題解決システムへの操作回数を緩和できる。

大量タスク実行シミュレーションにおけるパラメータの値選択

マルチエージェントシミュレーションによる託児所配置問題 (7 章) では、シミュレーション実施者の意思決定に基づいて適切なパラメータの値を選び出す。

このような場合、意思決定を支援する方法としてはパラメータ変更ユーザインターフェイスよりも結果出力ユーザインターフェイスを充実するほうがよい。7 章で示したように、スコアの高い順に並べた実行結果を提示することで最適なパラメータの値を探す労力が緩和できること、最適な結果を得る保証が生まれる、といった利点がある。

8.3.4 意思決定動作の記録とその利用

本論文で提案するタスク実行支援フレームワークでは、計算結果の精度を高めること、および短期間で最適解を発見することを中心にシミュレーション実施者の意思決定を積極的に利用する。

本論文て取り上げたタンパク質立体構造予測において、タンパク質立体構造予測の研究者たちは立体構造予測の自動化に向けた取り組みに注目している。タンパク質立体構造予測の研究者らは自動化への知見を得るために、どのようなパラメータを入力し、どのような結果が得られたかを体系化したいと考えている。また、託児所配置問題解決システムで取り上げたマルチエージェントシミュレーションは経済学、商学、心理学といった社会科学分野の研究者に注目されている。社会科学分野の研究は、人間の活動に関わる研究が多いため、理論の正しさを証明する実験は容易に実施できない。そのため、シミュレーションによる模擬実験は現時点で唯一の実験的なアプローチである。これらの置かれている状況から考えると、計算機によるシミュレーションを実施することの重要性は増すばかりである。

一般的に、問題解決のためのシミュレーションは繰り返し行われる。シミュレーション実施者はシミュレーションの試行動作と意思決定をノートに記録する原始的な方法や、作業用のコンピュータに電子化情報として記録する方法を採用し、研究成果などに役立てている。これらの方法では、試行動作の数が多くなるほど、記録することに限界が出てくる。人間は曖昧な記憶を頼りに試行動作を繰り返す場合が往々にしてある。問題解決環境に意思決定の経過を記録することは、研究開発において新たな知見を得るため第一歩である。特に、シミュレーションが失敗したときの知見は、シミュレーションの方法論を見直すきっかけになるだろう。畑村は文献 [68, 69] で、失敗例を新たな知見に活かすためのポイントについて触れている。具体的には、失敗に対して前向きに取り組むことで新たな創造が生ま

れること、仮想演習の数が多いほど新たな創造が生まれやすいと説いている。

本論文で提案したタスク実行支援フレームワークは、意思決定動作の記録は考慮に入れていない。しかし、システム状況監視モジュールにあるタスク情報格納データベースの管理方法を改良することで、意思決定動作の記録とその利用は可能になる。

第9章

結論

本論文は高スループット計算向けグリッドにおける問題解決のプロセスを重視した問題解決システムの構築方法についてタスク実行支援の観点から論じた。

学術研究や製品開発における競争は一層激しさを増している。これらに携わる研究者や技術者は高度な情報技術を駆使した問題解決システムを研究開発のツールとして使っている。問題解決システムにおけるシミュレーション時間の短縮は、シミュレーション実施者にとって研究開発競争に遅れをとってはなるまいとの切実な思いであり、問題解決システムを構築する側の課題である。また、シミュレーションは結果を出すだけではなく、問題解決の糸口を見い出すためにも行われる。このとき、パラメータを変更してタスク実行するといった試行を幾度と繰り返す。

従来、問題解決システムにおけるシミュレーション時間の短縮は、主として基盤技術である計算機システムの設備増強によってなされていたが、複雑かつ大規模なシミュレーションの要求から、その手法もいまや限界に達している。

シミュレーション実施者の意思決定はシミュレーション結果の改善およびシミュレーション時間短縮のために継続的に行われている。本論文では、シミュレーション実施者の意思決定に注目し、継続的改善の行動パターンである PDCA サイクルに基づいたタスク実行支援フレームワークを提案した。

タスク実行支援フレームワークは計算機環境にシミュレーションを依頼するだけの従来方式とは異なり、シミュレーション結果の確認とタスク再実行の操作を連動させた。タスク実行支援フレームワークでは、よりシミュレーションのための問題解決作業に時間を費やせるような工夫として、シミュレーション実施者のタスク実行操作に関わる部分の作業

