分散環境における記憶領域資源を考慮するタスクスケジューリング手法

小出 洋† 谷口慶介†

分散記憶管理が必須なアプリケーション・プログラムを実行する場合においても効率的にスケジューリングできる Grid コンピューティングのための新しいタスクスケジューリング手法を提案する．その基本的な考え方は，タスクスケジューラがクリティカルパス長を計算する際に分散記憶管理のためのコストを考慮し，総計算時間が最短になるようにスケジューリングすることである．また，分散テストベッドシステム上に Java Remote Method Invocation を用いて実装されたタスクスケジューリングシステムに提案手法の予備的な実装である Easy CP/MM を組み込み，分散プログラムをテスト実行して他の手法と比較した．その結果，Easy CP/MM がクリティカルパス長が比較的長い分散プログラムを効率的にスケジューリングできることも示す．

A Proposal of the Task Scheduling Method for a Grid Computing System Considering Information from the Distributed Memory Management System

Hiroshi Koide† and Keisuke Taniguchi†

A new task scheduling method for a Grid computing system which is effective even if the application programs require a distributed memory management is proposed in this paper. The basic idea is considering the cost for a distributed memory management when the task scheduling system calculates critical path lengths. Moreover, the authors built in Easy CP/MM, a preliminary implementation of the proposed method, in the task scheduling system which is implemented by using the Java Remote Method Invocation on the distributed testbed system. The authors show that Easy CP/MM effectively schedules distributed programs which have long critical paths as a result of the experiments on the environment.

1. はじめに

Grid コンピューティングは，PC，ワークステーション，高性能計算機，データベース等をネットワークで接続した Grid コンピューティング・システム上で，高速化や大規模データ処理が必要な場合に有効となるネットワークコンピューティング技術である．一般に Grid コンピューティング・システムは，複数の異なるアーキテクチャの計算機により構成され，多くの利用者により共有されるため，計算機やネットワークの負荷が時間的に変動する．このシステム上で適切な負荷分散を行うためには，計算機やネットワークの負荷変動を予測し，それらを考慮して，個々のタスクやジョブを適切な計算機に割り付けるスケジューリングシステムが必要になる．

† 九州工業大学大学院工学研究科
Department of Electrical, Electronic and Computer Engineering, Kyushu Institute of Technology

Grid コンピューティング・システムにおけるタスクスケジューリングに関する研究は，世界的に盛んに行なわれている．例えば，Grid コンピューティングの世界的な研究プロジェクトである Globus[1] で研究されている資源予約を行なうスケジューリング手法[2]，計算機の動的負荷情報を考慮することが可能な AppLeS[3]（Application Level Scheduling）等の方式，自動並列化コンパイラや利用者により与えられるタスクに関する静的情報（タスク間の依存関係や静的に見積もられた各タスクの実行時間）とプロセッサやネットワークの負荷に関する動的情報の両方を考慮して，Grid コンピューティング・システムにおける単一プログラムの実行時間最小化を目的とするメタスケジューリングに関する研究[4),5]，スケジューリングに必須となる資源の負荷を動的に予測することを目的とする研究も行なわれている（例えば，[6)~8)] 等）．

一方，Globus では，大規模な科学技術データを国際的な研究グループ間で共有することを目的とする

Data Grid に関する研究を行なっている．Data Grid 研究グループは，科学技術計算に直接関係する大規模データ以外に，それらのデータの内容やデータ構造，レプリカ（データ転送時間の短縮化等を目的としてネットワーク上に置かれた複製）の所在等のデータ管理のための情報であるメタデータが必要であると指摘し，大規模な科学技術データを共有する枠組を提案している[9]．メタデータは，主として XML で表現されており，はっきりとしたスキーマが定義できないという点で半構造データの分散データベースと捉えることができる．また，さまざまなネットワーク・プログラミングのための枠組として優れており，OS や計算機を問わず広く使用できるオブジェクト指向型言語である Java や Ruby が Grid コンピューティングに積極的に利用されつつある（例えば，[10)~12)] 等）．

今後，Data Grid のように半構造データを扱ったり，Java のようなオブジェクト指向型言語による Grid コンピューティングを行なう機会が増えると予想される．このとき，分散した記憶領域の管理，例えば，ネットワーク上に分散した計算に利用できないオブジェクトの占める記憶領域を再利用する分散ガーベジコレクション（以下，簡単のためガーベジコレクションを GC と略す）や 1 台の計算機では確保できないような記憶領域をネットワーク上の複数の計算機に分散して割り当てる分散アロケータのような機構が必須となる．このような分散した記憶領域の管理は，一般的には本来の計算を一時停止して行なう必要があり，記憶管理を行なう側に何らかの仕組みがない限り，いつ分散記憶管理のための作業が起動されるのか，どのくらい本来の処理を中断しなければならないのか予測困難であり，タスクスケジューリングに多大な影響を及ぼすことになる．

