卒業論文

題目

タスクスケジューリングの可視化によるスケジューリング情報の取得

指導教員

大野和彦 先生

2012年

三重大学 工学部 情報工学科
計算機アーキテクチャ研究室

武田竜一(408826)

内容梗概

近年，気象予測シミュレーションや構造解析など複雑な物理計算を要する分野において高い計算能力が求められている．しかし，単一プロセッサにおける性能向上は限界に達しており大規模並列処理への需要が高まっている．

このような並列処理において効率よく計算を行うために，実行単位であるタスクをどのコンピュータにどの順番で割り当てるかを決めるタスクスケジューリングが重要となってくる．その際，スケジューリング手法の問題点などを発見することなどが必要となってくる．

しかし，現状のスケジューリング情報はテキスト形式となっておりスケジューリング環境やスケジューリング実行状況などが把握しづらく，問題点などの発見が困難なものとなっている．

そこで本研究では，タスクスケジューリングに関する情報の視覚化手法を提案し実装を行った．

Abstract

In recent years, high computing power is demanded in the field which required complex physical calculationsuch as weather forecasting and structural analysis simulation. But Performance enhancement in the uniprocessor has reached the limit, so demand for large-scale parallel processing has been growing.

To calculate efficiently such as parallel processing, Task Scheduling which decide to assign the task to the computer becomes important. Then, to discover the problem of Scheduling method becomes necessary.

However, the current scheduling information has become a text format , and such as scheduling execution status and scheduling environment are difficult to understand, and to discover the problem is difficult.

In this study, a method of visualize information about the task scheduling has been proposed and implemented.

目 次

図 目 次

表 目 次

1　はじめに

近年，気象予測シミュレーションや構造解析など複雑な物理計算を要する分野においてPFLOPS以上の高い計算能力が求められている．しかし，単一プロセッサにおける性能向上は限界に達しており百万規模のプロセッサを用いた大規模並列処理への需要が高まっている．

そのため現在，本研究室ではメガスケール規模での環境下において並列処理を実現するためのタスク並列スクリプト言語 MegaScript [1] を開発している．MegaScriptは百万台規模の並列処理を実現するために開発されているプログラミング言語である．このような並列処理を行う際，実行単位であるタスクをどのコンピュータにどの順番で割り当てるかを決めるタスクスケジューリングが重要となってくる．そのため現在様々なスケジューリング手法が研究されている [2] [3].

スケジューリング手法の問題点，例えばアルゴリズムの一部の欠点などを発見することは重要であるが，即座にそのようなことを行うのは難しい．そこでユーザがスケジューリングの疑問点，例えばホストの実行の偏りなどを指摘し，その情報を元にスケジューリングの問題点などを指摘するようなことが求められる．

現在，本研究室にはMegaScriptなどに対応したシミュレータ [4] が開

発されているが，そのスケジューリング情報はテキスト形式となっている．従ってスケジューリングに関する疑問点などの指摘も難しい．そこで本研究ではスケジューリングに関する情報の可視化手法を提案し実装を行った．これにより，スケジューリングに関する疑問点などをテキスト形式よりも簡単に発見できるようになっている．

本文の構成は以下のようになっている．2 章で MegaScript の概要やタスクスケジューリングに関する説明をし，現状の MegaScript のスケジューリング情報とその問題点について述べる．3 章でその問題点を解決するための視覚化手法を提案する．4 章で実装するにあたっての環境や実装方法を実装する．5 章で本研究をまとめている．

2　背景

2.1　MegaScript

2.1.1　概要

MegaScript は数百万規模の並列処理を実現することを目的として開発されているプログラム言語である．MegaScript は逐次プログラムや規模の小さな並列プログラムのような独立性の高い外部プログラムを計算タスクとして扱い，これらのタスクを並列に実行させることで大規模な処理を実現するものである．

また，各タスク間の通信にはストリームと呼ばれる論理的な通信路を介してお互いにメッセージ交換を行う．計算のメインとなる各タスクは外部プログラムとして用意するため，MegaScript プログラム内では主に並列実行に関する制御情報を記述する．MegaScript ではこれらを解析し，スケジューリング結果に従って各タスクを指定された計算機で実行する．タスクのスケジューリングは MegaScript が自動で行うので，ユーザはメガスケール規模でのプログラミングを少ない負担で実行できるという特徴を有している．

2.1.2 動作環境

MegaScript は，プログラミングの容易性・記述性からベース言語としてオブジェクト指向スクリプト言語 Ruby を用いている．そのため，MegaScript 処理系は Ruby を拡張する形で実装されている．よって，MegaScript の動作には Ruby の動作する環境に加え，MegaScript ランタイムのインストールされた環境が必要となる．

2.1.3 動作モデル

MegaScript のプログラミングモデルは，並列動作させる逐次・並列プログラムを定義し，それらタスクを並列プロセスとして複数のタスクとストリームの接続に関して記述することで大規模な並列実行システムを可能としている．本稿では MegaScript によるワークフローをタスクネットワークと呼ぶ．タスクネットワークの例を図 2.1 に示す．タスクは外部プログラムのことでイメージとしては図 2.1 の丸 A，B，C のようなものである．ストリームはイメージとしては図 2.1 の円柱のようなものであり入力端と出力端を持ち，入力端に繋いだタスクの標準出力の内容が出力端に繋いだタスクの標準入力へと流し込まれる．イメージとしては図 2.1 の矢印に相当する．