を軽減させる。具体的にはタスクフロー差分実行機構によってシミュレーション実施者はタスクフローの情報修正だけでタスクの再実行を可能にした。また、大量タスク実行におけるタスク優先実行機構によって、パラメータリストの実行順序を柔軟に変更できるようにした。これらの機構を用いることで、パラメータ修正によるシミュレーション結果の早期対応を実現し、パラメータ誤入力によるシミュレーション結果の修正が可能になる。また、大量タスク実行にかかる時間を短縮することができ、短期間で複数の問題を解けるようになる。

提案するタスク実行支援フレームワークをタンパク質立体構造予測とマルチエージェントシミュレーションによる託児所配置問題に適用し、フレームワークの有用性について検討した。タンパク質立体構造予測では、タスクフローに基づくシミュレーション環境を構築し、シミュレーション実施者の操作とシミュレーション結果の改善について確認した。シミュレーションの改善結果から、シミュレーション実施者の意思決定を可能にすることで適切な結果を導き出せることが確認した。また、マルチエージェントシミュレーションによる託児所配置問題では、大量タスク実行を可能にする環境を構築し、シミュレーション実施者の継続的改善による処理時間の短縮について確認した。シミュレーション実施者の意思決定を可能にすることで、すべてのパラメータが実行されるまでにかかる時間よりも短期間でかつ、ふさわしい結果を得られることを確認した。これらの適用事例から、本研究は問題解決システムの構築において、タスク実行支援フレームワークの有用性を確認するとともに、シミュレーションの実施において問題解決のプロセスを重視すべきであることを示した。

本論文は、タスク実行の観点からデスクトップ計算機の負荷を考慮に入れたノード選抜機構について議論し、高スループット計算による問題解決システムを構築する際の方法論として有益な情報を提供した。計算資源のひとつであるプロセッサはますます増える一方であるが、プロセッサの処理能力が有効利用されていない。本論文では大量タスク実行の高速化のためにはデスクトップ計算機を利用することが望ましいと考え、デスクトップ計算機を利用する上での問題点とタスク実行をするために必要なノード選抜について議論した。

自然科学や工学の分野から利用がはじまったコンピュータ・シミュレーションは、経済や政治といった社会科学の分野に広がっている。このことから、シミュレーション実施者の視点にたった問題解決システム構築の方法論はますます重要になる。本研究で得られた知見をもとに、シミュレーション実施者がより使いやすいとされる問題解決システムの構築方法について研究を進めていく。

付 録

A.1. 大量タスク実行用スクリプト

大量タスク実行用スクリプトでは、各パラメータのタスク実行および放棄を容易にするために、一定値以上のタスクを投入しない工夫を施した。具体的には、ローカルのタスク管理システムに対して一定の時間 (サンプリング時間) 間隔で保留状態のタスク数を確認する。そして、保留状態のタスク数が一定値以下になったらタスクを投入する。