本論文では，分散記憶管理が必須となる Grid コンピューティングを行なう際にも，効率的なタスクスケジューリングを行なえるようにすることを目的として，分散記憶管理からの情報を考慮するタスクスケジューリング手法である CP/MM (Critical Path/Memory Management) の提案を行なう．その基本的な考え方は，タスクスケジューラがクリティカルパス長を計算する際に分散記憶管理のためのコストを考慮し，総計算時間が最短になるようにスケジューリングすることである．また，広域分散テストベッドシステム上に Java RMI[13)] (Remote Method Invocation，以下単に RMI と略す) を用いて実装された実験用タスクスケジューリングシステムに提案手法の予備的な実装（3.1 で説明する Easy CP/MM）を組み込み，分散プログラムをテスト実行して他の手法と比較した．その結果，Easy CP/MM はクリティカルパス長が比較的長い分散アプリケーションを効率的にスケジューリングできることを示す．

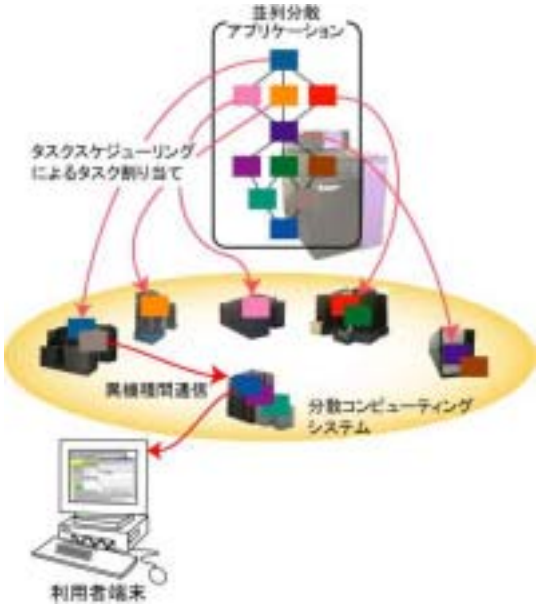

図 1　メタスケジューリング手法．
Fig. 1　The Metascheduling Method.

2. メタスケジューリング

本章では，提案するタスクスケジューリング手法の基本となるメタスケジューリング手法について簡単に説明する (**図 1**)．小出らは，Grid コンピューティング・システムで効率的なタスクスケジューリングを実現する問題に焦点を絞り，メタスケジューリング手法の開発・評価を行なっている．

2.1 逐次プログラムの自動並列分散化

メタスケジューリング手法では，その実行準備段階において，利用者が用意した逐次プログラムを以下のような手順により，分散プログラムに変換する必要がある．この変換は，並列化コンパイラを使用するか，利用者自身がプログラムを書き換えることにより行なう必要がある．

(1) ループやサブルーチン等の処理単位であるマクロタスク（粗粒度タスク，以下単にタスクと呼ぶ）に分解する．
(2) 各タスクの静的実行時間（他に負荷が無い場合の実行時間）を推定する．
(3) 各タスク間のデータおよび制御依存関係を表すマクロフローグラフ（MFG）を作成する．

(4) MFG に基づいてタスクを実行時に動的にスケジューリングするためのタスクスケジューリングコードを生成し，タスクの集合である並列化されたユーザプログラムに埋め込む．

例えば，[4),5)] におけるメタスケジューリングの実装では，並列化コンパイラ（早稲田大学 OSCAR マルチグレイン並列化コンパイラ[14)~16)] を使用し，逐次プログラムをループやサブルーチン等の処理単位であるタスクに分解し，タスク間の並列性を解析後，タスクを実行時に各計算機に割り当てるスケジューリングコードを，分散化されたユーザプログラムとともに生成している．この生成されたスケジューリングコードは，実行時に計算機やネットワークの負荷変動，マクロタスク間のデータおよび制御依存関係を考慮して，Grid コンピューティング・システム上の利用可能な各計算機に動的に割り付ける．

2.2 メタスケジューリング実行時の動作

メタスケジューリングで扱うタスクスケジューリング問題は，実行順序関係の制約がある n 個の処理時間の異なるタスクを，処理能力の異なる m 台のプロセッサで並列処理する際，プロセッサ間のデータ転送を含めた実行時間（スケジューリング長）を最小にするノンプリエンティブ（割り込み不可）なスケジュールを求める問題である．このスケジューリング問題は，強 NP 困難であることが知られているため，最適解を多項式時間で効率良く求めるアルゴリズムは知られていない．そこで最適解とは限らないが，経験上良い解を出すことができるヒューリスティクアルゴリズムを利用している．