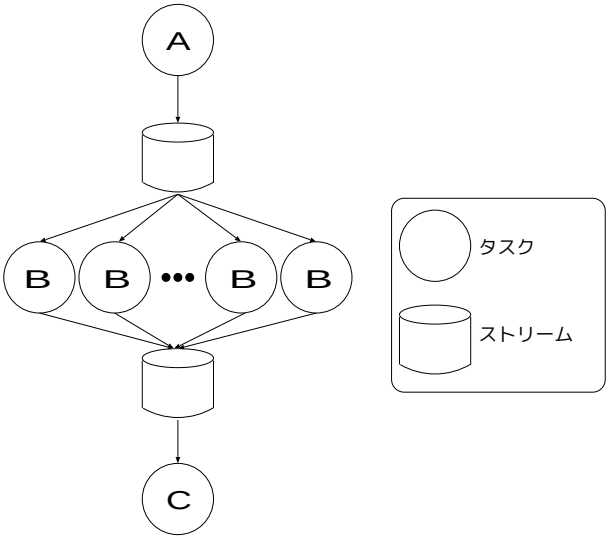

図 2.1: MegaScript のタスクネットワーク

ここで具体例として，未知の病原菌の DNA 配列群を既知の病原菌のデータと比較を行うタスクネットワークを図 2.2 に示す．まず未知の病原菌の DNA 配列群に対し conv でデータを生成し，BLAST [5] を使い既知のデータベースとの一致検索を行い，その結果を利用して ClustalW [6] で樹形図を得るといった動作を表している．

2.2 タスクスケジューリング

タスクスケジューリングとはタスク（仕事）をどのようにホスト（コンピュータ）に割り当てるかということであり， 大規模なワークフロー型の並列処理の実行効率に大きな影響を与える．このため，HEFT [2] や

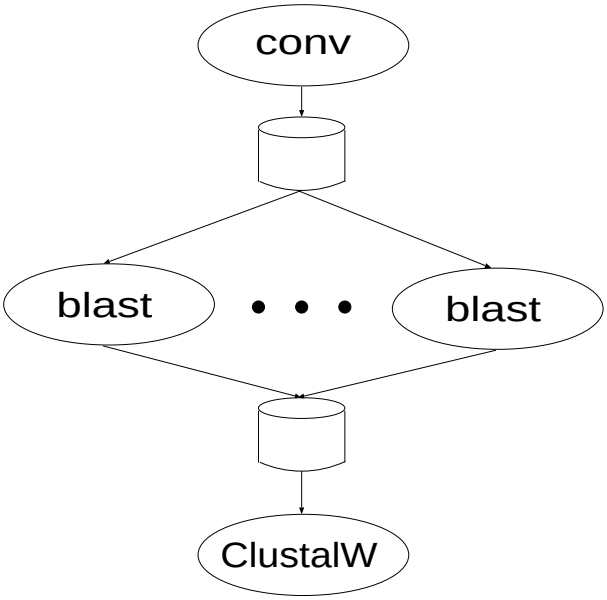

図 2.2: タスクネットワークの具体例

適応型スケジューリング [3] のようなタスク間の依存関係を考慮した様々なスケジューリング手法が考案されている．

タスクスケジューリングの例を以下で説明する．図 2.3 は今回の説明に使用するホストの例である．ホストは 3 台使用しそれぞれに 1，2，3 と名前をつけ，各矢印はそれぞれが通信可能であることを表している．各ホストにはそれぞれ演算性能が存在し，各ホスト間の通信性能も異なる．図 2.3 は今回の説明に使用するタスクネットワークの例である．タスク名をそれぞれ A，B，C とし，各矢印はそれぞれの依存関係を表している．各タスクにはそれぞれ計算にかかる計算量が存在する．

