社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－MPS－61 (8)
2006／9／14

余剰計算力を用いるグリッドのジョブスケジューリングにおける優先度制御の一手法

峰 久也[†] 藤本典幸[†] 東 竜一[†]
薄田昌広[††] 萩原兼一[†]

インターネットに接続された複数の計算機の余剰計算力を利用するデスクトップグリッドを複数のユーザで利用する場合，余剰計算力の変動を高い精度で予測することが難しいため，後から投入された計算量のより大きいジョブが，先に投入された計算量のより小さいジョブよりも先に完了してしまう（ジョブの追い越し）ことがある．本論文では，余剰計算力が動的にどのように変化しようとも，その変動の予測の必要無しに，ジョブの追い越しが決して起こらないことを保証するスケジューリングが可能であることを示す．そしてジョブの追い越しが起こらず，かつ，余剰計算力の消費量が小さいスケジューリングアルゴリズムを提案する．

HISAYA MINE[†] ,NORIYUKI FUJIMOTO[†] ,RYUICHI HIGASI[†]
,MASAHIRO SUSUKITA[††] and KENICHI HAGIHARA[†]

A desktop grid is a computing environment whose computing power is the spare computing power of computers connected via the Internet. Since precise performance prediction of the spare computing power is difficult in a desktop grid environment, a large job that is submitted later may be completed earlier than a small job that is submitted earlier if multiple users utilize the grid. We call this phenomenon job passing. This paper shows that job passing can be always avoided without any performance prediction regardless how the spare computing power varies over time. This paper also presents a novel scheduling algorithm such that job passing never occurs and the consumed spare computing power is small.

1. はじめに

インターネットに接続された複数の計算機の余剰計算力を利用して高速に計算する技術をデスクトップグリッドコンピューティングと呼ぶ．近年の高速なインターネットの普及や高性能な計算機の低価格化に伴い，デスクトップグリッドコンピューティングが注目され，様々な応用プロジェクトに利用され成果を挙げている．

グリッドを利用するユーザにとって，仕事を投入してから結果が返って来るまでの早さは重要な要素の一つである．また，ユーザが複数で使用する場合，ユーザ間の公平性が重要な要素となる．余剰計算力の変動を高い精度で予測することが難しい環境では，複数のユーザが同じグリッドを利用している場合，スケジューリングの仕方によっては，(1) 先に投入した計算量の小さいジョブよりも後に投入した計算量の大きいジョブが先に完了してしまう状況や，(2) ジョブを投入してから実行開始まで長時間待たされてしまう状況，(3) 性能の劣る計算機ばかりに割り当てられてしまう状況など一部のユーザにとって望ましくない状況が起こり得る．

そこで，本論文では，上記のような望ましくない状況のうち，状況 (1) が決して起こらず，かつ，余剰計算力の消費量が小さいスケジューリングアルゴリズムを提案する．本アルゴリズムはジョブの優先度を動的に計算する手法であるとみなすことができる．

2. 既存手法

スケジューリングは，その手法によって，静的スケジューリングと動的スケジューリングに大きく分けられる．静的スケジューリングでは実行前に，動的スケジューリングでは実行中にスケジューリングが決定さ

† 大阪大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†† 関西電力株式会社 電力技術研究所
Power Engineering R&D Center The Kansai Electric Power Co., Inc.

れる．グリッドのような，各プロセッサの性能が異なり実行中にプロセッサの速度が変動する環境では，動的スケジューリングが静的スケジューリングよりも適していることが知られている．そこで，本研究では動的スケジューリングを対象とする．

以降では，動的スケジューリングアルゴリズムである先着順と空間分割法の概要を説明する．

2.1 先着順

先着順アルゴリズムは，性能予測を行わず，ジョブサイズやプロセッサ速度といった情報を必要としない単純なスケジューリングアルゴリズムである．

先着順アルゴリズムでは，ジョブの投入順にタスクをプロセッサに割り当てる．これにより，性能で劣るプロセッサに割り当てられたタスクを処理している間に，性能の優れたプロセッサでより多くのタスクを割り当てて処理することができる．

