# ボランティアコンピューティングにおける締切時間を考慮したクライアントレベルスケジューリング手法

村田善智[†1]　遠藤聡明[†3]　江川隆輔[†1]
滝沢寛之[†2]　小林広明[†1]

遊休計算資源を活用するボランティアコンピューティングは，単体では実現困難な巨大な計算能力を安価に実現可能なことから，高性能計算基盤として着目されている．しかし，遊休計算資源が提供する計算能力が動的に変動するため，ボランティアコンピューティングではタスクの計算が締め切り時間までに完了する保証がない．もし締め切り時間までにタスクの計算が完了しなかった場合，クライアントの計算能力が浪費されたことになる．本論文は，クライアントの計算効率を高めるために，締め切り時間を考慮したタスクスケジューリング手法を提案する．提案するタスクスケジューリング手法は，タスク取得処理と CPU スケジューリング処理で構成される．タスク取得処理では，ワークキュー長を調整し，サーバからの余剰タスク取得を回避する．CPU スケジューリング処理では，締切時間までに計算が完了しないタスクをあらかじめキャンセルすることで，計算能力の浪費を防ぐ．シミュレーションによる評価結果では，提案手法が締め切りまでに計算が完了するタスクの数を増やし，高効率なボランティアコンピューティング環境を実現出来ることが示された．

# A Client-Level Deadline Scheduling Strategy for Volunteer Computing Systems

Yoshitomo Murata,[†1] Toshiaki Endo,[†3] Ryusuke Egawa,[†1]
Hiroyuki Takizawa[†2] and Hiroaki Kobayashi[†1]

Volunteer computing is a new and promising style of large-scale distributed computing, which uses idle computing resources of individuals. However, a volunteer computing environment cannot guarantee to achieve a certain processing speed. One problem is that the CPU cycles spent for job execution are wasted when the deadline of the job is missed. This paper proposes a deadline scheduling for volunteer computing systems. The proposed deadline scheduling employs two policies: work fetch policy and CPU scheduling policy. The work fetch policy resizes the length of work queue to avoid downloading excess tasks. The CPU scheduling policy rejects tasks that might miss a deadline. By using a BOINC-client simulator, this paper examines the effect of the proposed mechanism. The experimental results show that the proposed scheduling policies reduce the number of deadline missed task and increase the usage efficiency of volunteer clients.

## 1. はじめに

近年，民生用コンピュータの高性能化と低価格化により，高性能な個人用コンピュータやゲーム機器が一般家庭に広く普及している．しかし，それらの計算資源は常に利用されているわけではなく，稼働時間の大半は遊休状態となっている．そこで，これらの遊休状態にある計算資源の計算能力を集約して大規模科学技術計算を行う，ボランティアコンピューティングが提案されている．SETI@home[1] や Folding@home[2] などのボランティアコンピューティングプロジェクトでは，膨大な数の遊休計算資源を活用することで数百 TFlops を超える計算性能を実現している．

ボランティアコンピューティングでは，プロジェクトに対して計算資源を提供する者をユーザと呼び，ユーザによって提供された計算資源をクライアントと呼ぶ．計算資源の提供はユーザの有志によって行われるものであり，各クライアントはプロジェクトから配布されたタスクの計算に責任を持つわけではない．このため，クライアントはボランティアコンピューティングプロ

†1 東北大学サイバーサイエンスセンター
Cyberscience Center, Tohoku University
†2 東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University
†3 NTT コムウェア
NTT COMWARE

ジェクトへの自由な参加・離脱や，タスクの計算の中断を行うことが可能である．一方，ボランティアコンピューティングでは，サーバがクライアントでのタスク実行のタイミングを指示したり，実行中のタスクを管理したりすることができない．これら 2 つの制約により，ボランティアコンピューティングではタスクの計算が一定時間以内に完了することを保証できない．ボランティアコンピューティングでは，タスクの計算結果が長期間に亘って返却されない場合，そのタスクを失敗とみなし異なる計算資源へ再割り当てする．このため，計算効率の低下が生じ，高効率な分散計算の実現が困難であるという問題がある．このようなボランティアコンピューティングの問題を解決するためには，プロジェクトサーバにおけるスケジューリングの改善だけではなく，クライアントでのタスクの計算を管理する適切なスケジューリングを同時に実現する必要がある．