メタスケジューリングでは，このヒューリスティックアルゴリズムとして，現在のところ CP/ETF/MISF for Metascheduling を使用している．これは，MFG において出口ノードから各タスクまでの最長パスが長いタスクほど，高い優先順位を与え，次に同じ条件のタスクが存在すれば直接の後続タスクが多いタスクを優先する CP/ETF/MISF (Critical Path/Earliest Task First/Most Immediate Successors First) を基本とし，計算機やネットワークの負荷が変動する異機種で構成された Grid コンピューティング・システムに適用できるように，実行時に現在の計算機やネットワークの負荷，それらの将来の予測値を用いて各タスクをどの計算機で行なうと最小時間で処理できるか実行時に再計算してスケジューリングするように変更したものである[4),5)]．

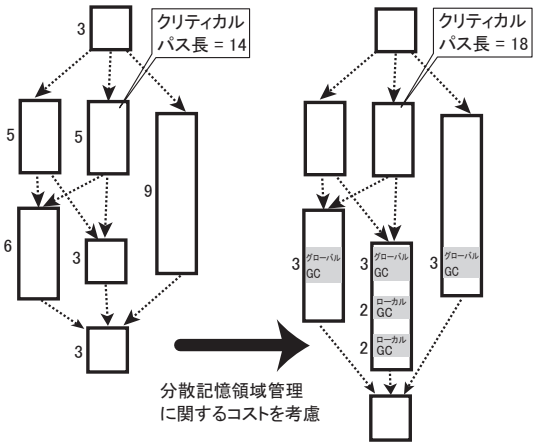

図 2 CP/MM におけるタスクスケジューリング．
Fig. 2 A Task Scheduling of CP/MM Method.

3. CP/MM

本論文で提案するタスクスケジューリング手法である CP/MM は，CP/ETF/ MISF for Metascheduling を基本として，記憶領域に関する情報も考慮するように変更したものである．その基本となる CP/ETF/MISF for Metascheduling では，以下の情報を利用してスケジューリングを行なっている．

1 各計算機の負荷とその予測値
2 ネットワークの負荷とその予測値
3 コンパイラや利用者が静的自身に見積もったタスクの実行時間や依存関係

CP/MM では，これらに付け加えて

4 分散記憶管理からの動的情報
  A 各計算機のヒープ領域の残容量
  B GC 予想所要時間
5 コンパイラや利用者からの各タスクのヒープ領域の消費量

を利用する．なお，**4** の **A** は，分散記憶領域管理が一般に保持している内部変数である．**4** の **B** は，過去に行なった GC の所要時間から容易に推測可能である．

CP/MM は，CP/ETF/MISF for Metascheduling におけるクリティカルパス長の計算を行なう段階において，分散記憶領域管理のためのコストが含まれるように，**5** を利用してヒープ領域の使用量を計算し，GC が起動されるタイミングとそれに要する時間を含めたクリティカルパス長を求める（**図 2**）．

例えば，**図 3** に示す 2 台の計算機 A，B を利用可能なとき，2 つのタスク x，y が実行可能状態であり，それらが実行されれば，プログラムが終了する場合，CP/ETF/MISF for Metascheduling では，x が A

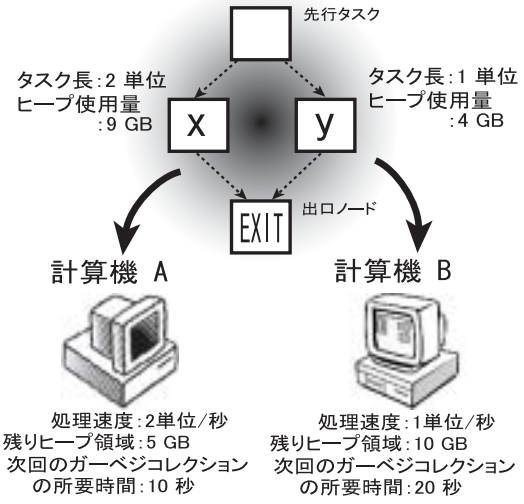

図 3 CP/MM の実行例.
Fig. 3 An Example of CP/MM Method.

に y が B に割り付けられ，計算時間は 11 秒になる．CP/MM では，各計算機のヒープ領域の残容量と各タスクのヒープ領域の消費量が考慮されるため，x が B に，y が A に割り付けられ，計算時間は 2 秒で済むことになる．

3.1 Easy CP/MM

上で説明した CP/MM では，タスク割り付け時にまだ割り付けられていないタスク全体に対し，分散記憶領域管理に関するコストを考慮してクリティカルパス長を再計算する必要がある．タスク数が大きい場合，この計算時間が問題となる．