2.2 空間分割法

空間分割法は，性能予測を行わず，ジョブサイズやプロセッサ速度といった情報を必要としない単純なスケジューリングアルゴリズムである．

空間分割法では，プロセッサ全体をジョブ数の分だけ均等になるよう分割し，分割したプロセッサ群とジョブを対応づけてタスクを割り当てる．これにより，プロセッサの数がジョブの数より多ければ，各ジョブは必ず1台以上のプロセッサを使用することができるため，先に投入されたジョブを待つこと無くジョブの実行が開始される．また，各ジョブはタスクが実行されている状況が常に維持できる．

3. 提案手法

提案するスケジューリングアルゴリズムは，ジョブの計算量の情報を用いる，タスク複製を行う，性能予測を行わないといった特徴を持つマスターワーカ型のアルゴリズムである (表1)．

提案アルゴリズムの疑似コードを図1に示す．提案アルゴリズムでは，ジョブをラウンドロビン順で選択し，更に，そのジョブのタスクをラウンドロビン順で選択し，そのタスクのインスタンスをプロセッサに割り当てる．但し，選択したタスクTが以下の2条件を同時に満たす場合，その2条件のいずれかが満たされなくなるまで，Tの割当を保留する．

- Tが属するジョブにおいて，Tが最後の未割り当てタスクである．
- Tが属するジョブJが投入された時点でグリッドで計算中で，かつ，Jの計算量よりも小さいジョブが，Tの選択時にもグリッドで計算中である．

表1 各アルゴリズムにおける特徴の比較

	複製	利用情報
提案アルゴリズム	する	ジョブサイズ
先着順	しない	—
空間分割法	しない	—

これにより，先に投入された計算量の小さなジョブが後に投入された計算量の大きなジョブよりも実行完了が遅くなることを回避する．この方法は単純であるが，各プロセッサの性能がどのように動的に変動しようとも，その性能予測なしに，1節で述べた望ましくない状況 (1) を必ず回避できることに注意されたい．

また，性能の劣るプロセッサにサイズの大きいタスクが割り当てられた場合には，そのタスクが実行終了するまでにタスクの割当が一巡して，同じタスクが割当の対象になる場合がある．このときは，タスクを複製して別のプロセッサに割り当てる．すなわち，1つのタスクの異なるインスタンスを別のプロセッサで実行開始する．これは，後から実行開始した複製インスタンスの方が，先に実行開始したオリジナルインスタンスよりも早く完了する可能性があるからである．サイズが等しいタスクからなるシングルジョブの場合には，ラウンドロビン順で複製タスクを選択すると，スケジュール全体の余剰計算力の消費量が小さくなることが証明されている[1]．このため本アルゴリズムでもラウンドロビン順に複製タスクを選択する．

4. 評価実験

シミュレーションにより，各アルゴリズムの比較を行った．シミュレーションに用いたスケジューリングアルゴリズムは，提案アルゴリズム，先着順，空間分割法である．

4.1 シミュレーション条件

シミュレーションに用いたパラメータを表2に示す．

Paranhos らの性能モデル[2]を参考に，企業や研究機関の PC は3年間使われると仮定し，ムーアの法則[3]から，デスクトップグリッド中の PC のピーク性能の比率は，1, 2, 4 とした．

PC 1台につき，プロセッサは1台であるとし，各プロセッサは，負荷が低く余剰計算力が大きい定常状態と，負荷が高く余剰計算力が小さい高負荷状態の2つの状態を一定の確率で遷移する．各状態ではプロセッサ毎に決められた範囲内でランダムに性能を決定する．

PC を 128 台とし，32 個のジョブを等間隔で投入する．ジョブは，サイズの異なる 128 個のタスクから構成され，ジョブは全て，同じタスク構成となっている．