我々はこれまでに，ボランティアコンピューティング環境においてクライアントが行うべきスケジューリングとして，タスク締切時間を考慮したスケジューリング手法を提案してきた[3]．提案手法では，クライアントが複数プロジェクトに参加することでタスクを締め切りまでに計算完了できない状況を考慮し，締め切りに間に合わないタスクをキャンセルし，締め切りに間に合うタスクのみを選択して計算する．

一方で，クライアントが複数のプロジェクトに参加している場合，プロジェクト間での計算能力の分配による計算時間の増加を考慮せずにタスクの取得が行われる．そのため，クライアントが締め切りまでに余裕が無いタスクを取得してしまう問題が生じる．無駄なタスク取得は，ネットワークやサーバに大きな負荷をかける一方，大部分のタスクは計算されずにキャンセルされるため，ボランティアコンピューティング環境における計算性能の低下を引き起こす．この問題に対し，我々が提案した締め切りに間に合わないタスクをキャンセルする手法だけでは，クライアントがサーバから計算が完了する見込みが無いタスクを取得することを防ぐことができない．そのため，クライアントのサーバからのタスク取得処理において，無駄なタスク取得を抑制するための仕組みが必要である．

本論文は，ボランティアコンピューティングにおける計算資源の利用効率を高めるため，クライアントにおける締切時間を考慮したタスクスケジューリング手法を提案する．提案手法は，各タスクが必要とする計算時間と締切時間の関係を考慮したタスク選択を行うとともに，締切時間内に計算を完了できるタスク数を予測し，その数だけサーバからタスクを取得する．また，シミュレーションによる性能評価から，本手法によりボランティアコンピューティングにおけるクライアントの利用効率を改善できることを明らかにする．

以下，第 2 節では，ボランティアコンピューティングを実現するデファクトスタンダードな基盤ミドルウェアである BOINC について説明し，クライアントが行うスケジューリングの問題点について述べる．第 3 節では，締切時間を考慮したタスクスケジューリング手法を提案する．第 4 節では，シミュレータを用いた提案手法の有効性の評価を行う．最後に，第 5 節で本論文のまとめを行い，今後の課題について述べる．

## 2. Berkeley Open Infrastructure for Network Computing

### 2.1 BOINC プラットフォーム

BOINC[4] は，現在最も一般的に利用されているボランティアコンピューティングミドルウェアである．BOINC はプロジェクトを管理しタスクを配布するプロジェクトサーバ[5] と，サーバからタスクを受け取り計算を行う多数の計算クライアント[6] から構成される．BOINC を用いたボランティアコンピューティングの処理手順を以下に示す．

(1) クライアントがサーバにタスクのリクエストを送信
(2) サーバがクライアントにタスクを配布
(3) クライアントがタスクを計算
(4) クライアントがサーバに計算結果を提出

ボランティアコンピューティングによる分散計算では，クライアントマシンのシャットダウンなどによるタスクの計算中断によって，計算結果がいつまでもプロジェクトサーバに提出されなくなる，タスク消失の可能性がある．そこで BOINC では，各タスクに計算結果提出の締め切りが定められている．締切時間までに計算結果がプロジェクトサーバへ提出されない場合，プロジェクトサーバはそのタスクの計算が失敗したとみなし，そのタスクを他のクライアントへ再配布する．これにより，特定のタスクがいつまでも完了しない問題を解決している．

一方，タスクの締切時間の決定は，ボランティアコンピューティングの計算効率に係わる重要な問題である．タスクの計算時間に対して締切時間が短く設定された場合，締め切り後にクライアントから提出された計算結果をサーバは受け取らないため，タスクの計算に費やされたそのクライアントの計算能力が無駄になる．その結果，ボランティアコンピューティングによ

り多数のクライアントを利用した計算を行ったとしても，実際に有効な計算を行えるクライアントの台数は少なくなり，計算資源の利用効率の低下が生じる．逆に，タスクの計算時間に対して締切時間が長く設定された場合には，クライアントのシャットダウンなどによるタスク消失の検出が遅れ，タスクの他クライアントへの再割り当て実行が適切に行われない問題がある．特に，ボランティアコンピューティングのようにクライアントが頻繁に参加離脱を繰り返す環境では，他クライアントへタスク再割り当てを適切に行うことがより困難になるため，計算性能の低下や各タスクの計算が完了するまでのターンアラウンドタイムが増加する恐れがある．また，ボランティアコンピューティング環境には様々な計算性能のクライアントが存在し，タスクの計算時間も一定ではない．そのため，あるクライアントでの計算時間を基に適切な締切時間を設定したとしても，違うクライアントにおいても同様に適切な締切時間であるとは限らない．