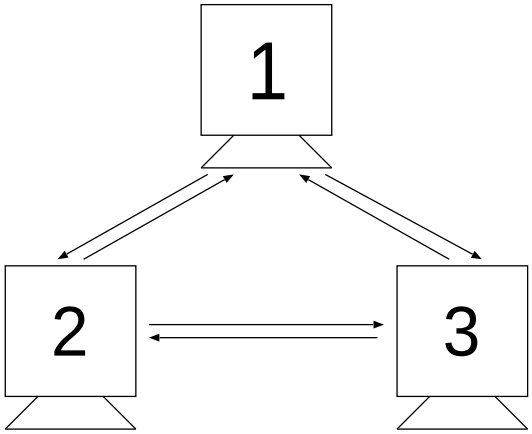

図 2.3: ホストの例

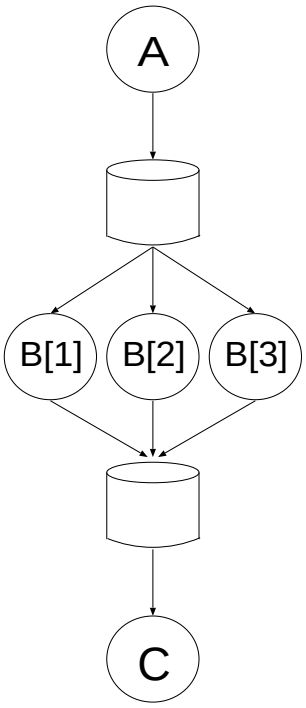

図 2.4: タスクネットワークの例

図 2.5 はタスクスケジューリングの例である．縦軸は各ホストを表し，横軸は実行時刻を表している．各線はそれぞれ，立ち上がっている間はそのホストがタスクを実行しており，立ち下がっている間は休止していることを表している．立ち上がりの表記のなかに記述されている文字列は実行中のタスク名を表している．

ホスト 1 を見ると開始時刻にタスク A が実行されているが，ホスト 2，ホスト 3 は最初のタスクの実行に時間がかかっている．これはタスク A とタスク B の間に依存関係がありタスク B はタスク A の実行結果が必要

であるためタスク A の実行が終了した後でしか実行できないためである．ホスト 2 のタスク C も同様に各タスク B の実行結果が必要であるため，それぞれのホストで実行されているタスク B が終了した後でしか実行できない．それゆえ，各タスク B の実行結果をホスト 2 へ送信する必要がある．このように各ホスト間の通信性能も全体の実行時間に大きな影響を与える．

以上のことから，良いスケジューリング結果を得るためには，ホストの演算性能だけでなく各ホスト間の通信性能やタスクの依存関係も考慮する必要がある．

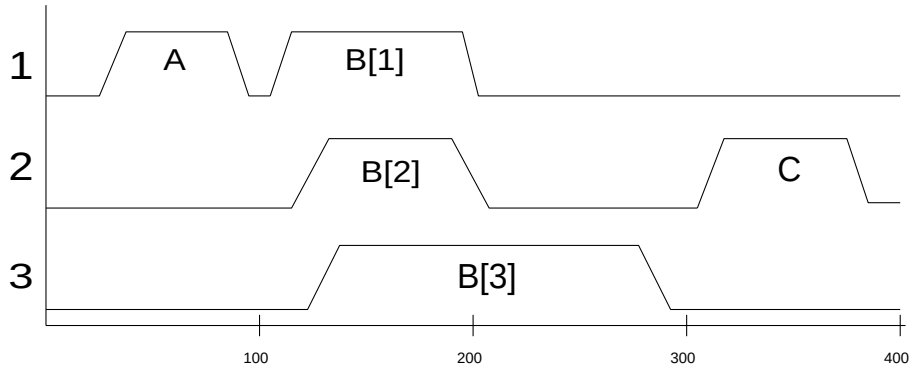

図 2.5: タスクスケジューリングの例

2.3　スケジューリングの実行情報

現在，本研究室にはタスクスケジューリング結果に従ってワークフローの実行を行うシミュレータ [4] が存在し，そのシミュレーションに関する情報は図 2.6 のようにテキスト形式で出力され，ホストで実行されたタスクの名前，その開始時刻や終了時刻などのスケジューリングに関する情報が記録されている．また，タスクやホストに関する情報も同様にテキスト形式でそれぞれ独立したファイルに記述されている．

```
initialized simulator...
loaded task file...
loaded host file...
[EventEngine] Initializing engine...
[EventEngine] Starting engine...
[Engine] time=0 RecvStartTaskCommand is Executed : Task task10
[Engine] time=0 RecvStartTaskCommand is Executed : Task task19
[Engine] time=0 RecvStartTaskCommand is Executed : Task task24[3]
[Engine] time=0 RecvStartTaskCommand is Executed : Task task24[1]
[Engine] time=0 RecvStartTaskCommand is Executed : Task task11
[Engine] time=0 RecvStartTaskCommand is Executed : Task task24[6]
[Engine] time=0 RecvStartTaskCommand is Executed : Task task24[8]
[Engine] time=0 RecvStartTaskCommand is Executed : Task task7
[Engine] time=0 RecvStartTaskCommand is Executed : Task task24[5]
[Engine] time=0 RecvStartTaskCommand is Executed : Task task40
[Engine] time=0 RecvStartTaskCommand is Executed : Task task12
[Engine] time=0 RecvStartTaskCommand is Executed : Task task13
[Engine] time=0 RecvStartTaskCommand is Executed : Task task6
[Engine] time=0 RecvStartTaskCommand is Executed : Task task24[9]
[Engine] time=0 RecvStartTaskCommand is Executed : Task task14
[Engine] time=0 RecvStartTaskCommand is Executed : Task task28
[Engine] time=0 RecvStartTaskCommand is Executed : Task task25
[Engine] time=0 RecvStartTaskCommand is Executed : Task task43
```