このタスク割り付け時の計算時間を短くする簡易的な方法として，動的資源情報に **4** の **A** しか使用しない，

> クリティカルパス法において，優先順位の高いタスクほど利用可能な計算機のうちヒープ領域の残容量の大きいものに割り付ける．

が方法が考えられる．この方法だと，クリティカルパス法と比較してヒープ領域の残容量で利用可能な計算機を整列する手間が増えるだけで済む．

4. 予備実験とその結果

筆者らは，Easy CP/MM の評価を行なうため，現実の Grid コンピューティング・システムに近い環境上に構築されたタスクスケジューリングシステムに Easy CP/MM を含む 3 種類のタスクスケジューラを組み込み，それらによりスケジューリングされ分散実行される分散プログラムの計算時間を測定・比較した．

以下，実験に使用したテストベッドシステムについて 4.1 で，タスクスケジューリングシステムについて 4.2 で，分散プログラムについて 4.3 で説明する．その実験により得られた結果を 4.4 に示す．

4.1 テストベッドシステム

筆者らは，現実の Grid コンピューティング・システムに近い環境で実験を行なうため，タスクスケジューリングを含む Grid コンピューティング研究のためのテストベッドシステムを構築しており，その上で 4.2 で説明するタスクスケジューリングシステムを実装し，そこで Easy CP/MM とその他のアルゴリズムの比較を行なった（**表 1**，**図 4**）．

九州工業大学における飯塚キャンパスと戸畑キャンパス間は直線距離で 32 km 離れているが，北九州地域情報ネットワーク（`http://www.city.kitakyushu.jp/%7Ek1102030/network/index.html` ）と放送・通信機構の学術研究開発用ギガビットネットワーク（`http://www.jgn.tao.go.jp`）を利用して最も細い部分で 100 Mbps のバンド幅を持つ実験用ネットワークで接続されている（途中，九州大学を経由するため，本テストベッドシステムにおける両キャンパス間のネットワーク的な距離は 80 km 以上離れている）．この実験用ネットワークにおける ping コマンドで測定したパケットの往復に要する所要時間は 2.7 ミリ秒であった．

4.2 タスクスケジューリングシステム

Sun Microsystems 社が提供している RMI は，リモート計算機上のオブジェクトの呼び出しを容易に行なえるように，オブジェクト間でメッセージとして通信されるオブジェクトのシリアル化が自動的に行なわれ，リモートとローカルを問わず，オブジェクトをシームレスに呼び出すことができる Java 言語専用の通信基盤である[13]．著者らは，Grid コンピューティング・システム上でひとつのまとまった計算をする分散プログラムを効率的に実行できるタスクスケジューラに関する研究のため，RMI 上にタスクスケジューリングシステムを実装している．

今回の予備実験もこのタスクスケジューリングシステムを利用した．以下，4.2.1 でその構成，4.2.2 で実行動作，4.2.3 で利用法を説明する．

4.2.1 タスクスケジューリングシステムの構成

本タスクスケジューリングシステムは，RMI を通信基盤とし，以下のモジュールから構成されている（**図 5**）．

ユーザプログラム: 本タスクスケジューリングの利用者が 4.2.2 に示すメソッドを用いて作成する．事前に利用する計算機，タスクの登録，タスク間の依存関係の登録のみ行なえば，プログラム中の

表 1 テストベッドシステムのマシン構成.
Table 1 The Machine Configration of the Testbed System.

飯塚キャンパス	戸畑キャンパス
Pentium III 1GHz (1GB) Linux × 3	Pentium III 1GHz dual (1GB) FreeBSD × 4 Sun Fire V100 500MHz (640MB) Solaris 9 × 2

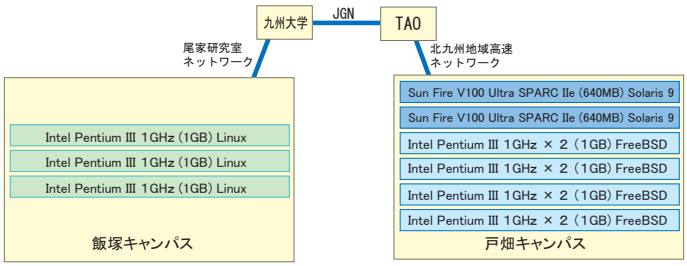

図 4 テストベッドシステムのネットワーク構成.
Fig. 4 The Network Configration of the Testbed System.

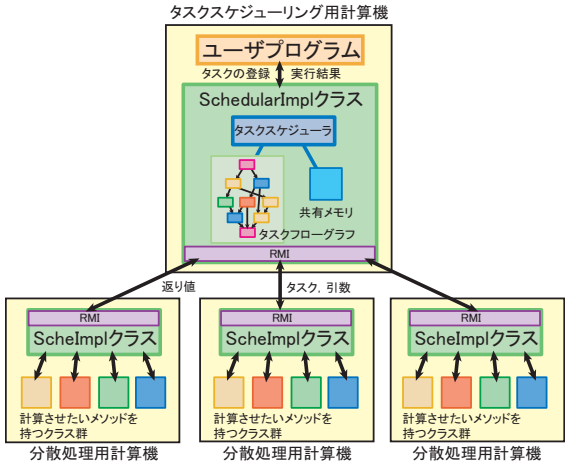

図 5 タスクスケジューリングシステムの構成
Fig. 5 The Configration of the Task Scheduling System.