### 2.2 BOINC におけるスケジューリング

Anderson らは文献[4] において，ボランティアコンピューティングでは 1 台のクライアントが同時に複数のプロジェクトへ参加することで，計算資源の利用効率を改善できると述べている．あるプロジェクトのサーバがメンテナンス等の理由によって停止している場合，クライアントはサーバからタスクを取得することができない．従って，1 つのプロジェクトにしか参加していないクライアントでは，タスクを取得できない間，計算能力を有効活用できない．一方，複数プロジェクトに参加しているクライアントでは，メンテナンス中でない他のプロジェクトからタスクを取得することで，計算を続行することが可能である．

一方，クライアントが複数のプロジェクトに参加している場合，どのプロジェクトのタスクをどのような順番で実行するかが問題となる．このような複数のプロジェクトのタスクを計算するためのスケジューリング問題に対し，従来の BOINC は 4 つのポリシーを用いたスケジューリングを行う[7),8)]．

まずクライアントは，タスク取得ポリシーに従って，各プロジェクトからタスクを取得する．クライアントは内部のワークキューに複数のタスクをあらかじめ蓄えておくことで，一時的にネットワークにアクセスできなくなっても，遊休計算能力を用いたタスク計算を継続することができる．ワークキューの長さは蓄えられているタスクの合計計算時間で表され，各クライアントのワークキューには最小ワークキュー長が設定される．もし，ワークキュー長が最小ワークキュー長を下回った場合，クライアントはサーバに対し最小ワークキュー長を充足するのに必要なタスクの計算時間 (以下，要求計算時間) を通知する．またタスク取得ポリシーでは，各プロジェクトのタスクにこれまでに割り当てられた CPU 時間の統計値 (*long-term debt)* を保持している．この統計値をもとに，これまでに過剰に計算能力が割り当てられているプロジェクトからのタスク取得を制限することで，クライアントの持つ計算能力を複数のプロジェクト間で公平に分配する．

サーバは，クライアントからのタスク取得要求を受け取ると，タスク送信ポリシーに従ってタスクを選択し，クライアントへ送信する．タスク送信ポリシーは，未配布のタスクをそのクライアントで計算した時の計算時間を予測し (以下，予想計算時間)，予想計算時間の合計が要求計算時間以上となるような 1 つ以上のタスクを選択する．次に，選択したタスクが締切時間までに計算を完了し，且つクライアントが既にワークキューに保持しているタスクについても締切時間までに計算を完了できる場合に，そのタスクをクライアントへ送信する．

次にクライアントは，CPU スケジューリングポリシーに従って，ワークキューからタスクを選択し，計算を行う．CPU スケジューリングポリシーには，タスク毎の締切時間情報を参照し，取得しているタスクの中から締め切りの最も近いタスクを選択して計算を行う，Earliest Deadline First[9)](以下，EDF) スケジューリングが用いられている．また，同一のプロジェクトに属する複数のタスクがある場合には，FCFS(First-Come First-Serve) によるスケジューリングが行われる．

クライアントにおけるタスクの計算時間の予測には，タスク完了予測ポリシーが用いられる．タスク完了予測ポリシーでは，基本ベンチマークから得られたクライアントの計算能力と，各プロジェクトがタスクに設定した演算回数から，そのクライアントにおけるタスクの計算時間が予測される．また，タスクの計算時間の予測値と，実際の計算時間とには誤差が生じる可能性が高いため，プロジェクト毎に補正係数 (*Duration Correction Factor:* 以下，*DCF)* を持つ．タスク完了予測ポリシーは，タスク計算時間の予測値に対し *DCF* を乗じることで予測誤差を最小化する．また，タスクの計算完了時には，計算時間の予測値と実際の計算時間から，*DCF* の動的な更新が行われる．

### 2.3 BOINC におけるクライアントスケジューリングの問題点

ボランティアコンピューティングにおいて，各プロジェクトは互いに独立して運営されている．そのため，

プロジェクトサーバからは，自らのプロジェクトに参加しているクライアントが他のプロジェクトにも参加しているかどうかを知ることはできない．そのため，BOINC のプロジェクトサーバは，クライアントが 1 つのプロジェクトにしか参加していないと想定して，タスクの計算結果返却までの締切時間を設定し，クライアントへタスクを送信する．しかしクライアントが複数プロジェクトに参加している場合，その計算能力は各プロジェクトに対して均等に分配されるため，プロジェクトには想定していたよりも低い計算能力しか割り当てられない．そのため，タスクの計算を完了させるためはサーバが予想したよりも長い時間が必要であり，締め切りまでに計算を完了できないタスクが発生する．その結果，締め切りまでに計算を完了できないタスクに費やしたクライアントの計算資源が無駄となり，ボランティアコンピューティングプロジェクト全体の計算効率が低下する．