図 2.6: 実行情報

2.4 スケジューリング情報の可視化

2.3節で説明したように，シミュレーション結果のログがテキスト形式であるため，タスクスケジューリング手法の問題箇所の特定が難しい．そこで，ユーザにログをグラフィカルに見せることで問題箇所をより簡単に発見することができる．また，ワークフローとホストの情報を同時に表示することで，それらの情報を関連づけて把握することも容易になる．

例えば，あるホストを指定するとそのホスト上で実行されるタスクが表示され，その配置が正しいかを容易に把握することが可能となる．さらに挙動が不安定な箇所をユーザが指摘することで問題箇所を自動で絞り込めれば，タスクスケジューリング手法の開発効率はより高くなる．

そこで本研究では，タスクスケジューリングのシミュレーションログをグラフィカルに視覚化する手法を提案し実装した．これにより，シミュレーションの実行状況から問題箇所を容易に把握することができる．問題箇所の自動絞り込み機能を実装できればタスクスケジューリングの開発はより容易になるが，これは今後の課題である．

3　視覚化手法

本節では，タスクスケジューリングの情報を視覚化するにあたり以下の項目について考慮する．

- 視覚化ツールの画面構成
- タスク配列の描画方法
- ホスト情報の取得方法
- 実行状況の描画
- タスクの割当箇所の描画方法

3.1　視覚化ツールの画面構成

視覚化ツールの画面構成は図 3.7 のようになる．ウインドウを分割しそれぞれにワークフロー (図 3.7 右)，ホスト (図 3.7 中央)，ホストの詳細情報 (図 3.7 左) の描画を行う．これは，単一のウインドウで全ての情報を描画すると表示が煩雑になってしまうからである．

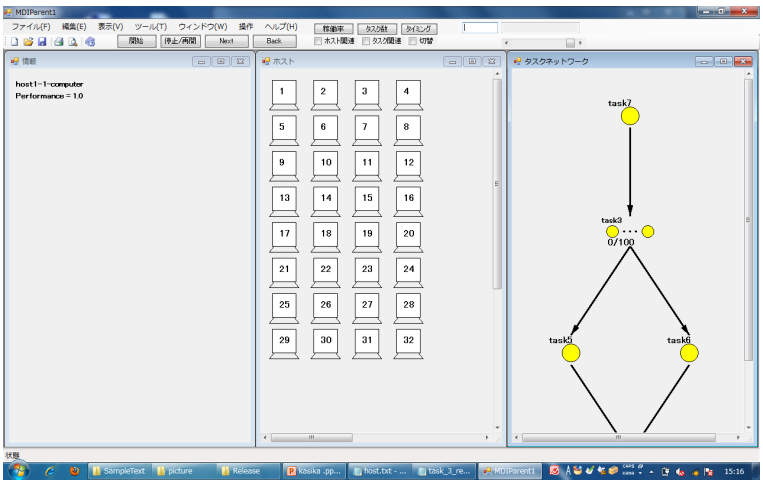

図 3.7: ファイル読込後の初期状態

3.2 タスク配列の描画方法

同じプログラムを異なる条件で多数実行するパラメータスイープと呼ばれるものがあり，通常大規模なタスク配列が用いられる．タスク配列は同じ種類のタスクの集合であり，MegaScript では同種のタスクを多数生成する場合を想定したタスク配列クラスを用意している．

通常のタスクの場合は表示に影響を与えることは少ないが，タスク配列の場合その配列のサイズが何千となることがある．従ってそれらすべてを表示させた場合，ワークフローの表示サイズが大きくなってしまいウインドウ内に収まりきらず，図 3.8 中央丸内のようになってしまい見にくい表示となってしまう．そこで，タスク配列に対する描画方法を提

案する．

タスクが配列の場合は，タスクを個別に描画する代わりにタスクが配列であることを示すコンパクトな表現を用いる．これよってワークフローの表示が小さくなり，全体像を把握しやすくなる．

しかしタスク配列をコンパクトな表示にした場合，タスク配列内の個々のタスクへアクセスすることが困難である．そこでマウスクリックによるタスク配列の展開・縮小をできるようにする．ワークフロー内のタスク配列の部分をクリックするとそのタスク配列だけを展開することができ，これによって必要なときだけ個々のタスクへアクセスすることができるようになる．

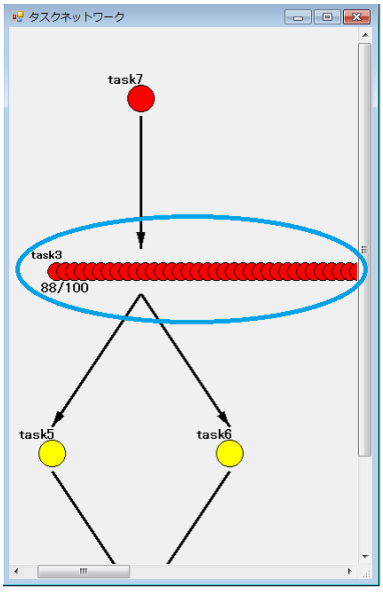

図 3.8: タスク配列の描画

3.3 ホスト情報の取得方法

タスクスケジューリングにおいて，スケジューリングの実行動作が意図したものとなっているか把握するために各ホストが実行したタスク等のホスト情報が必要である．そこで，ホスト情報取得の手法を提案する．まず，図 3.7 の中央ウインドウに各ホストを描画させる．そこで確認したいホストの描画部分をクリックすると，図 3.7 の左側のウインドウにホスト名， ホスト性能， ホストが実行したタスク名，そのタスクの実行開始時間と実行終了時間を表示させるようにする．これにより，ホストに関する情報を簡単に取得することができる．例えば図 3.9 は時刻 300 での 7 番目のホストの状況を表示させたものである．

