分散システム／インターネット運用技術　27－2
（２００２．１０．１８）

ボランティアコンピューティングにおけるスケジューリングの一手法

妹尾 博之 †　　中村 康弘 †

<あらまし>
この報告では複数ジョブを同時実行するボランティアコンピューティング (VC) 環境においてタスクを各ジョブに公平に配分するスケジューリングアルゴリズムを提案する. VC は世界的な規模のプロジェクトが進められている一方, 大学・企業等の閉じた環境における余剰計算機パワーを有効活用する技術としても注目されている.VC においては計算に従事する個々のコンピュータの稼働状況等を一元管理しないことを前提としているため, 利用可能な計算資源を事前に予測することができない. このため, ジョブの終了時刻を予測することが難しい. この報告では, 複数ジョブの処理の公平性を維持することを目的として,PC の性能評価に着目したタスク配分アルゴリズムを提案し, シミュレーションによりその有効性を示す.

A Scheduling Algorithm on Volunteer Computing

Hiroyuki SENO †　　Yasuhiro NAKAMURA †

< Summary >
This paper presents a scheduling algorithm on volunteer computing. It has been made advanced in study of volunteer computing, and many practical projects has been a great success on a scale of world wide. Volunteer computing has some problems.One of them is difficult to accurately predict the end time of a job,because the administrator can not know calculation resources which can be used. This paper propose a task distribution algorithm which keep the fairness of two or more jobs,and show the validity by simulation.

1. はじめに

ボランティアコンピューティング (以下 VC) はネットワークに接続された多数のパーソナルコンピュータ (以下 PC) の余剰計算資源を有効活用する試みであり, SETI@home [1] をはじめ, いくつかのパラメータサーチ型プロジェクトが世界規模で実運用されている. しかし, これらの実装では単一目的のジョブを実行し続けるのみであり, 複数のジョブを動的に実行する環境にない.

† 防衛大学校 National Defense Academy

本研究では, 大学・企業等の閉じた組織内に数多く存在する PC の余剰計算資源を有効活用するシステムを想定する. このシステムに複数のジョブが投入されたとき, 各ジョブを公平に PC 群 (以下, サーバントと呼ぶ) に割り当てるアルゴリズムを提案する. 処理の公平性の概念を一意に規定することは難しいが, ここではジョブ処理の困難さを定量化した上で, 複数のジョブを同時に処理中のシステムが各々のジョブに同等の計算資源を費やすことと定義する. サーバントを組織に所属する PC に限定するため,

世界規模で実運用されているプロジェクトと比較すると, システムの処理能力は低い. しかし一方で,PC 所有者の情報を把握しやすいため, 不正な処理結果を返すユーザを識別でき, 一部のプロジェクトで生じている不正者対策 [2] に重点を置かずに済む利点がある.

VC 以外の分散コンピューティング環境 [3] の場合は, 利用可能計算資源に関する情報収集及び負荷予測 [4] を行なうことで適切にスケジューリングを行なうことができる. しかし,VC の場合は一般的な PC 群を用いて計算を実行するため, サーバントの処理能力を事前に知ることができず, また, いつ電源投入・遮断されるかについても不確定である. このため, 情報収集及び負荷予測データの有効利用が難しい.

本稿では, サーバントの処理能力を順位づける指標 ω_i 及びシステムがジョブ処理に費やした計算資源を示す指標として累積処理量を導入し, 累積処理量がジョブごとに同値となるよう補正を加え, 処理能力の高いサーバントからスケジューリングしていく. 本アルゴリズムを適用すると,PC の将来負荷を予測することなく, 複数のジョブを公平に処理するスケジューリングが可能となる.

以下, 2 章で本研究が想定する環境及び用語を定義する. 3 章で計算量, ω_i, 公平さについて述べた後, 提案するスケジューリングアルゴリズムを記述する. 4 章で本アルゴリズムに基づくシミュレーションの方法を示し,5 章でその結果及び考察を述べる.

2. システムの定義

2.1 想定する環境

ここでは以下のような VC 環境を想定する.

- サーバントの台数は一定
- サーバントの OS は MS Windows や Mac OS 等の一般のパソコン用 OS
- サーバントの CPU 負荷,CPU 性能, 電源投入・遮断状態に差異がある. すなわち常時稼働し高負荷なサーバントがある一方で, 一日に 1 時間程度メールの読み書きのみに利用されるサーバントも存在する.
- サーバント間の通信は発生しない.
- サーバントの負荷状態を予測しない.

2.2 用語の定義

本報告で使用する用語を以下のとおり定義する.