一方，クライアントは自分が参加している複数のプロジェクトに関する情報を参照することが可能である．そのため，クライアントが複数のプロジェクトに参加するボランティアコンピューティング環境において高い計算性能を実現するためには，プロジェクトサーバによるスケジューリングのポリシーよりも，クライアントがスケジューリングを行う際の，タスク取得ポリシーと CPU スケジューリングポリシーが重要である．

我々はこれまでに，ボランティアコンピューティングにおける CPU スケジューリングポリシーとして，タスク締切時間を考慮したスケジューリング手法を提案してきた[3]．BOINC が用いている CPU スケジューリングポリシーでは，タスクの締切時間のみを用いた EDF スケジューリングしか行っていない．そのため，締切時間までに計算が完了できないことが明らかなタスクでも計算を継続して行うため，クライアントの計算能力の浪費が問題となる．提案手法では，締め切りに間に合わないタスクの計算をキャンセルし，締め切りに間に合うタスクのみを選択して計算することで，クライアントの計算能力の回収効率を向上させている．

一方，タスク締切時間を考慮したスケジューリング手法では，クライアントが締め切りまでに余裕が無いタスクを過剰に取得してしまう．この問題は，クライアントが BOINC のタスク取得ポリシーに基づき，プロジェクト間での計算能力の分配による計算時間の増加を考慮せずにタスクを取得することが原因で生じる．このようなクライアントによる過剰なタスク取得は，ネットワークやサーバに大きな負荷をかける一方，大部分のタスクは計算されずにキャンセルされるため，ボランティアコンピューティング環境における計算性能の低下を引き起こす．このことから，締め切りに間に合わないタスクをキャンセルする手法である CPU スケジューリングポリシーだけでは，クライアントがサーバから計算が完了する見込みが無いタスクを取得することを防ぐことができない．このため，クライアントのサーバからの過剰なタスク取得を抑制するタスク取得ポリシーが必要である．

以上の理由から，クライアントが複数のプロジェクトに参加するボランティアコンピューティング環境において，クライアントの利用効率を高め，高性能な分散計算を実現するため，クライアントスケジューリングを実現する必要がある．本論文では，クライアント自身がタスクの進捗状況を常に監視し，締め切りまでに計算結果を返却可能かどうかを動的に判断するタスク取得ポリシーおよび CPU スケジューリングポリシーを提案する．

## 3. ボランティアコンピューティングのための締切時間を考慮したタスクスケジューリング

本論文では，ボランティアコンピューティングにおける締切時間を考慮したタスクスケジューリング手法を提案する．まず，提案するタスクスケジューリング手法について，その概要を説明する．次に，締切時間を考慮した CPU スケジューリングとタスクスケジューリングを実現する，2 つのスケジューリングポリシーについて述べる．

### 3.1 クライアントにおけるスケジューリング手続き

本論文では，BOINC のクライアントスケジューラによって行われる CPU スケジューリング処理とタスク取得処理に対し，締め切りを考慮したスケジューリングポリシーを導入する．提案するタスクスケジューリングの処理手順を図 **1** に示す．提案手法は，図 1 の破線上側に示すタスク取得処理と，破線下側に示す CPU スケジューリング処理によって構成される．

クライアントはまず，各プロジェクトからタスクを取得し，ワークキューへ充填する．次にワークキュー内からタスクを選択し，計算を行う．一定時間が経過，あるいはタスクの計算が完了した後，クライアントは再びタスク取得処理に戻り，同様の手順を繰り返す．また，クライアントはタスク毎の計算の進捗や，締め切りまでに計算が完了したタスク数や完了しなかったタスク数などの統計情報を収集する．タスク取得処理