3.4 実行状況の描画

シミュレーションの実行状況の可視化は，タスクスケジューリングの動作確認や問題点の発見において非常に重要である．効果的な視覚化手法を実装できれば，バグの規則性やホストの利用情報の偏り等を容易に発見することができる．そこで本研究で提案する実行状況の視覚化手法について説明する．

図 3.10 の右図のウインドウはワークフローを示しているが，各時刻の

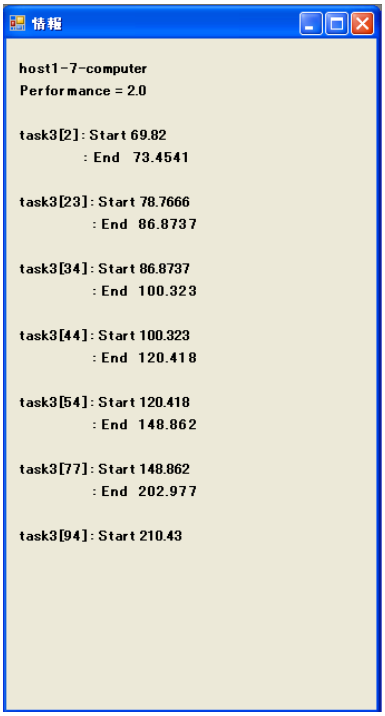

図 3.9: ホスト情報

各タスクの実行状況についてタスクに色付けをして描画する．色を用いることによって一見して全体像を把握することが可能であり，またアニメーションによる実行状況の変化も確認しやすいためである．ワークフローにおいてある時刻までに実行されたタスクとそうでないタスクを色分けして表示する．タスク配列に関しては，(実行完了タスク数/タスク配列) のサイズを求め，その値に応じて色の濃淡を変更する．これにより，シミュレーションの実行が進むにつれて徐々に色が変化していく．

図 3.10 の中図のウインドウはホストの利用状況を描画しているが，こ

れについてもワークフローの実行状況と同等の理由で色を用いる．各ホストを塗りつぶす色について，何かしらのタスクが実行されているホストと実行されていないホストを色分けして表示する．これにより，各ホストが効率良く利用されているかについて把握することが可能となる．

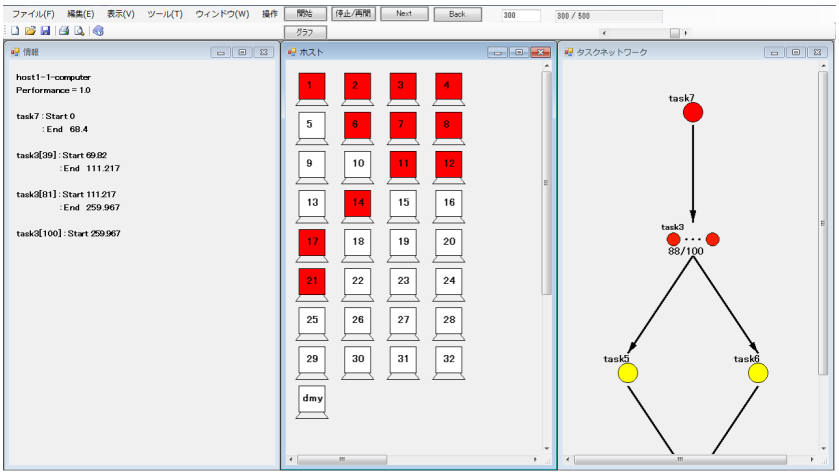

図 3.10: 動作状況

3.5 タスクの割当箇所の描画方法

3.3 節では各ホストにおいてどのタスクが実行されたかなどのホスト情報が取得できるようになっている．しかし，それらのタスクがタスクネットワーク上のどの位置に当たるかなどの情報は一見して把握することができない．そこでタスクネットワーク上における実行タスクの位置情報の描画手法を提案する．

各ホストをクリックするとそのホストで実行されたタスクに対応したタスクネットワーク上のタスクが点滅するようになっている．このように点滅させることで直感的にスケジューラによるタスクの割当位置が把握できるようになる．

4　実装

4.1　視覚化ツールの実装環境

このツールを利用する際に，利用環境が利用者により異なることがある．そのため利用環境になるべく依存せず動作する必要がある．そのためにはOSやCPUアーキテクチャなどの環境が異なっていても共通の動作が保証されることが望ましい．