- **ジョブ**　ユーザがシステムに処理を依頼するプログラムの全体.
- **タスク**　ジョブをパラメータにより分割した処理の単位. タスクは分割できないので必ず 1 サーバントにて処理される.
- **計算量**　1 タスクの計算処理に必要な計算資源の大きさを表す値.
- **累積処理量**　計算量と処理済みタスク数を乗じた値. システムがそのジョブを処理するために費やした計算資源を示す.

3. 提案スケジューリング方式

3.1 タスクの計算量

提案するスケジューリング法を実装・運用するには, 処理の目安となる計算量を決定する必要がある. 簡便な決定方法として, ジョブの演算数又はジョブサイズの大小による決定方法が考えられるが, 実際に数タスクを処理してみることにより計算量を見積ることもできる. 本稿では, これらの手段により計算量はあらかじめ既知であるものとする.

3.2 サーバント性能の評価

各サーバントは CPU 性能や稼働時間,CPU 負荷が異なるため, これらの尺度のみを用いてサーバントの性能を一意に決定することはできない. たとえば CPU 性能が最も高くても稼働時間が極めて短ければサーバントとしての処理能力は最高であるとは限らない.

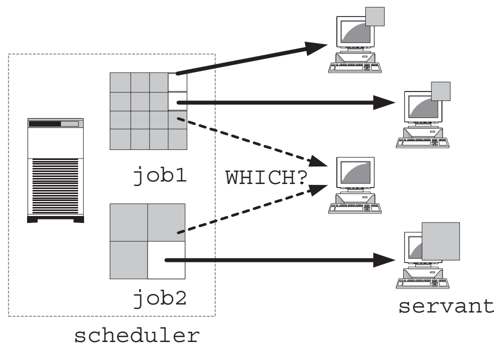


図 1　複数タスク配分の概念図

本稿ではサーバント i の性能を表す尺度として,式 (1) に示す ω_i を導入する. ω_i は, サーバント i が電源を投入されており, かつ処理すべきタスクを持たない場合に値を持ち, それ以外の場合は 0 とする.

$$\omega_i = \begin{cases} \sum_{j=1}^{n} (d_j \times r_j) \\ 0 \end{cases} \qquad (i = 1, 2, \ldots, N) \qquad (1)$$

ここで d_j はジョブ番号 j の計算量,r_j はサーバント i が処理したジョブ j のタスク数である. また,n は処理要求されたジョブの総数,N はサーバントの総数である. VC では, 単位時間により多くのタスクを処理したサーバントほど性能が高いと評価する. 式 (1) 右辺に CPU 性能や稼働率は直接現れないが,r_j にそれらの尺度が反映されている. 提案するスケジューリングアルゴリズムでは, 各ターンで常に ω_i の大きいサーバントから順にタスクを割り振る. これにより, ω_i を考慮に入れない場合と比較すると, タスク処理の効率が向上するものと期待できる.

3.3　ジョブ処理の公平性

通常のスケジューリング方式では, 一般に, 使用した CPU 時間等を計算資源の大きさを表すパラメータとして用いる. しかし VC においてはサーバントの CPU 性能や稼働時間の差が極めて大きいため, 実際に処理されたタスク数を用いてジョブ処理に費やした計算資源を表す手法がより適切であると考えられる. そこでジョブ処理のためにシステムが費やした計算資源を表すパラメータとして, 累積処理量を導入する. ジョブ j の累積処理量 C_j を式 (2) で定義する.

$$C_j = d_j \times \sum_{i=1}^{N} f_i \qquad (j = 1, 2, \ldots, n) \qquad (2)$$

ここで f_i はサーバント i が処理完了したジョブ j のタスク数,N はサーバントの総数,n は処理要求されたジョブの総数である. 各ジョブの累積処理量が等しいとき, システムは各ジョブを公平に扱ったとみなすことができる.

3.4　スケジューリングアルゴリズム

ジョブ 1 を処理中のシステムにジョブ 2 を投入する場合を考える. 単位時間ごとに, 以下のアルゴリズムに基づいてタスクをサーバントに配分していく.

step1 タスク配分可能なすべてのサーバントを ω_i の降順に並べる.

step2 各ジョブの累積処理量が同値でなければ,3.5 節に記述する誤差補正を行なう.

step3 サーバントの ω_i を合計し,Ω とする.

$$\Omega = \sum_{i=1}^{N} \omega_i$$

step4 ω_i をジョブ 1, ジョブ 2 に割り当てるためのキューの大きさ Q_j を求める.

$$Q_1 = \Omega \times \frac{d_2}{d_1 + d_2}$$
$$Q_2 = \Omega \times \frac{d_1}{d_1 + d_2}$$

step5 未割り当てのサーバントのうち ω_i が最大のものをキューの残り容量が大きいジョブに割り当てる. この処理をタスク配分可能なサーバントが無くなるまで繰り返す.