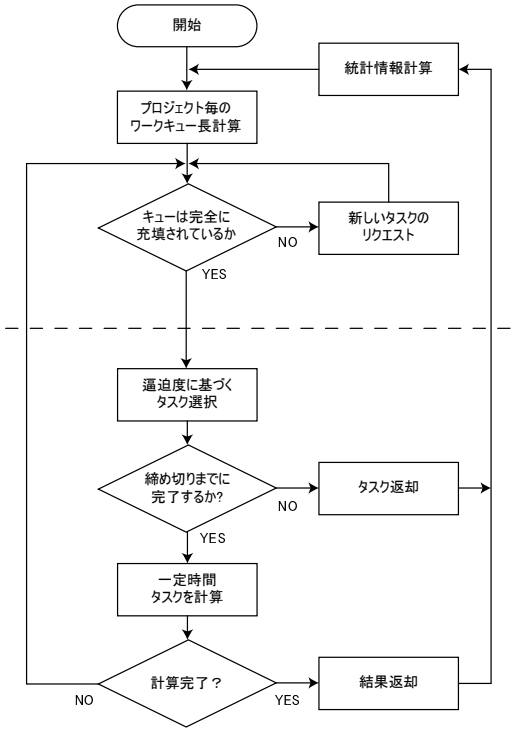



図 1　提案スケジューリング手法の処理手順

では，これらの収集された統計情報を用いて，タスク取得の最適化を行う．

### 3.2　締め切りを考慮した CPU スケジューリングポリシー

本節では，タスク完了時間予測に基づき，締め切りに間に合うタスクのみを選択計算する CPU スケジューリングポリシーについて述べる．BOINC において，締め切りを過ぎたタスクの計算を続けることは計算能力の浪費を意味する．そこで，締め切りを過ぎた，あるいは過ぎることが予測されるタスクをスケジューリングの対象から除外し，その分の CPU 時間を締め切りに間に合うタスクの計算へ振り替える．これにより，計算中に締切時間を過ぎてしまうタスクの数を減らしたうえで，ボランティアコンピューティングによる遊休計算能力の回収効率を向上させることが可能となる．

初めに，タスク $i$ の逼迫度 $Tight_i$ を，式 (1) と定義する

$$Tight_i = Remain_i/(Deadline_i - Now) \quad (1)$$

ここで，$Remain_i$ はタスク $i$ の計算が完了するまでの残り予想時間，$Deadline_i$ はタスク $i$ の締め切り時刻，$Now$ は現在の時刻である．逼迫度 $Tight_i$ は，タスクの締め切りまでの余裕の無さを表しており，1 を超えるとそのタスクは締め切りまでに計算が完了しないことを示している．

まず CPU スケジューリングでは，ワークキュー内のタスクに対して逼迫度の計算を行い，逼迫度が最も大きなタスクを選択する．次に，選択したタスクの逼迫度が 1 以下であればそのタスクの実行を決定し，逼迫度が 1 を超える場合にはそのタスクのキャンセル処理を行う．タスクがキャンセルされた場合，そのタスクに関する統計情報の収集とタスク取得処理が行われ，再び逼迫度によるタスク選択が行われる．最終的に，完了見込みがあるタスクが選択されるまで，上記の手順を繰り返す．タスクの選択が完了すると，タスクの計算が開始される．

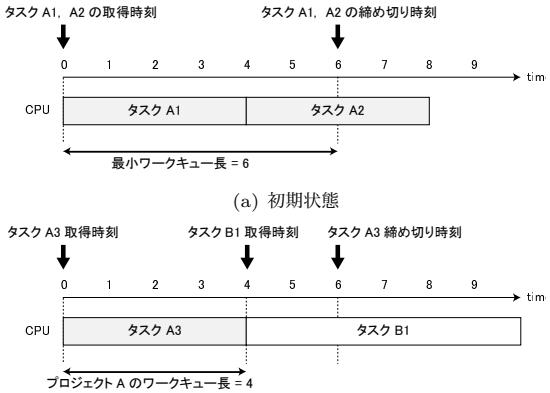
次に，計算開始から一定時間経過後，CPU スケジューリングによる実行タスクの切り替えを行う．タスクがすでに完了していた場合は，その結果をプロジェクトサーバに返却し，統計情報の収集を行った後，タスク取得処理を開始する．タスクがまだ完了されていなかった場合には，ワークキューの充填度を確認したのち，逼迫度によるタスク選択に戻り，同様の手順を繰り返す．

ここで，同一プロジェクトに属し，計算時間も締切時間も同一のタスクが存在する場合，逼迫度によるタスク選択ではこれらのタスクを交互に選択してしまう．そこで，同一プロジェクトに属するタスクに関しては，すでに計算が開始されているタスクを優先することとする．