各アルゴリズム毎に，ジョブの投入間隔を 50, 100,

```
入力:  プロセッサ数 P
       N 個のジョブ J_1, J_2, ..., J_N
       各ジョブは独立タスクの集合
         (J_i = {T_{i,1}, T_{i,2}, ..., T_{i,m(i)}})
出力:  各タスクのスケジュール

/* 初期処理 */
for(i = 0; i < N; i++){
   ジョブ J_i のタスクキュー Q_i を作成
}
/* 割当処理 */
while(未割当のタスクが存在している){
   if(アイドル状態のプロセッサが存在しない){
      アイドル状態のプロセッサが現われるまで待つ
      実行が完了したタスクを T_{i,j} とする
      if(T_{i,j} の実行中のインスタンスが存在する){
         T_{i,j} のインスタンスを強制終了
      }
      T_{i,j} を Q_i から削除
      if(Q_i が空){
         /* J_i は全てのタスクが実行完了 */
         Q_i を削除
      }
   }
   Q_1, Q_2, ..., Q_m の1つをラウンドロビン順に
   選んで，その先頭タスク T_{i,j} を取り出し，
   キュー Q_i の末尾に T_{i,j} を入れる
   if(J_i の未割り当てタスクが残り1個 &&
   J_i の計算量が，J_i の先発ジョブの
   いずれかの計算量と同等かそれ以上である){
      /* 何もしない */
      continue;
   }
   else{
      T_{i,j} のインスタンスをアイドルの
      プロセッサに割当
   }
}
```

図 1　提案アルゴリズムの疑似コード

200 の 3 段階に変化させてシミュレーションを行った．

4.2 評価基準

本論文では，ユーザ間の公平性を評価するために，(1) ジョブを投入した時点で先に投入された計算量の小さいジョブよりも先に完了してしまう割合，(2) 各ジョブの投入からジョブの実行開始までに要する時間の分散，(3) 各ジョブの投入からジョブの実行終了までに要する時間の分散を基準に用いた．また，余剰計算力の消費量を評価するため，各ジョブの実行開始からジョブの実行終了までの時間を基準に用いた．

4.3 実験結果

まず，ジョブの投入間隔別に，ジョブの待機時間，ジョブの実行時間，ジョブの全実行時間の平均と分散の値を表 3〜5 に示す．表の上段が平均値，下段が分散値を表す．

表 2　シミュレーションに用いた条件

ジョブ数	32	
ジョブ投入間隔	50,100,200	
タスク数	128	
タスクサイズ (タスクの命令数)	2000000〜6000000 で ランダムに設定	
プロセッサ台数	128	
プロセッサのピーク性能 (無負荷の場合に単位時間に 計算できる命令数)	10000,20000,40000	
プロセッサ速度変動パターン	定常状態と高負荷状態を 一定確率で遷移する 2 種類のパターン	
定常状態から高負荷状態への 遷移確率	1/1000	1/100
高負荷状態から定常状態への 遷移確率	1/100	1/10
定常状態でのプロセッサ速度	ピーク性能の 95%〜100%	ピーク性能の 80%〜100%
高負荷状態でのプロセッサ速度	ピーク性能の 0%〜5%	ピーク性能の 0%〜20%

表 3　投入間隔 50

	提案アルゴリズム	先着順	空間分割法
待機時間	17.46 288.25	2481.22 2151992.71	946.06 1681811.36
実行時間	6038.94 1520696.30	861.47 30197.87	3736.71 1771239.26
全時間	6056.40 1540032.09	3342.70 1875579.13	4682.78 3230730.94

表 4　投入間隔 100

	提案アルゴリズム	先着順	空間分割法
待機時間	12.80 103.21	1712.72 993998.45	338.91 476401.08
実行時間	4480.38 1389558.67	858.39 27868.82	3712.50 1745349.17
全時間	4493.19 1401586.82	2571.12 826775.51	4051.42 2333383.48

次に，提案アルゴリズム (図 2)，先着順 (図 3)，空間分割法 (図 4) における 1 回のシミュレーションで得たそれぞれのガントチャートを示す．ここで，ラストタスクブロック時間とは，ジョブの最後の未割り当てタスクの割当が保留され続けた時間を指す．

4.4 考　　察

提案アルゴリズムは，待機時間に注目すると 3 手法の中で一番短く，その分散値も他のアルゴリズムに比べて小さいものとなっている．また，一旦ジョブの最後の未割り当てタスクの割当が保留されると，次々と後続のジョブも最後の未割り当てタスクの割当が保