そこで.NET Framework [7] について着目する．.NET Frameworkは Microsoft社が開発しているアプリケーション開発，実行環境のことである．.NET Frameworkは Common Language Infrastructure (CLI) と呼ばれるものをサポートしている．CLIは IL(Intermediate Language) と呼ばれる.NET Frameworkに対応した言語をコンパイルした際に生成される中間言語を実行するための環境である．このCLIの仕様にそった実装さえされていれば任意のプラットフォームで実行することができる．そこで現在，Microsoft 自身による Windows 上の CLI 実装である Common Language Runtime (CLR) が提供されている．一方，Linux版のCLIである mono [8] なども開発されている．

以上の内容から.NET Frameworkに沿った形で開発を行えば，様々な環境でソフトウェアを実行できる．

そこで本研究では.net Framework の開発用プログラミング言語である Visual Studio C# [9–11] を使用する.

4.2 タスクやホストに関する情報の表示

タスクの情報について，図 4.11 のようにタスクを表す丸の形を描画し，そのすぐそばにそのタスクの名前を描画する．タスクが配列の場合，図 4.11 の task3 のように配列を表すアイコンで描画し，タスク名の他に配列の要素数についても描画する．ホストの情報については別のウインドウに描画する．図 4.12 はその一例である．テキストファイルに記述されているホストの数だけ図 4.12 のようなアイコンを用いて描画し，そのアイコン内にホスト番号を表示させる.

4.3 大規模ワークフローの表示

タスクの配列表示をコンパクトなものにしても，大規模なワークフローの場合タスクのアイコン数が多くなり，ワークフロー全体が巨大なものになってしまうことがあり得る.

そこでワークフロー全体の大きさを調整できる調節メモリを用意する．ワークフローの描画が大きすぎてウインドウ内に入りきらない場合このメモリを調節することで表示を縮小させる．これによってウインドウを

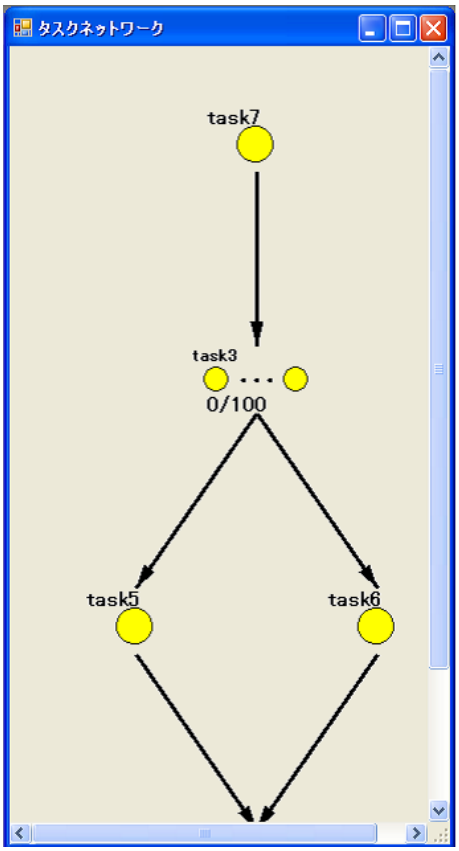

図 4.11: タスクネットワーク描画例

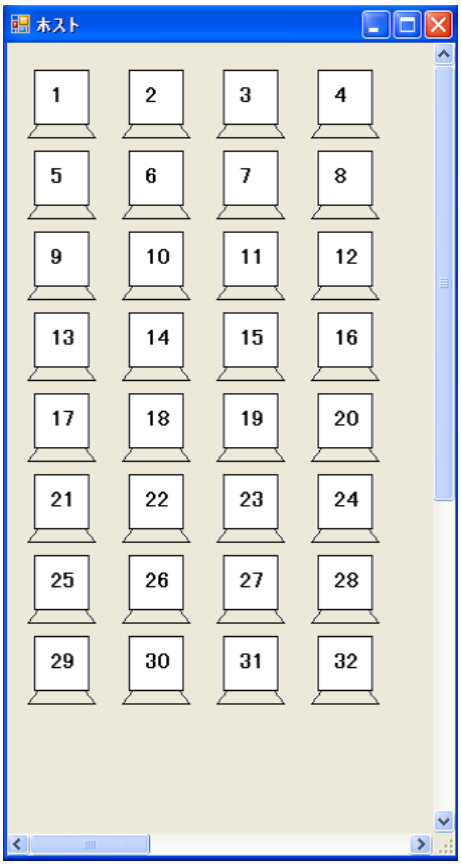

図 4.12: ホスト描画例

スクロールすることなく全体を把握することができる．図 4.13 はワークフロー縮小前の一例である．このように全体の表示が大きすぎてウインドウ内に入りきらない場合は丸枠で囲ってある調節メモリを変動させることで図 4.14 のようにワークフロー全体を縮小させることができる．調整については 1%毎に縮小・拡大できるようになっている．