図 2 にサーバント割り当ての例を示す. 図 2 では, キュー Q_1, Q_2 の大きさを考慮し, サーバント 1 をジョブ 1 に割り当てる. サーバント 2 は残り容量

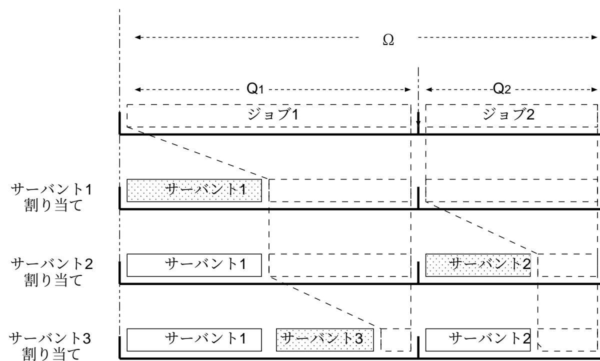


図 2　ω_i のサイズによるジョブ割り当て

がジョブ 2 のほうが大きいため, ジョブ 2 に割り当てる. タスク配分可能なすべてのサーバントに対して, この処理を繰り返す.

3.5　誤差補正

タスク配分を繰り返すと, step4 の処理だけでは時間経過につれて累積処理量 C_j を同値に保てず, 誤差 $\Delta C = |C_1 - C_2|$ が生じる. これはサーバントが返してくる計算結果の量が不定なためである. そこで step2 において以下の誤差補正を行なう.

ジョブ 1 とジョブ 2 がシステムで同時に処理される場合を考える. ある時刻に，タスク配分可能なサーバントが N_{active} 台あり，各ジョブの累積処理量はそれぞれ C_1,C_2 とする．すなわち，図 3 の太線内の面積が 累積処理量 C_i = 処理済みタスク数 × 計算量 である．図 3 では, ジョブ 2 が ΔC だけ小さいため ΔC の補正が必要になる．また，次ターンで増加が見込まれる累積処理量の増加分を ΔC_1,ΔC_2 とする.

いま，ジョブ 2 に対して誤差補正を行なう場合,N_{active} 台のサーバントを，$N_{active} = N_{\Delta C} + N_{\Delta C_1} + N_{\Delta C_2}$ の 3 種類に分けて考える．N_{active} 台のサーバントのうち, 累積処理量の誤差補正に $N_{\Delta C}$ 台使用し, サーバント $N_{\Delta C_1}, N_{\Delta C_2}$ をそれぞれのジョブの計算促進に使用する．$C_1 > C_2$ の場合,ΔC の補正のために

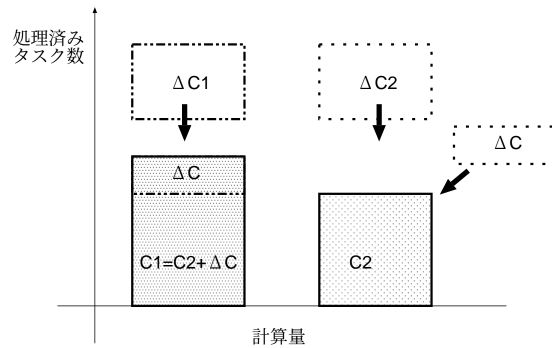


図 3　誤差補正の概念図

処理すべきタスク数を T_{todo} とすれば,

$$T_{todo} = \frac{\Delta C}{d_2}$$

であり，一度に 1 台のサーバントが複数のタスクを処理できないことから，以下の場合分けを考える.

$N_{active} \leq T_{todo}$ の場合　N_{active} 台すべてをジョブ 2 に割り当て ΔC の補正を行なう.

$N_{active} > T_{todo}$ の場合　T_{todo} 台をジョブ 2 に割り当て ΔC の補正を行ない, 残りのサーバントをスケジューリングアルゴリズムの step4, step5 に従い計算促進用としてジョブ 1, ジョブ 2 にそれぞれ割り当てる.

3.6　ジョブ数 3 以上の場合

アルゴリズムの説明をジョブ数 2 として行なったが, ジョブ数 3 以上の場合でも, 同様のスケジューリングが可能である. 各ターンで, 累積処理量が最小のジョブから順にアルゴリズムを適用すれば, 各ジョブを公平に扱うことができる.