ここでは、ローカルのタスク管理システムとして Condor を用いた場合の大量タスク実行用スクリプトを示す。

```
#!/usr/bin/perl

$progname=$ARGV[0];     # プログラム名
$paramIniFile=$ARGV[1]; # パラメータリストファイル名
$sleeptime = 20;         # サンプリング時間
$max_lines = 12;         # Condor にタスクを投入する際の最大値
$submitcounterFile = "/var/log/submitcounter.dat";
$allCountFile="/var/log/suballcount.ini";
$startlines = 0;

open(OIn, $allCountFile) or die;
@readdataCount = <OIn>;
close(OIn);
@lines_count = split(/\n/,$readdataCount[0]);
$allCount=$lines_count[0];

open(OIn, $paramIniFile) or die;
@paramlistdata = <OIn>;
close(OIn);
open(OIn, $submitcounterFile) or die;
@countdata = <OIn>;
close(OIn);
@lines_count = split(/\n/,$countdata[0]);
$i=$lines_count[0];
```

```
while($#paramlistdata >= $i) {
    # タスク実行状況の確認
    $str = &checktask;
    @lines_tmp = split(/\n/,$str);
    $lines = 0;
    $lines = $#lines_tmp;

    # タスクバッファに空きが生じたら新たなタスクを投入する
    if($lines < $max_lines) {
        $j=$i+1;
        $tmp_char = $readdata[$i];
        @get_filename = split(/\s/,$tmp_char);

        # タスク実行スクリプトの生成およびタスク実行
        if($startlines < $j) {
            # パラメータを付与したタスク実行
            &tasksubmit;

            # 実行したタスクのパラメータを出力する
            open(FOut, ">>./submitdat.bat");
            print FOut "$paramlistdata[$i]";
            close(FOut);

            $i++;
            open(OOut, ">./submitcounter.dat") or die;
            print OOut "$i";
            close(OOut);

            $allCount++;
            open(OOut, ">$allCountFile") or die;
            print OOut "$allCount";
            close(OOut);
            sleep $sleeptime;
        }
        else {
            print "already submit a job [$i]\n";
            $i++;
            open(OOut, ">./submitcounter.dat") or die;
            print OOut "$i";
            close(OOut);
        }
    }
    else {
        sleep $sleeptime;
    }
}
```

```
# パラメータを付与したタスク実行のサブルーチン (Condor の場合)
sub tasksubmit{
    open(FOut, ">./calcv4-$allCount.cmd");
    print FOut "UNIVERSE = vanilla\n";
    print FOut "Executable = $progname\n";
    print FOut "Log=calcv4-$i.log\n";
    print FOut "Output = calcv4-$i.out\n";
    print FOut "Error = calcv4-$i.err\n";
    print FOut "\n";
    print FOut "Arguments = $paramlistdata[$i]\n";
    print FOut "\n";
    print FOut "should_transfer_files = YES \n";
    print FOut "when_to_transfer_output = ON_EXIT\n";
    print FOut "\n";
    print FOut "transfer_input_files = AGE.txt, C18.txt, ED.txt,
                                        HI.txt, parameter.txt, WORK.txt \n";
    print FOut "transfer_output_files = $get_filename[16]\n";
    print FOut "\n";
    print FOut "Queue\n";
    print FOut "\n";
    close(FOut);

    $submit_command = "/usr/local/condor/bin/condor_submit
                                                    ./calcv4-$allCount.cmd\n";
    $system_message = system($submit_command);
}

# ローカルのタスク管理システムに投入されているタスクの数を確認する
sub checktask {
    $taskstr = `/usr/local/condor/bin/condor_q | /bin/grep calcv`;
    return $taskstr;
}
```

ローカルのタスク管理システムがOpenPBSの場合、tasksubmitとchecktaskはそれぞれ以下のようになる。

```
sub tasksubmit {
    open(FOut, ">./calcv4-$allCount.sh");
    print FOut "#!/bin/sh\n";
    print FOut "./$progname $paramlistdata[$i]\n";
    close(FOut);

    $submit_command = "/usr/local/openpbs/bin/qsub ./calcv4-$allCount.cmd\n";
    $system_message = system($submit_command);
}
```

```
sub checktask {
    $taskstr = `/usr/local/openpbs/bin/qstat | /bin/grep calcv4`;
    return $taskstr;
}
```

謝 辞

博士論文をまとめるにあたり、研究面では多くの方々のお力添えにより実現できたこと、また、精神面でも多くの方々に支えられていることを感じました。ここに感謝の意を表します。

奈良先端科学技術大学院大学 情報科学研究科インターネット・アーキテクチャ講座 砂原 秀樹 教授には長期にわたり多大なるご指導を賜わりました。本研究の機会を与えていただき、研究の方針や内容についてご指導いただいたことは研究活動を行う上で非常に有益でした。砂原教授に出会えたからこそグリッドの研究を遂行することができました。ありがとうございました。

奈良先端科学技術大学院大学 情報科学研究科インターネット工学講座 山口 英 教授には本論文を取りまとめる上で有益なご教示を賜わりました。また、研究に対する姿勢を教えていただきました。ありがとうございました。

奈良先端科学技術大学院大学 情報科学研究科インターネット・アーキテクチャ講座 藤川 和利 助教授には本研究を遂行する上での議論、研究方針、論文執筆に関する指導を賜わりました。また、バイオグリッド・プロジェクトでの活動に参加させていただくきっかけをつくっていただきました。ありがとうございました。