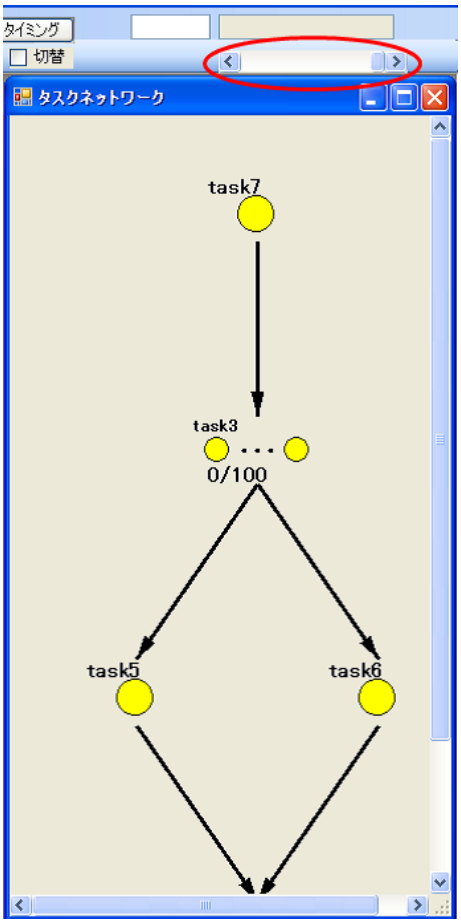

図 4.13: 縮小前

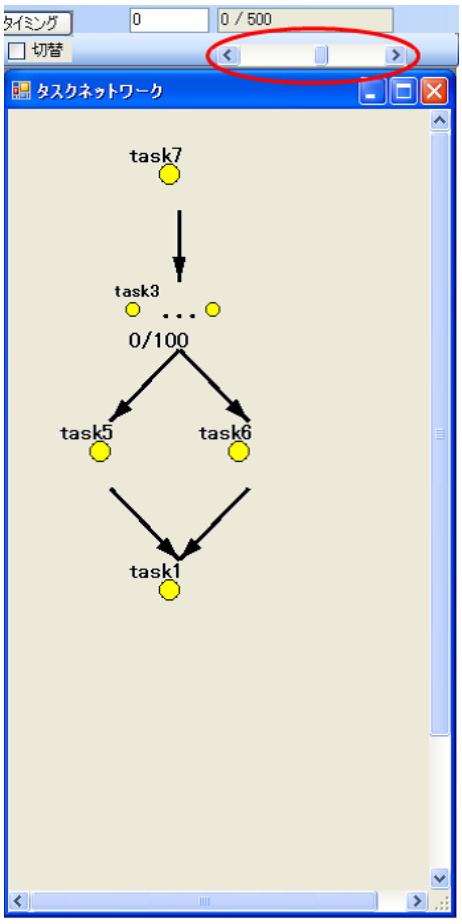

図 4.14: 縮小後

4.4　グラフによる情報表示

3.3 節のようにホストが実行したタスクについての詳細な情報を知ることができるが，このままでは逐一各ホストを確認しなければならず非常に煩わしい．そこで各ホストで実行されたタスクのタイミングチャートを表示させることができるようにする．図 4.15 は各ホストにおけるタスク実行時間間隔のホスト全体のタイミングチャートである．横軸は実行時間，縦軸はホスト名となっており，線が立ち上がっている間はホストがタスクを実行していて立ち下がっているときはタスクを実行していないことを示している．また，描画しているホストをクリックすると，それに対応するタイミングチャートの線の色が変化して表示されるようになっている．このようにしてホスト全体の時効時間間隔などが一見して把握できるようになっている．例えば，10 番目のホストを見ると最後のタスクの実行開始までにかなりの開きがあるというようなことが簡単に把握できるようになる．

また，3.4 節の実行状況の描画についてはホスト全体の大まかで直感的な実行状況はわかるが各ホストが実際にどの位動作していたかなどの詳細なホストの稼働率を知ることができない．そこで各ホストでの全体の実行時間に占める稼働時間の割合を見ることができるグラフを表示でき

るようにする．図 4.16 は各ホストにおける稼働率を表示したグラフである．横軸はホスト名，縦軸は実行時間となっておりこのようにグラフで表示させることでより正確な情報を取得できる．例えば，24 から 32 番目のホストの稼働時間の少なさが一目で把握できるようになる．