提案手法では，計算がキャンセルされたタスクは，即座にプロジェクトサーバに返却される．プロジェクトサーバはタスクが返却されると，そのタスクの締め切りを再設定したうえで他のクライアントに対して配布し直す．提案手法では，締切時間まで待つことなく，より迅速にタスクの再配布を行うことができる．そのため，タスク計算の平均ターンアラウンドタイムを短縮し，プロジェクトのタスク計算効率を高めることが可能である．

### 3.3　締め切りを考慮したタスク取得ポリシー

BOINC が用いるタスク取得ポリシーは，各タスクが計算開始されるまでの待ち時間を考慮せず，ワークキューに蓄えられているタスクの計算時間が最小ワークキュー長を下回った場合にのみプロジェクトからタスクを取得する．ここで図 **2(a)** に示すように，時刻 0 においてクライアントが，プロジェクト A から計算時間が 4 のタスク A1 と A2 を取得し，最小ワークキュー長の 6 を充足する場合を考える．この場合，タスク A1 の計算は締切時間までに完了するが，タスク




(a) 初期状態

(b) 最適化後

図 **2** ワークキュー長の最適化

A2 の計算は締切時間までに完了しない．ここで，タスク A1 は取得後すぐに計算が開始されるため待ち時間は 0 であるが，タスク A2 の計算開始までの待ち時間は 4 であり，計算開始までの待ち時間が原因でタスク A2 の計算が締切時間までに完了しないことがわかる．タスクの計算を締切時間までに完了させるためには，タスクの計算開始までの待ち時間とタスクの計算時間の合計が，締め切りまでの残り時間より短い必要がある．しかし，従来の BOINC のタスク取得ポリシーはクライアントの最小ワークキュー長しか考慮しておらず，また最小ワークキュー長もタスクの締切時間や計算開始までの待ち時間を考慮した値が設定されていない． 本節では，タスクの計算開始までの待ち時間増加による締め切り超過を検出し，プロジェクト毎にワークキュー長に上限を動的に設定することで，プロジェクトサーバからのタスク取得を制御するタスク取得ポリシーについて述べる．

まず提案手法では，タスクの計算が完了またはキャンセルされる度に，**表 1** に示す統計情報を集計する．これらの統計情報は，各プロジェクトに固有の情報として記録される．タスクの平均待ち時間は，タスクをサーバから取得した時刻と計算を開始した時刻の差分であり，プロジェクト毎のワークキュー長の動的設定のためのスケジューリング情報として用いる．

**図 3** に，クライアントが行うプロジェクト毎のワークキュー長の最適化手順を示す．ここで，$Qsize_i$ はプロジェクト $i$ に関する最小ワークキュー長である．$EstimatedExecTime_{ave,i}$ はプロジェクト i のタスクの予測計算時間の平均であり，プロジェクト毎に集計した統計情報を利用して計算を行う． まずクライアントは，表 1 に示す統計値を用いて，締め切りに間に合ったタスクの平均待ち時間と，締め切りに間に合わなかったタスクの平均待ち時間をプロジェクト毎に比較する．もし 2 つの平均待ち時間の差が閾値以上であれば，クライアントはタスクの過剰取得が行われていると判断し，最小ワークキュー長の再設定を行う．ここで，図 2(a) から分かるように，2 つのタスクを連続して計算した場合の待ち時間の差は，先に計算が行われたタスクの計算時間と等しい．また，連続して計算するタスクが 3 つ以上の場合，あるいは CPU スケジューリングによって他のプロジェクトのタスクが途中で計算された場合，締め切りに間に合ったタスクの待ち時間と締め切りに間に合わなかったタスクの待ち時間の差はさらに長くなる．そのため，平均待ち時間差の閾値としてプロジェクト毎の平均タスク計算時間 $EstimatedExecTime_{ave,i}$ を用いることで，最小ワークキュー長の再設定を適切に実施できる．次にクライアントは，締め切りに間に合ったタスク数と間に合わなかったタスク数の比率から，最小ワークキュー長を計算する．クライアントは最後に，計算された最小ワークキュー長を再設定し，余分なタスクを取得しないようにする．

図 2(a) に対して提案手法を適用した場合の結果を，**図 2(b)** に示す．図 2(b) では，プロジェクト A のワークキュー長が 4 に設定され，時刻 0 においてタスク A3 のみを取得している．このように，提案手法を用いることで，タスク A2 の様な締め切りまでに計算が完了しないタスクの取得を回避することが可能である．