奈良先端科学技術大学院大学 情報科学研究科インターネット・アーキテクチャ講座 垣内 正年 助手、和泉 順子 助手、情報コミュニケーション講座 河合 栄治 助手、インターネット工学講座 森島 直人 助手には研究を遂行する上での議論に付き合っていただきました。特に、垣内助手と和泉助手は一学年上の先輩として、公私に渡り相談に乗っていただきました。ありがとうございました。

奈良先端科学技術大学院大学 情報科学研究科インターネット・アーキテクチャ講座事務補佐員 呂 悠妃 さん、旧事務補佐員 田坂 (旧姓 能勢) 佳苗 さんには、私の研究活動を影から支えていただきました。学位申請などの事務手続きが円滑に進められたのもおふたりの力があったからと感じております。ありがとうございました。

奈良先端科学技術大学院大学 情報科学研究科インターネット・アーキテクチャ講座の学生の皆さん、OB や OG の皆さんには私の研究において有益な助言を賜わりました。タス

クフローシステムの研究において、共同開発者である増田 慎吾 君にはシステムの設計、実装、性能評価実験等において、いろいろと協力していただきました。心から感謝いたします。

タンパク質立体構造予測へのグリッド環境への適用については、神戸大学 理学部化学科 高田 彰二 助教授をはじめ、同学科 朴 聖俊 研究員、金 文珍 研究員、東京大学 大学院農学生命科学研究科 古田 忠臣 助手 (旧 神戸大学 理学部化学科) の御協力を賜わりました。タンパク質立体構造予測のシミュレーション技法について無知だった私に対して丁寧に教えていただきました。また、シミュレーション実施者の立場からグリッド環境構築に関する様々な助言を賜わりました。ありがとうございました。

本研究の一部はバイオグリッド・プロジェクトにおける活動の一環として研究活動を遂行させていただきました。バイオグリッド・プロジェクトにおいて研究活動の機会を与えていただいたとともに、有益な助言をしていただいた大阪大学 サイバーメディアセンター 下條 真司 教授をはじめ、大阪大学 大学院 情報科学研究科 伊達 進 助教授、およびバイオグリッド・プロジェクトの関係者の皆様に心より感謝いたします。

また、バイオグリッド・プロジェクトの一研究グループとして SC2003 および SC2004 にタンパク質立体構造予測システムの出展をする機会を与えていただきました。システム出展に関して、HTC グループのメンバーとして協力していただいた有限会社キャトルアイ・サイエンス代表取締役上島 豊さん (旧 日本原子力研究所 光量子科学研究センター) にはグリッドによるタンパク質立体構造予測システム構築に関して様々な意見、提案をいただきました。ありがとうございました。

本論文は関西大学 経済・政治研究所 政策グリッドコンピューティング実験センターに勤めながら執筆することになりました。私の上司である関西大学 経済・政治研究所 政策グリッドコンピューティング実験センター センター長 村田 忠彦 助教授には本論文をまとめる上で応援をしていただきました。政策グリッドコンピューティング実験センターでの研究活動を軌道に載せる時期なのにも関わらず、学位論文執筆を最優先するようにとの御指示をいただきました。また、本論文の一部であるマルチエージェントシミュレーションによる託児所配置問題のグリッドへの適用に関する研究の機会を与えていただきました。遅々として進まなかった論文執筆状況に対して励ましの言葉をいただけたこと、暖かく見守っていただいたことを感謝しております。応援していただきありがとうございました。

また、勤務先の同僚である関西大学 文学研究科 松原 光也さん、関西大学 社会学研究科 曹 陽さん、関西大学 工学研究科 番匠 大輔さんには私の学位論文執筆に際して激励を賜わりました。論文の構成で悩んでいたとき、研究分野が全く違うにも関わらず親身になっ

て相談にのっていただきました。また、様々な助言を賜わりました。ありがとうございました。

最後に、父 進、母 雅子、弟 晃、弟 晋の理解なくしては本論文をまとめ上げることはできませんでした。私が研究者として歩んでいきたいと両親や兄弟に告げたとき、それに反対することなく、応援してくれたことは私にとって何よりの支援でした。また、幾度とあった挫折を克服できたのは、「何があっても博士号だけは取得してほしい」という両親の願いに励まされたと思っています。家族に感謝し、論文の結びとします。

研究業績一覧

学術論文

1. **蟻川 浩**, 増田 慎吾, 古田 忠臣, 金 文珍, 朴 聖俊, 高田 彰二, 藤川 和利, 砂原 秀樹, “対話的動作を考慮したタンパク質立体構造予測システム”, 情報処理学会論文誌:コンピューティングシステム Vol.46, No.SIG 12 (ACS 11), pp.407-419, August 2005.