4.　シミュレーション

アルゴリズムを検証するためにシミュレーションを行なった．ジョブ 1 処理開始から 10 ターン後にジョブ 2 を投入するものとする. 表 1 にシミュレーションに用いたパラメータを示す．CPU 性能及び

サーバント台数	500
ジョブ数	2
タスク数	100,000
CPU 性能	0.00 - 1.00
稼働率	0.00 - 1.00
CPU 負荷	0.00 - 1.00
ジョブ 2 投入	10 ターン後

表 1　シミュレーションパラメータ

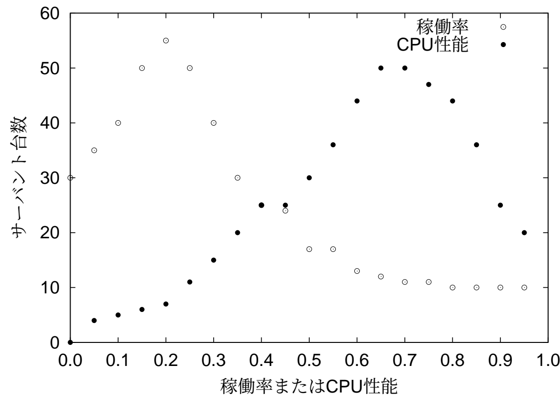


図 4　初期データ分布

稼働率は環境により異なるのが一般的であるが, ここでは図 4 に示す初期条件を用いた. CPU 負荷はターンの都度, サーバントごとに乱数で決定した.

また, 表 1 のほか, ジョブ 1, ジョブ 2 それぞれの計算量パラメータがあり, シミュレーションにより異なるので, その都度明記する.

サーバントの電源投入・遮断の状態は,0 から 1 の範囲の乱数を生成し, 乱数と稼働率の大小により決定した. 稼働サーバントがタスク処理中でなければ, アルゴリズムに従いタスクを配分する. サーバントは配分されたタスクの計算量を残り計算量パラメータとして保存する. その後, 電源投入中のサーバントに対し, 残り計算量から CPU 性能 $\times$ 稼働率 $\times (1.00 - CPU$ 負荷$)$ を減算した値を新たな残り計算量とする. 残り計算量が 0 以下になればタスク処理終了とし, サーバからの次のタスク配分を待つ.

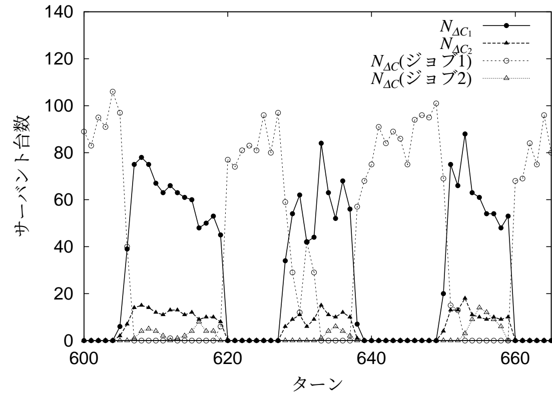
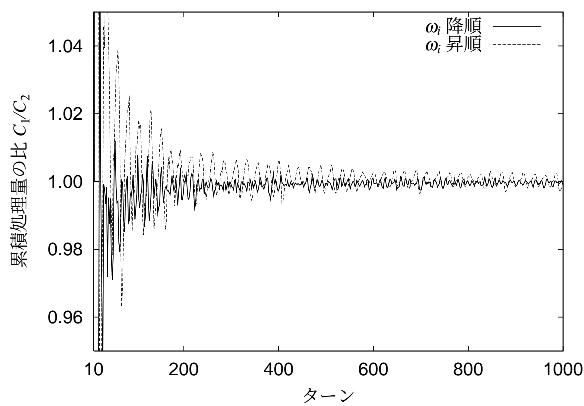