## 4. 性 能 評 価

本節では，提案する 2 つのスケジューリングポリシーを BOINC のクライアントスケジューラへ実装し，提案手法の有効性をシミュレーションによって評価する．シミュレーションには，BOINC のソースコードに付属するクライアントシミュレータ[8),10)] を用いる．クライアントシミュレータは，仮想的なボランティアコンピューティング環境における，クライアントスケジューラ 1 台の動作を再現する．

表 **1** プロジェクト $i$ に関する統計値

| 統計値名 | 説明 |
|---|---|
| $N_{succeeded,i}$ | 締め切りに間に合ったタスク数 |
| $N_{missed,i}$ | 締め切りに間に合わなかったタスク数 |
| $WaitTime_{succeeded,i}$ | 締め切りに間に合ったタスクの平均待ち時間 [秒] |
| $WaitTime_{missed,i}$ | 締め切りに間に合わなかったタスクの平均待ち時間 [秒] |
| $ExecTime_{ave,i}$ | タスクの平均計算時間 [秒] |

```
for all Project_i do
  if N_missed,i ≠ 0 then
    if WaitTime_missed,i − WaitTime_succeeded,i > ExecTime_ave,i then
      Qsize_i ← Qsize_i * N_succeeded,i / (N_succeeded,i + N_missed,i)
      WaitTime_succeeded,i ← 0, WaitTime_missed,i ← 0, N_succeeded,i ← 0, N_missed,i ← 0
    end if
    if Qsize_i < EstimatedExecTime_ave,i then
      Qsize_i ← EstimatedExecTime_ave,i
    end if
  end if
end for
```

図 3　ワークキュー長の計算

### 4.1 評価指標

提案手法の有効性を評価するための評価指標として，タスク成功数，タスク失敗数，クライアントの計算効率を用いる．

タスク成功数は締め切りまでに完了したタスク数を示し，この指標が高い値を示すほど遊休計算資源を有効に活用できていることを表している．タスク失敗数は，計算が完了した時点で締切時間を超過していたタスク数を示し，この指標が高い値を示すほど遊休計算資源の計算能力が浪費されていることを表している．

クライアントの計算効率は，ボランティアコンピューティングによってクライアントの遊休計算能力をどれだけ効率的に回収できたかを評価する．クライアントの計算効率の評価には，計算資源が稼働していた時間のうち，締切時間までに計算が完了したタスクに費やした計算時間，締切時間までに計算が完了しなかったタスクに費やした計算時間，計算資源上で BOINC のクライアントが動作していなかった時間，そしてクライアントが稼働していたにもかかわらず何もタスクを計算していなかった時間を用いる．ここで，クライアントが稼働していたにもかかわらずタスクを計算していない時間には，プロジェクトサーバからのタスク配布待ち時間や，タスクを転送する際のネットワークオーバヘッドの時間などが含まれる．これらの計算資源の計算時間の中で，締切時間までに計算が完了したタスクに費やした計算時間の割合が大きい場合ほど，計算資源のより高い効率を実現すると評価できる．

評価実験では，提案手法のほかに 2 種類のスケジューリング手法を比較対象として用いた．Round-Robin スケジューリングは，サーバから配布されたタスクを交互に計算する CPU スケジューリングポリシーである[11]．このスケジューリング手法は締め切りまでの時間を考慮しないため，締切時間が短い場合や，複数のプロジェクトに参加する場合に，著しい性能低下が予想される．EDF スケジューリングは，従来の BOINC のクライアントが行うスケジューリングであり，締め切りまでの時間が短いタスクを選択する CPU スケジューリングポリシーである．また，Round-Robin スケジューリングおよび EDF スケジューリングは，設定締め切り時間を考慮しない従来の BOINC クライアントのタスク取得ポリシーを持つ．

### 4.2 シミュレーションパラメータ

BOINC プロジェクトで用いられるシミュレーションパラメータ[10]，および文献[8] を基に，評価に用いるシミュレーションパラメータを決定する．表 **2** に評価に用いるシミュレーションパラメータを示す．

クライアントの参加するプロジェクト数は 2 とする．代表的なボランティアコンピューティングプロジェクトを参考[1),2),12)] に計算時間が平均 1 時間，標準偏差 $\sigma$=10 分の正規分布に従うタスクを持つプロジェクトと，計算時間が平均 10 時間，標準偏差 $\sigma$=100 分の正規分布に従うタスクを持つ 2 つのプロジェクトを想定する．ボランティアコンピューティング環境における締め切り時間の影響をより詳細に評価するために，本シュミレーションでは各プロジェクトのタスクに平均計算時間を 2 倍した時間を締め切りまでの時間として用いる． スケジューリング間隔は，クライアントがスケジューリングを行う間隔を表す．CPU コア数は，1 台のクライアントが同時に計算できるタスク数を表す．シミュレーション時間は，シミュレーションを行った時間の長さを表す．BOINC 稼働率は，シミュレーション時間のうち，計算資源が BOINC クライアントを動作させている時間の割合を表す．BOINC クライアントは計算資源が遊休状態の時に動作していることから，BOINC 稼働率は計算資源が遊休状態にある割合も表している．計算資源は 1 分毎に BOINC 稼働率