国際会議 (査読あり)

1. **Hiroshi Arikawa**, Kazutoshi Fujikawa and Hideki Sunahara, “A Node Selection Mechanism Based on the Node Usage Pattern on Campus Grid”, Proceedings of 2003 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM ’03) Vol 1 pp.217-220, August 2003

2. Kazutoshi Fujikawa, Wenzhen Jin, Sung-Joon Park, Tadaomi Furuta, Shoji Takada, **Hiroshi Arikawa**, Susumu Date and Shinji Shimojo, “Applying a Grid Technology to Protein Structure Predictor “Rokky””, HealthGrid 2005, March 2005

国内研究会等 (査読なし)

1. **蟻川 浩**, 河合 栄治, 砂原 秀樹, “グローバルコンピューティングに適した Java 実行環境の研究”, 第 42 回プログラミングシンポジウム報告集, pp.143-150, January 2001, (第 42 回 プログラミングシンポジウム)

2. **蟻川 浩**, 砂原 秀樹, “Campus Grid における負荷状況を考慮したノードの選抜方法”, 情報処理学会研究報告, 2002-HPC-91, pp.167-172, August 2002.

3. 村口 貴信, **蟻川 浩**, 田村 坦之, “バーチャルマシン技術を適用したデスクトップグリッド環境”, 第 49 回 システム制御情報学会研究発表講演会 (SCI’05), pp.391-392, May

2005

4. 村田 忠彦, 松本 優美, **蟻川 浩**, 山口 正聡, 小橋 博道, 門岡 良昌, “グリッドによるマルチエージェントシミュレーションの分散処理”, 計算工学講演会論文集, Vol.11, No.2, pp.545-548, June 2006

5. **蟻川 浩**, 濱田 英信, 村田 忠彦, 小橋 博道, 門岡 良昌, “実計算時間を考慮したデスクトップグリッドのためのタスクスケジューリング”, 計算工学講演会論文集, Vol.11, No.2, pp.549-552, June 2006

参 考 文 献

[1] 伊達 進, 奥村 利幸, 秋山 豊和, 下條 真司, 松田 秀雄, 中村 春木. バイオグリッドプロジェクト「スーパーコンピュータネットワークの構築」. 情報処理, Vol. 44, No. 6, pp. 601–607, June 2003.

[2] Satoshi Matsuoka, Shinji Shimojo, Mutsumi Aoyagi, Satoshi Sekiguchi, Hitohide Usami, and Kenichi Miura. Japanese Computational Grid Research Project: NAREGI. *Proceedings of the IEEE*, Vol. 93, No. 3, pp. 522–533, March 2005.

[3] Ian Foster and Carl Kesselman, editor. *The Grid: Blueprint for a New Computing Infrastructure Second Edition*. Morgan Kaufmann, 2004.

[4] Ian Foster. What is the Grid? A Three Point Checklist. GRIDToday, July 2002.

[5] 武宮 博, 首藤 一幸, 田中 良夫, 関口 智嗣. Grid 環境上における気象予報シミュレーションシステムの構築. 情報処理学会論文誌: コンピューティングシステム, Vol. 44, No. SIG11 (ACS 3), pp. 23–33, 2003.

[6] Hironori Hiraishi and Fumio Mizoguchi. A Cellular Telephone-Based Application for Skin-Grading to Support Cosmetic Sales. *AI Magazine*, Vol. 25, No. 3, pp. 17–26, 2004.

[7] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastracture Toolkit. *International Journal of Supercomputer Applications*, Vol. 11, No. 2, pp. 115–128, 1997.

[8] Larry Smarr and Charles E. Catlett. Metacomputing. *Communication of the ACM*, Vol. 35, No. 6, June 1992.