どのタイミングでも分散計算を行なうことができ，その結果を処理する方法も自由に記述することができる．

計算させたいメソッドを持つクラス: このタスク中にスケジューリングするタスクを処理するメソッドを定義する．RMI における制限のため，返り値，引数ともにシリアライズ可能な任意のオブジェクト型である必要がある（例えば，オブジェクト型ではない `int` 型をやりとりする場合，代わりにオブジェクト型である `Integer` 型を利用することになる.）.

`SchedularImpl` **クラス:** 指定されたタスクスケジューリング・アルゴリズムを用いて，実行可能タスクをどの計算機に割り付けるか決定し，`ScheImpl` クラスと RMI を用いて通信することにより，タスクを割り付ける．MFG，タスクの返り値，引数，タスク間における共有変数の管理もこのクラスが受け持つ．

`ScheImpl` **クラス:** タスクの実行を行なうオブジェクトとそのオブジェクトが持つメソッドの管理を行ない，`SchedularImpl` クラスとの通信に応じてそれらのメソッドを実行する．どのメソッドが呼び出されるか実行時まで不明であるため，この部分の実装には Java のリフレクション機能を利用している．

4.2.2 タスクスケジューリングシステムの動作

本タスクスケジューリングシステムを使用した分散計算における実際の処理の流れは，以下のようになる．

(1) ユーザプログラムは，(`SchedularImpl` クラスのパブリックなメソッドを用いて) 以下の設定を行なう．同時に `SchedularImpl` クラスは，その内部にこれらの情報を格納するクラスに関する配列をつくる．

(a) メソッド `serveradd` により，使用するサーバを順に指定する．

(b) メソッド `setSchedular` により，使用するスケジューリングアルゴリズムを指定する．また，必要に応じてスケジューリングアルゴリズムをチューニングするパラメータも指定する．

(c) メソッド `taskadd` により，分散プログラムを構成するタスクを順に登録する．この際，タスクのクラス名，メソッド名，引数の型の配列，引数の配列を指定する．

また，タスク間の制御依存，データ依存を指定するため，先行タスク番号（登録順にタスク番号が付けられる）の配列，引数として実行済タスクの返り値を使用する場合，そのタスク番号の配列も指定する．さらに，コンパイラや利用者により静的に見積られたタスクの重み（実行時間やヒープ領域使用量）も指定する．

(2) ユーザプログラムから run メソッドが呼び出されると，前述の配列を用いて指定されたタスクスケジューリングに基づく分散計算が行なわれる．タスクの割り付けは，待機状態にある ScheImpl クラスのインスタンスに RMI を通じて行なう．

(3) タスクスケジューラがトレースしている MFG 中で後続タスクが存在しない出口ノードに達すると，分散プログラムは実行を終了する．これを確認すると run メソッドは結果をユーザプログラムに返し終了する．

4.2.3 タスクスケジューリングシステムの利用

本タスクスケジューリングシステムでは，タスクスケジューリングと RMI による通信基盤とタスク間の通信は，カプセル化されているため，各タスク自体の定義，各タスク間の依存関係を与えるだけで，タスクスケジューリングやタスク間のメッセージ通信の記述無しに，分散プログラムを Grid コンピューティング・システム上で自動実行することができる．さらに，各タスクの実行時間やヒープ領域使用量等の静的な見積もりを与えれば，タスクスケジューラにより，実行時間を短縮化することができる．本節では，このことを簡単な例を用いて説明する．

例えば，**図 6** は 1 + 2 の結果と 3 + 4 の結果を足す分散プログラムのタスクフローグラフである．本スケジューリングシステムを使用して，この 3 回の加算を分散計算で行なうユーザプログラムは**図 7** のように記述することができる．各行で行なわれている作業内容は，以下の通りである．

03 行目: タスクスクスケジューラ本体のインスタンスを生成する．

04 行目: タスクスケジューラを設定する．この場合は，CP/MM を指定した．

05 行目: 使用計算機を指定する．servers.txt ファイルから読み込むように指定した．

07-10 行目: 1 + 2 を行なうタスクの登録を行なう．第 1 引数から順に，クラス名は ex.sche.Task，実行すべきメソッド名が Add であることを指定している．第 3, 4 引数で，メソッドに渡す第 1 引数，第 2 引数がどちらも Integer 型であり，それぞれ Integer(1)，Integer(2) であることを指定している．第 5, 6 引数で， 直前の先行タスクは，最初から実行済のダミータスク 0 番のみであり，第 1 引数も第 2 引数も他のタスクの実行結果を利用しないこと (存在しないタスク番号である −1 で示されている）を指定している．最後に，第 7 引数で静的に評価されたタスクの重みは 1 であることを指定している．