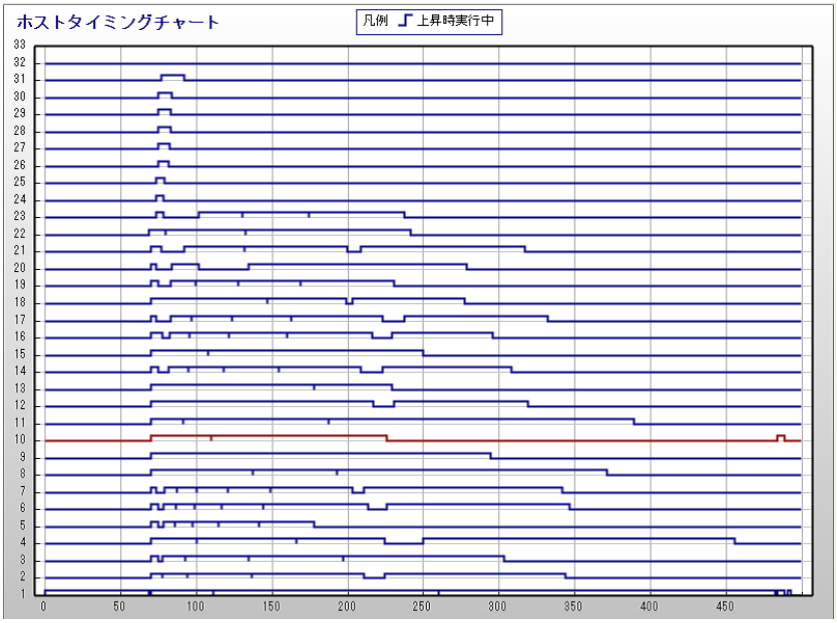

図 4.15: タイミングチャート

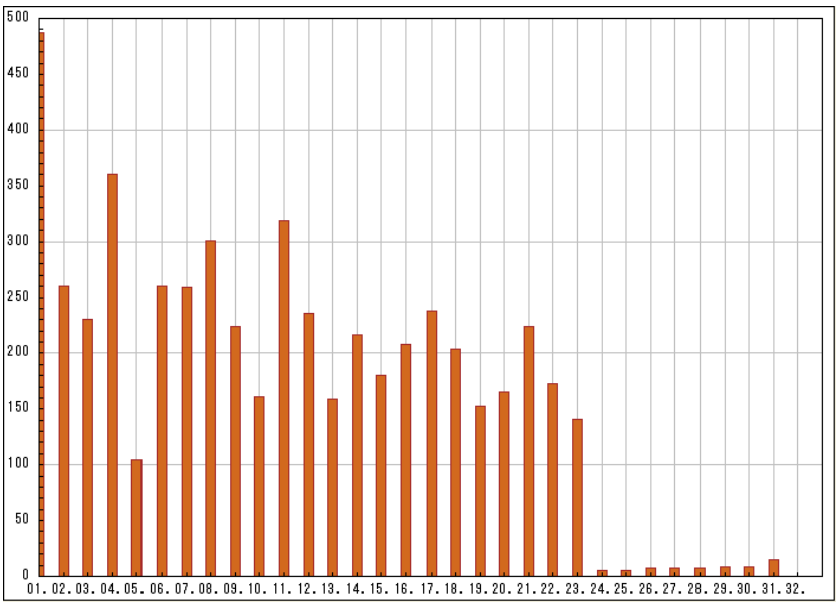

図 4.16: ホスト稼働率のグラフ

5　おわりに

本研究では，タスクスケジューリングの可視化手法の提案，実装を行った．その結果従来のテキスト形式の表記よりも，実行情報の全体像や動き等のスケジューリングに関する情報が簡単に取得できるようになった．

今後は，スケジューリングの動作に問題がありそうな箇所をユーザが指摘していくことで，スケジューリング手法の問題箇所を自動的に絞り込んでいく機能を実装していく必要がある．

謝辞

本研究を行うにあたり，御指導，御助言頂きました大野和彦講師，並びに多くの助言を頂きました近藤利夫教授，佐々木敬泰助教に深く感謝致します．また，様々な局面にてお世話になりました研究室の皆様にも心より感謝いたします.

参考文献

[1] 大塚保紀，深野佑公，西里一史，大野和彦，中島浩 タスク並列スクリプト言語 MegaScript の構想 先進的計算基盤システムシンポジウム SACSIS2003 pp.73-76 2003

[2] H.Topcuoglu, S.Hariri and Wu, M.-Y.: A high-performance mapping algorithm for heterogeneous computing system IPPS/SPDP Workshop on Het-erogeneous Computing pp. 3-14 1999

[3] 松本真樹，片野聡，佐々木敬泰，大野和彦，近藤利夫，中島浩 非均質環境における適応型スケジューリング手法の提案と評価 電子情報通信学会論文誌 Vol. J93-D, No. 6 2010

[4] 松本真樹，佐々木敬泰，大野和彦，近藤利夫，中島浩広域分散環境における大規模タスク群の挙動を求める高速シミュレータ情報処理学会 第72回全国大会6K-5 2010

[5] S.F. Altschul, W. Miller, E.W. Myers, D.J. Lipman Basic local alignment search tool J.Molecular Biolpgy vol.215 pp.403-410 1990

[6] D.J. Thompson, D.G. Higgeins, T.J. Gibson CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice Nucleic Acids Reseach vol.22 pp.4673-4680 1994

[7] Microsoft .NET のすべて http://msdn.microsoft.com/ja-jp/library/dd297763.aspx 2012年2月20日

[8] mono http://www.mono-project.com/ 2012年2月20日

[9] Visual C# デベロッパーセンター http://msdn.microsoft.com/ja-jp/vcsharp/aa336706 2012年2月20日

[10] 松下浩則，松下典子つくって覚えるC#入門株式会社アスキー・メディアワークス2010

[11] WisdomSoft http://wisdom.sakura.ne.jp/ 2012年2月20日