[9] T. DeFanti, I. Foster, M. Papka, R. Stevens and T. Kuhfuss. Overview of the I-WAY: Wide Area Visual Supercomputing. *International Journal of Supercomputer Applications*, Vol. 10, No. 2, 1996.

[10] Nirav H. Kapadia and José A. B. Fortes. PUNCH: An Architecture for Web-enabled wide-area network-computing. *Cluster Computing*, No. 2, pp. 153–164, 1999.

[11] J. Almond and D. Snelling. UNICORE: uniform access to supercomputing as an element of electronic commerce. *Future Generation Computer Systems*, Vol. 15, pp. 539–548, 1999.

[12] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky and Dan Werthimer. SETI@home - An Experiment in Public-Resource Computing. *Communication of the ACM*, Vol. 45, No. 11, November 2002.

[13] David P. Anderson. BOINC:A System for Public-Resource Computing and Storage. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)* , pp. 4–10, 2004.

[14] Michael Litzkow, Miron Livny and Matt Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pp. 104–111, June 1988.

[15] Franck Cappello, Samir Djilali, Gilles Fedak, Thomas Herault, Frédéric Magniette, Vincent Néri, and Oleg Lodygensky. Computing on large-scale distributed systems: XtermWeb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems*, Vol. 21, pp. 417–437, 2005.

[16] R. Henderson. Job Scheduling under the Portable Batch System. *LNCS 949*, pp. 279–294, 1995.

[17] 石川 裕, 堀 敦史, 手塚 宏史. RWCP におけるクラスタ開発記. 情報処理, Vol. 39, No. 11, pp. 1095–1100, 1998.

[18] Thomas L. Sterling, John Salmon, Donald J. Becker, Daniel F. Savarse. *How to Build a Beowulf: A Guide to the Implementation and Appliation of PC Clusters*. MIT Press, 1999.

[19] 北野 宏明 (監訳), 奥乃 博, 諸橋 峰雄, 京田 耕司, 中臺 一博. PC クラスタ構築法. 産業図書, 2001.

[20] Hiroshi Tezuka, Atsushi Hori, Yutaka Ishikawa, and Mitsuhisa Sato. PM: An Operating System Coordinated High Performance Communication Library. In Peter Sloot and Bob Hertzberger, editors, *High-Performance Computing and Networking, volume 1225 of Lecture Notes in Computer Science*, pp. 707–717. Springer-Verlag, 1997.

[21] Atsushi Hori, Hiroshi Tezuka, and Yutaka Ishikawa. Overhead Analysis of Preemptive Gang Scheduling. In D.G. Feitelson and L.Rudolph, editors, *IPPS'98 Workshop on Job Scheduling Strategies for Parallel Processing, volume 1291 of Lecture Notes in Computer Science*, pp. 262–276. Springer-Verlag, 1998.

[22] Jason Novotny. The Grid Portal Development Kit. *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13–15, pp. 1129–1144, 2002.

[23] Gregor von Laszewski, Jarek Gawor, Peter Lane, Nell Rehn and Mike Russell. Features of the Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp. 1045–1056, 2002.

[24] Toyotaro Suzumura, Hidemoto Nakada, Satoshi Matsuoka, and Henri Casanova. GridSpeed: A Web-based Grid Portal Generation Server. In *Proceedings of HPC Asia 2004*, pp. 26–33, July 2004.

[25] 白砂 哲, 鈴村 豊太郎, 中田 秀基, 松岡 聡. アプリケーションのインストール、データの配布、更新をサポートするグリッドポータル構築ツールキット (PCT4G) の開発. 情報処理学会研究報告 2003-HPC-95, pp. 173–178, 2003.

[26] R. Buyya and S. Venugopal. The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report. In *1st IEEE International Workshop on Grid Economics and Business Models, GECON 2004*, pp. 19–36, April 2004.

[27] J. Yu and R. Buyya. The Novel Architecture for Realizing Grid Workflow using Tuple Spaces. In *5th IEEE/ACM International Workshop on Grid Computing*, 2004.

[28] Andreas Hoheisel and Uwe Der. An XML-Based Framework for Loosely Coupled Applications on Grid Environments. *LNCS 2657*, pp. 245–254, 2003.