11-14 行目: 3 + 4 を行なうタスクの登録を行なう．第 4 引数で渡している引数が Integer(3)，Integer(4) であること以外は，*07-10* 行目と同様である．

15-18 行目: *07-10* 行目に登録されたタスクの結果と，*11-14* 行目に登録されたタスクの結果を足すタスクの登録を行なう．*17* 行目で引数を渡さず（どちらも null としている），*18* 行目で直前の先行タスクが 1, 2 番目に登録されたタスクであり，第 1 引数，第 2 引数として，それぞれ，1, 2 番目に登録されたタスクの返り値を使用することを指定している．それ以外は *07-10* 行目と同様である．

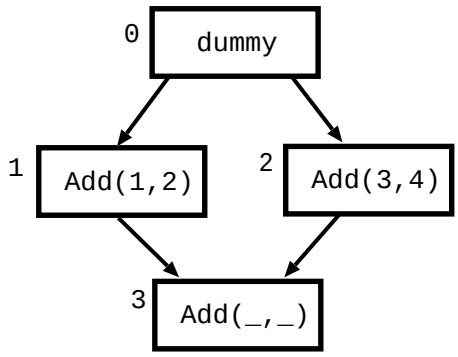

図 6 加算を行なうプログラムのタスクフローグラフ.
Fig. 6 A Task Flow Graph of a Program which Executes Addition Operationons.

また，このプログラムは 2 つの整数の加算を行ないその結果を返すタスク 1 種類だけからできている．このタスクは Object 型の Integer を使用し，**図 8** のように定義できる．

4.3 実験に用いた分散プログラム

本実験では，標準タスクグラフセット[17]（Standard Task Graph Set, STG Set と略す）に基づく分散プログラムを利用している．STG Set は，早稲田大学の笠原研究室がタスクスケジューリング研究の標準的

```
01 public class Client {
02  public static void main(String args[]) throws Exception {
03   SchedularImpl sche = new SchedularImpl(3); //スケジューラ
04   sche.setSchedular(sche.CPMEM_SCHEDULAR);   //スケジューラ設定
05   sche.serveradd("servers.txt");             //サーバー登録
06   // タスク登録
07   sche.taskadd("ex.sche.Task", "Add",
08         new Class[]   { Integer.class, Integer.class },
09         new Integer[] { new Integer(1), new Integer(2) },
10         new int[] { 0 }, new int[] { -1, -1 }, 1);
11   sche.taskadd("ex.sche.Task", "Add",
12         new Class[]   { Integer.class, Integer.class },
13         new Integer[] { new Integer(3), new Integer(4) },
14         new int[] { 0 }, new int[] { -1, -1 }, 1);
15   sche.taskadd("ex.sche.Task", "Add",
16         new Class[]   { Integer.class, Integer.class },
17         new Integer[] { null, null },
18         new int[] { 1, 2 }, new int[] { 1, 2 }, 1);
19   Object vals[] = sche.run();               // 分散実行
20   System.out.println("Result = "+vals[3]);  // 結果表示
21  }
22 }
```

図 7 簡単なユーザプログラムの例.
Fig. 7 A Simple Exapmle of User Program.

```
01 public class Task extends Thread {
02     public Integer Add(Integer x, Integer y) {
03        return new Integer(x.intValue() + y.intValue());
04     }
05 }
```

図 8 簡単なタスクの定義の例.
Fig. 8 A Simple Exapmle of Task Definition.

な例題集となるように，さまざまな複雑な依存関係があり，各タスクの処理時間も異なる 500 個のタスクを持つ MFG を 60 種類まとめ，タスクスケジューラ評価用の例題集として提供しているものである．

実験では，STG Set の 60 種類ある各例題を「タスクが 500 個あり，各 STG における依存関係に従って計算する必要がある分散プログラムであり，各タスクは，Integer 型のオブジェクトを各 MFG における計算時間 ×10000 個消費する（STG にある各タスクの 1 単位時間をテストベッド上の標準的な計算機である Pentinum III 1GHz のマシンで 0.1 ミリ秒程度で処理できるように設定した.）．各タスクの処理時間は消費するオブジェクトの個数に比例する.」とみなし，Easy CP/MM を含む 3 種類のタスクスケジューリング手法により分散プログラムを実行し，そのプログラム全体が終了するまでの計算時間を測定した．

これにより，クリティカルパス長で 41 から 2017 単位時間，並列度で 5.27 から 24.83 にわたる，さまざまな特性を持つ分散プログラム 60 種類に対するタスクスケジューラの特性を評価できることになる．