表 5　投入間隔 200

	提案アルゴリズム	先着順	空間分割法
待機時間	5.85	209.11	85.70
	22.58	8578.96	21311.28
実行時間	1679.17	846.82	2416.54
	97433.16	25156.13	290454.05
全時間	1685.02	1055.94	2502.24
	98187.77	11847.35	364775.54

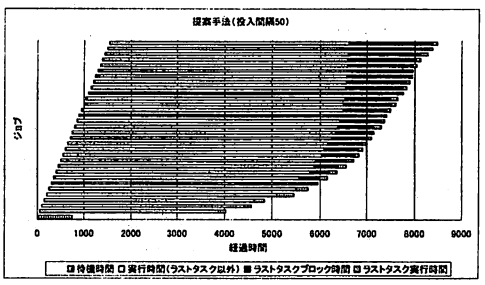

図 2　提案アルゴリズム

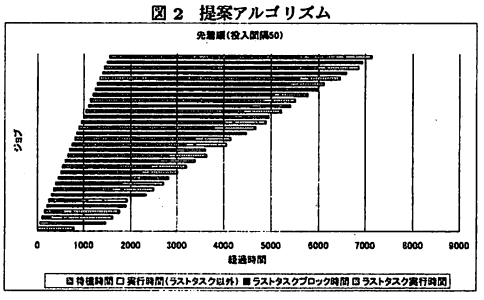

図 3　先着順

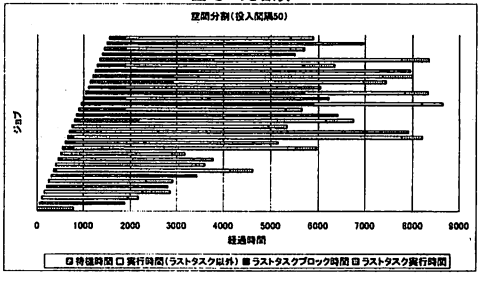

図 4　空間分割法

留され，1 つのジョブ以外は全て割当が保留された状態になる現象が発生した．これは，タスクの保留により別のジョブにプロセッサが割り当てられることで，他のジョブの実行が促進されたことによる．この状態に陥ると，以降 1 つのジョブを全力で処理する方式となる．

先着順は，実行時間に注目すると 3 手法中で一番短いが，投入の時刻によっては，待ち時間に差が現われる．また，ジョブの投入時刻と終了時刻の逆転が起こることが確認された．これは，先に投入されたジョブの最後に終了するタスクが性能の劣るプロセッサに割り当てられた際に，後に投入されたジョブよりも実行時間が延びたと考えられる．

5. ま と め

提案アルゴリズムは，シミュレーションの結果から先に投入した計算量の小さいジョブよりも後に投入した計算量の大きいジョブが先に完了することはない，ジョブを投入してから実行開始まで長時間待たされることが少なく各ジョブの待ち時間の差が少ない，といった点から先着順，空間分割法の 2 つのアルゴリズムと比較して公平なアルゴリズムであると言える．

しかし，ジョブの実行時間や待ち時間も含めた全体の時間では先着順に大きく離され，提案アルゴリズムには改善の余地が残されている．各ジョブ間の公平さを保ちながらも，余剰計算力の消費量を更に抑えるよう，提案アルゴリズムの改良を今後の課題とする．

参 考 文 献

1) 藤本典幸, 萩原兼一: "グリッド上でのパラメータスウィープ計算を対象として消費余剰計算力の最小化をねらった動的タスクスケジューリングのための近似アルゴリズム", 情報処理学会論文誌: 数理モデル化と応用,No.SIG10(TOM 12),pp.1-9,2005

2) D. Paranhos, W. Cirne, and F. V. Brasileiro : "Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids", International Conference on Parallel and Distributed Computing (Euro-Par), Lecture Notes in Computer Science, Vol.2790, pp.169-180, 2003

3) G. E. Moore : "Cramming More Components onto Integrated Circuits", Electronics, Vol.38, No.8, 1965