[29] Tadao Murata. Petri nets: Propaties, analysis and applications. *Proceedings of the IEEE*, Vol. 77, No. 4, pp. 554–580, April 1989.

[30] T. Tanncnbaum, D. Wright, K. Miller, and M. Livny. Condor - A Distributed Job Scheduler. *The MIT Press*, 2002.

[31] 中田 秀基, 高木 浩光, 松岡 聡, 長嶋 雲兵, 佐藤 三久, 関口 智嗣. Ninf による広域分散並列計算. 情報処理学会論文誌, Vol. 39, No. 6, pp. 1818–1826, 1998.

[32] Yoshio Tanaka, Hidemoto Nakada, Satoshi Sekiguchi, Toyotaro Suzumura, and Satoshi Matsuoka. Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing. *Journal of Grid Computing*, Vol. 1, No. 1, pp. 41–51, 2003.

[33] 武宮 博, 中田 秀基, 田中 良夫, 関口 智嗣. Ninf-G2: 大規模 Grid 環境での利用に即した高機能, 高性能 GridRPC システムの実装と評価. 情報処理学会論文誌: コンピューティングシステム, Vol. 45, No. SIG 11 (ACS 7), pp. 144–159, 2004.

[34] 中田 秀基, 田中 良夫, 松岡 聡, 関口 智嗣. 耐故障性を重視した RPC システム Ninf-C の設計と実装. 情報処理学会論文誌: コンピューティングシステム, Vol. 45, No. SIG 11 (ACS 7), pp. 160–170, 2004.

[35] Henri Casanova and Jack Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. In *Proceedings of Super Computing '96*, 1996.

[36] 佐藤 三久, 朴 泰祐, 高橋 大介. OmniRPC: グリッド環境での並列プログラミングのための Grid RPC システム. 情報処理学会論文誌: コンピューティングシステム, Vol. 44, No. SIG 11 (ACS 3), pp. 34–45, 2003.

[37] 文部科学省研究委託事業 超高速コンピュータ網形成プロジェクト NAREGI シンポジウム 2005, February 2005.

[38] 文部科学省研究委託事業 超高速コンピュータ網形成プロジェクト NAREGI シンポジウム 2006, February 2006.

[39] Ian Taylor, Matthew Shields, Ian Wang, and Omer Rana. Triana Applications within Grid Computing and Peer to Peer Environments. *Journal of Grid Computing*, Vol. 1, pp. 199–217, 2003.

[40] *Business Process Execution Language for Web Services*, May 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

[41] 安里彰, 門岡良昌. 計算機リソースを有効活用するグリッドミドルウェア:CyberGRIP. *FUJITSU*, Vol. 55, No. 2, pp. 146–151, March 2004.

[42] 中村武雄, 山下智規. 製品開発を加速する CAD-Grid システム. *FUJITSU*, Vol. 55, No. 2, pp. 121–126, March 2004.

[43] B. Uk, M. Taufer, T. Stricker, G. Settanni and A. Cavalli. Implementation and Characterization of Protein Folding on a Desktop Computational Grid Is CHARMM a suitable candidate for the United Device MetaProcessor? In *Proceedings of the International Parallel and Distributed Processing Symposium*. IEEE, 2003.

[44] Arun Krishnan. GridBLAST: a Globus-based high-throughput implementation of BLAST in a Grid computing framework. *Concurrencty and Computation: Practice and Experience*, Vol. 17, No. 13, pp. 1607–1623, 2005.

[45] 池上 努, 武宮 博, 長島 雲兵, 田中 良夫, 関口 智嗣. Grid:広域分散並列処理環境での高精度分子シミュレーション. 情報処理学会論文誌: コンピューティングシステム, Vol. 44, No. SIG 11 (ACS 3), pp. 14–22, 2003.

[46] 鑓水 訟氏. PC グリッドの現在と展望. 情報処理, Vol. 44, No. 6, June 2003.

[47] IBM 金融グリッド・マネージャー. http://www-06.ibm.com/grid/jp/customer/nli.shtml, 2004.

[48] 鵜飼康東. 研究最前線 グリッドコンピューティングの研究最前線 1. 政策グリッドコンピューティング実験. 電子情報通信学会論文誌 情報・システムソサイエティ誌, Vol. 10, No. 3, pp. 4–5, November 2005.