図 5: 時間経過に伴う $N_{\Delta C}$ と $N_{\Delta C_1}, N_{\Delta C_2}$ の変化の一例

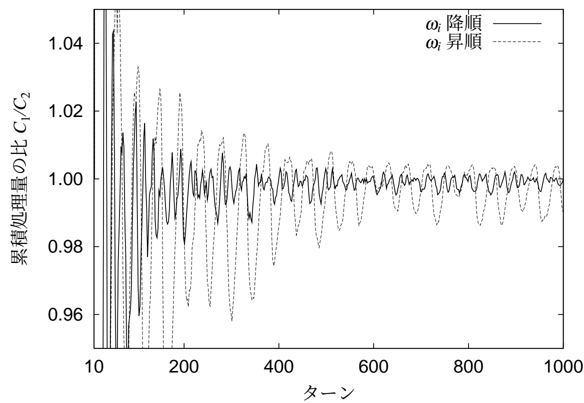
5.　結果および考察

時間経過に伴う $N_{\Delta C_1}, N_{\Delta C_2}$ と $N_{\Delta C}$ の変化を調べた. その結果を図 5 に示す. 図 5 は計算量 $d_1 = 0.1$, $d_2 = 1.0$ として行なった実験結果の一部をグラフ化した図であり,$N_{\Delta C}$ はジョブ 1 を処理する場合とジョブ 2 を処理する場合に区別してプロットした. 図 5 には, 誤差補正の目的でジョブ 1 にのみサーバントを割り当てるフェイズと, 計算促進の目的で N_{C_1}, N_{C_2} が増加するフェイズが重複しながら交互に現れている. 630 ターン付近に注目すると,$N_{\Delta C}$ が減少し,$N_{\Delta C_1}$,$N_{\Delta C_2}$ が増加している. つまり, 誤差補正のためにのみジョブ 1 の割り当てを行なわれていたが,ΔC が 0 に近づくにつれて計算促進のために各ジョブに割り当てが開始される様子を示している.

また, システムに複数ジョブが投入されている間におけるジョブ間の公平性の維持を確認するために, 累積処理量の比 C_1/C_2 の時間経過に伴う変化を調べた. 図 6 にその結果を示す. アルゴリズムの step2 でサーバントを ω_i の降順に並べた場合を実線で, 昇順に並べた場合を破線で示す. 実験に使用した計算量 0.1 は標準的なサーバントであればタスクを 1 回のターンで処理可能な計算量であり, 計算量 1.0 は高性能なサーバントであっても処理完了までに数ターンを要する計算量である.



(a) 計算量 d1:d2=0.1:0.5



(b) 計算量 d1:d2=0.1:1.0

図 6　累積処理量の比 $C_1/C2$ の変化

実験では, ジョブ 2 をシステムに投入した直後は 1.1 を大きく超えた累積処理量の比 C_1/C_2 を示し, ターンが進むにつれ 1.00 近辺に収束した. 累積処理量がほぼ同値になっているため, 複数ジョブを公平に扱っているものと考えられる. ただし, 累積処理量の比は 1.00 を中心に振幅 0.01 程度の振幅運動を繰り返す. これは提案したアルゴリズム上防げない現象であり, ジョブの処理が終了するタイミングによって, どちらか一方のジョブが微小であるが大きい累積処理量を持つ.

ω_i の昇順の場合は, 本アルゴリズムにおいて最も ω_i を考慮しないタスクの配分方法であるとみなせるので, ω_i の有効性を確認するために,ω_i の降順に並べた場合と昇順に並べた場合を比較する. 図 6 において,ω_i 昇順の場合に振幅が大きい傾向を示した. ジョブ間の計算量の比が大きくなるにつれ, この傾向は強くなる. ω_i 降順では,ω_i が大きいすなわち過去の貢献度が高いサーバントが累積処理量の誤差補正に利用され, 累積処理量の比のゆれを最小に抑える. ω_i 昇順では, ω_i が小さいサーバントを誤差補正に利用するため, 補正に遅延が生じる. このことから, サーバントの処理能力を順位づける指標 ω_i の有効性を確認できる.

6.　まとめ

本稿では, 大学・企業等の閉じた環境で行なわれる VC を想定し, 複数のジョブを公平にスケジューリングするアルゴリズム及びサーバントの処理能力を順位づける指標 ω_i を提案した. また, 利用可能な計算資源に関する事前情報の収集や将来負荷の予測を行なわずとも, 複数ジョブを公平にスケジューリングできることをシミュレーションにより確認した.

参考文献

[1] SETI@home
http://setiathome.ssl.berkeley.edu/

[2] Luis F.G.Sarmenta
:Sabotage-Tolerance Mechanisms for Volunteer Computing Systems: ACM/IEEE International Symposium on Cluster Computing and the Grid,(2001).

[3] Mark Baker, Rajkumar Buyya, and Domenico Laforenza :Grids and Grid Technologies for Wide-Area Distributed Computing, Software: Practice and Experience Journal, Wiley Press,(2002).

[4] Wolski,R.,Spring,N.T. and Hayes,J.:The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, Technical Report TR-CS98-599,UCSD(1998).