4.4 実験結果

テストベッドシステム上に実装されたタスクスケジューリングシステムに以下に示す 3 種類のタスクスケジューラを組み込み，60 種の分散プログラムを分散実行してその計算時間を測定した．なお，1 台に 2 つプロセッサが載っている計算機は，そのうちの 1 つのプロセッサを用いた．従って，利用したプロセッサは戸畑キャンパスのもので 6 台，飯塚キャンパスのもの 3 台，合計 9 プロセッサである．タスクスケジューラ本体は，戸畑キャンパスの Pentium III 1 GHz のディアルプロセッサマシン上（分散処理用計算機のひとつと兼用）で実行している．

LIST: リストスケジューラを素朴に実装したもので，空き計算機に優先順位を決めないで実行可能タスクを割り付ける．

CP: 各タスクの出口ノードまでの最短パス長を予め計算しておき，そのパスが長いものから順に空き計算機に割り付ける．

Easy CP/MM: 各タスクの出口ノードまでの最短パスを予め計算しておき，そのパスが長いもの

表 2 各タスクスケジューラによる平均計算時間.
Table 2 Mean of Elaps Time by using Each Taskschedular.

タスクスケジューラ	平均計算時間 (秒)
LIST	5.71
CP	6.08
Easy CP/MM	5.23

から順にヒープ領域の残りが大きい空き計算機に割り付ける.

組み込んだ 3 種類のスケジューラによる 60 個の分散プログラムの平均実行時間を**表 2** に示す. Easy CP/MM が最も分散プログラムの実行時間が短くて済み, LIST, CP の順に実行時間が長くなっている.

CP が最も実行時間を要しているのは, クリティカルパス長の計算や実行可能タスクの整列に時間を要したのと比較し, 分散計算用計算機数も 9 台程度と少ないため抽出した並列性をうまく活かせなかったのではないか, また CP の性能が良くないにもかかわらず, Easy CP/MM の実行時間が最も短くなっているのは, 計算機のヒープ領域の残容量を考慮して割り付けることにより, 限られた利用可能な計算機をより有効に利用することができ, 3 者のなかでは最も効率的にスケジューリングできた結果ではないかと, 筆者らは推測している.

特にクリティカルパス長が比較的長い分散プログラムでは Easy CP/MM が良い結果を出している (**図 9**). これは, クリティカルパス長が長ければ長いほどクリティカルパス上にあるタスクを優先的に割り付ける必要がより高くなるためと考えられる.

5. おわりに

本論文では, 分散記憶管理が必須となる Grid コンピューティングを行なう際にも, 効率的なタスクスケジューリングを行なえる CP/MM の提案を行なった. また, 広域分散テストベッドシステム上に RMI を用いて実装されたタスクスケジューリングシステムに提案手法の予備的な実装である Easy CP/MM を組み込み, 分散プログラムをテスト実行して他の手法と比較した. その結果, Easy CP/MM はクリティカルパス長が長い分散アプリケーションを比較的効率的にスケジューリングできることが示された.

筆者らは, プロセッサ台数が限られていると CP では, 計算時間をかけて抽出した並列性をうまく活かすことができないが, Easy CP/MM を利用すれば限られた利用可能計算機をより有効に利用することができ, 効率的にスケジューリングできているのはないかと推測した. 今後, この推測はよりプロセッサ台数の多い大規模なクラスタを利用して実験を行なえば, 確かめることができると考えられる.

今後の方針としては, 利用できる計算機の数とクリティカルパスの長さにより, スケジューラを切替えるハイブリッド方式を試したい. 例えば, 利用できる計算機が少なくクリティカルパス長も短い場合, 単純なリストスケジューリングを採用し, そうでない場合, Easy CP/MM を採用するなどが考えられる. この他に,(メタスケジューリング手法と同様に) タスクスケジューラに動的負荷変動の予測を組み入れたり, より多くのプロセッサを利用できるようにするため, プロセッサを割り付ける部分に分散プロセス割り当て手法 (例えば,[18] 等) を適用して評価していく予定である. また, 科学技術計算等, 実質的な処理を行なう分散アプリケーションに本手法を適用して評価することも合わせて行なう予定である.

謝辞 本研究の一部は, 文部科学省による科学研究費補助金 (課題番号 14019074) の支援を受けている. また, 九州工業大学 Grid 研究グループにおいて, 本研究に関する有益なご議論をいただいた. ここに記して謝意を表す.