[49] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, Vol. 15, No. 3, 2001.

[50] 上島 豊, 松田 行正, 岡本 隆. XML/Web 技術を利用したシミュレーション支援システム. 計算工学講演会論文集, 第 11 巻, pp. 525–526, June 2006.

[51] 中田 秀基, 松岡 聡, 関口 智嗣. Java による階層型グリッド環境 jojo の設計と実装. 情報処理学会論文誌: コンピューティングシステム, Vol. 44, No. SIG11 (ACS 3), pp. 46–56, 2003.

[52] 武田 伸悟, 伊達 進, 下條 真司. GSI-SFS: グリッドのためのシングルサインオン機能を有するセキュアファイルシステム. 情報処理学会論文誌: コンピューティングシステム, Vol. 45, No. SIG6 (ACS 6), pp. 223–233, 2004.

[53] Derrick Kondo, Michela Taufer, Charles L. Brooks III, Henri Casanova and Andrew A. Chien. Characterizing and Evaluating Desktop Grids: An Empirical Study. In *Proceegins of the International Parallel and Distributed Processing Symposium (IPDPS '04)*, April 2004.

[54] D. Baker and A. Sali. Protein Structure Prediction and Structural Genomics. *Science*, 2001.

[55] W. Jin, T. Furuta, S.J. Park, N. Koga, Y. Fujitsuka , G. Chikenji and S. Takada. ROKKY: structure prediction server that integrates PDB-BLAST, 3D-Jury, and the SimFold fragment assembly simulator. In *CASP6 Abstracts*, pp. 128–129, 2004.

[56] S.F. Altschul, F. Stephen, L.M. Thomas, A.S. Alejadro, Z. Jinghui, Z. Zheng, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, Vol. 25, pp. 3389–3402, 1997.

[57] J. Shi, T.L. Blundell, and K. Mizuguchi. FUGUE: sequence-structure homology recognition using enviroment-specific substitution tables and structure-dependent gap penalties. *Journal of Molecular Biology*, pp. 248–257, 2001.

[58] S. Takada. Protein Folding Simulation With Solvent-Induced Force Field: Folding Pathway Ensenble of Three-Helix-Bundle Proteins. *Proteins:Structure,Function and Genetics*, No. 42, pp. 85–98, 2001.

[59] Y. Fujitsuka, S. Takada, Z. A.Luthey-Schulten, and P.G. Wolynes. Optimizing Physical Energy Functions for Protein Folding. *Proteins: Structure, Function and Bioinformatics*, Vol. 54, pp. 88–103, 2004.

[60] A. Zemla. LGA - A Method for Finding 3D Similarities in Protein Structures. *Nucleic Acids Research*, Vol. 31, No. 13, pp. 3370–3374, 2003.

[61] 車谷浩一. マルチエージェント社会シミュレーション展望. システム/制御/情報, Vol. 46, No. 9, pp. 518–523, 2002.

[62] 鵜飼康東, 村田忠彦, 北埜裕子. 既婚女性の労働供給における政策グリッドコンピューティング実験. 関西大学経済論集, Vol. 55, No. 3, pp. 421–443, December 2005.

[63] Noriyuki Tanida and Masatoshi Murakami. Agent Based Modelling for Pension Systems(1) - Fundermental Design. *The Economic Review of Kansai University*, Vol. 54, No. 2, pp. 22–36, 2004.

[64] 平成 17 年度版 労働経済の分析 (労働経済白書). 厚生労働省.

[65] 「少子化の要因と少子化社会に関する研究会」報告書. 財務省財務総合政策研究所, 2005.

[66] Tadahiko Murata, Hiroko Kitano, Tomoharu Nakajima, and Hisao Ishibuchi. Application of a Multi-Agent Model with Pioneers and Followers to a Day Care Center Allocation Problems. In *International Conference on Intelligent Technologies 2003*, pp. 179–186, 2003.

[67] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transaction on Modeling and Computer Simulations*, Vol. 8, No. 1, pp. 3–30, January 1998.

[68] 畑村洋太郎. 失敗学のすすめ. 講談社, 2000.

[69] 畑村洋太郎. 決定版 失敗学の法則. 文春文庫, 2005.