参 考 文 献

1) Foster, I. and Kesselman, C.: The Globus Project: A Status Report, *Proceedings of IPPS/SPDP' 98 Heterogeneous Computing Workshop*, pp. 4-18 (1998).
2) Smith, W., Foster, I. and Taylor V.: Scheduling with Advanced Reservations, *Proceedings of the IPDPS Conference*, (2000).
3) Berman, F., Wolski, R. Figueira, S., Schopf, J. and Shao, G.: Application-Level Scheduling on Distributed Heterogeneous Networks, *Proceedings of Supercomputing '97*, (1997).
4) Koide, H. Hirayama, H., Murasugi, A. Hayashi, T. and Kasahara, H.: Meta-scheduling for a Cluster of Supercomputers, *Proceedings of International Conference on Supercomputers Workshop, Scheduling Algorithms for Parallel/Distributed Computing - From Theory to Practice* -, pp. 63-69 (1999).
5) 小出 洋, 笠原博徳: メタスケジューリング - 自動並列分散処理の試み -, bit, Vol. 33, No. 4, pp. 36-41 (2001).
6) Wolski, R., Spring, N. and Hayes, J.: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, *Journal of Future Generation Computing Systems*, Vol. 15, No. 5-6, pp. 757-768

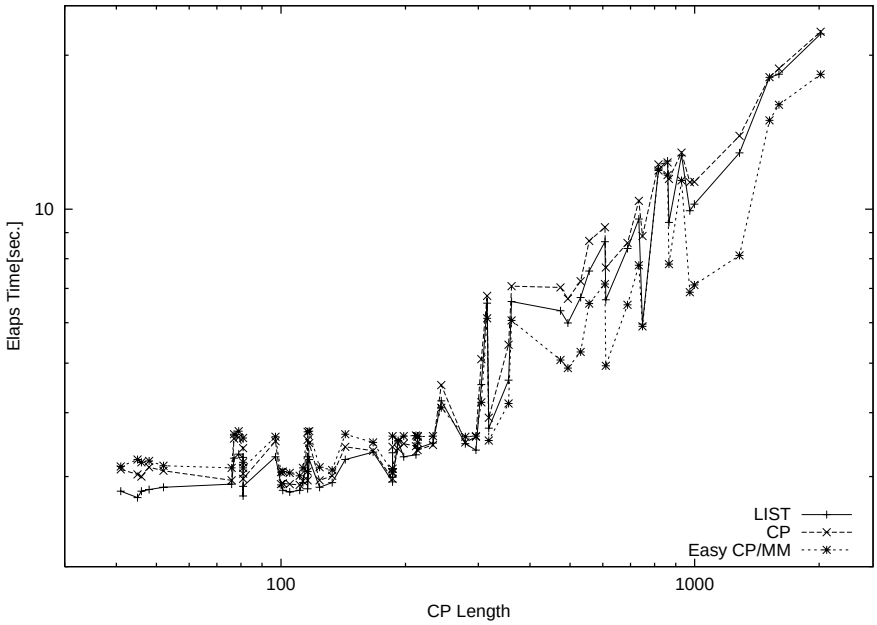

図 9 各分散プログラムのクリティカルパス長と各タスクスケジューラによる実行時間.
Fig. 9 Critical Path Length of Each Distributed Program and Elaps Time by using Each Taskschedular.

(1999).

7) 小出 洋, 山岸信寛, 武宮 博, 笠原博徳: 資源情報サーバにおける資源情報予約の評価, 情報処理学会論文誌：プログラミング, Vol. 42, No. SIG3(PRO10), pp.65-73 (2001).

8) 蟻川 浩, 砂原秀樹: Campas 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, Vol. 2002, No. 80, 2002-HPC-91, pp. 167-172 (2002).

9) Allcock, W., Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S.: The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets, *Journal of Network and Computer Applications*, Vol. 23, pp. 187-200 (2001).

10) Getov, V., Laszewski, G., Philippsen, M. and Foster, I.: Multi-Paradigm Communications in Java for Grid Computing, *to appear in ACM Communications* (2001).

11) dRuby: `http://www2a.biglobe.ne.jp/%7Eseki/ruby/druby.html`.

12) 秋山智宏, 中田秀基, 松岡 聡, 関口智嗣: Grid 環境に適した並列組合せ最適化システムの提案, 情報処理学会研究報告, Vol. 2002, No. 80, 2002-HPC-91, pp. 143-148 (2002).

13) Java Soft: *Java Remote Method Invocation Specification*, `http://java.sun.com/products/jdk/rmi`

(1997).

14) 笠原博徳:並列処理技術, コロナ社, pp. 135-162 (1991).

15) Kasahara, H., Obata, M., Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proceedings of 13th International Workshop on Languages and Compilers for Parallel Computing (LCPC' 00)* (2000).

16) 本多弘樹, 岩田雅彦, 笠原博徳: Fortran プログラム粗粒度タスク間の並列性検出手法, 電子情報通信学会論文誌, Vol. J73-D-1, No. 12, pp. 951-960 (1990).

17) 標準タスクグラフ: 早稲田大学理工学部笠原研究室, `http://www.kasahara.elec.ac.jp/schedule`.

18) 山本 寛, 川原 憲治, 滝根 哲哉, 尾家 祐二: グリッドコンピューティングにおける分割プロセス割り当て方法の検討とその基礎性能解析, 電子情報通信学会 技術研究報告, IN2002-8, pp.43-48 (2002).