表 2　シミュレーションパラメータ

| パラメータ名 | 値 |
|---|---|
| 平均タスク計算時間 [時間] | 1, 10 |
| 締切時間 [時間] | 2, 20 |
| スケジューリング間隔 [秒] | 60 |
| クライアント CPU コア数 | 2 |
| シミュレーション時間 [時間] | 1,000 |
| 参加プロジェクト数 | 2 |
| BOINC 稼働率 [%] | 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 |
| 初期ワークキュー長 [秒] | 86,400 |

に応じた動作状態の遷移を行い，BOINC クライアント動作時にはすべての計算能力を BOINC のタスクに割り当てるものとする．

### 4.3　評 価 結 果

BOINC 稼働率を変化させた時の，タスク成功数の変化を図 **4** に示す．各プロジェクトが持つタスクの計算時間は一定であるため，締切時間を超過することが全くない理想的な状態であれば，BOINC 稼働率とタスク成功数は比例関係にある．タスクスケジューリングに Round-Robin を用いた場合，すべての BOINC 稼働率において，締め切りまでに計算が完了するタスクは非常に少ない．Round-Robin では，クライアントの計算能力を各プロジェクトに対して均一に分配する．そのため，すべてのタスクの計算時間が 2 倍になり，ほとんど場合において締切時間までにタスクの計算が完了しない結果となる．一方，提案手法および EDF を用いた場合，BOINC 稼働率に比例してタスク成功数が増加している．これは，提案手法および EDF では，各プロジェクトに対するクライアントの計算能力の分配比率を締切時間に応じて変化させ，締切時間までに計算が完了する見込みのあるタスクから順番に，計算を確実に完了させていくためである．提案手法を用いた場合と EDF を用いた場合の結果を比較すると，すべての BOINC 稼働率において提案手法の方が高いタスク成功数を示している．また EDF を用いた場合では，BOINC 稼働率が 80%以下の時に BOINC 稼働率増加に対するタスク成功数の向上が低い．それに対して，提案手法ではタスク成功数が BOINC 稼働率の向上に合わせて急速に増加している．

次に，BOINC 稼働率を変化させた時の，タスク失敗数の変化を図 **5** に示す．結果より，Round-Robin を用いた場合および EDF を用いた場合，計算が完了した時点で締め切りを超過していたタスクが非常に多いことがわかる．一方，提案手法では，締切時間までに計算が完了しないタスクをあらかじめキャンセルするため，タスク失敗数はほぼ 0 である．

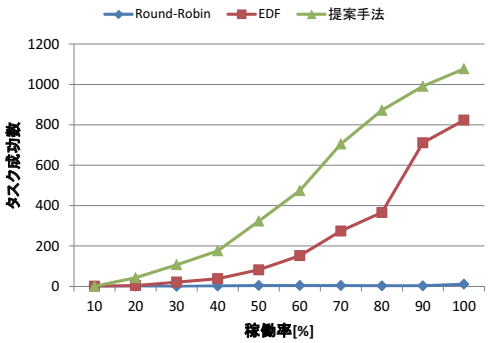



図 4　BOINC 稼働率とタスク成功数

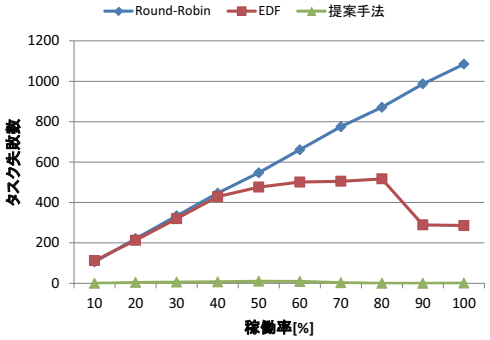



図 5　BOINC 稼働率とタスク失敗数

図 4 および図 5 に示す結果から，クライアントがボランティアコンピューティングプロジェクトに対してどれだけ貢献できるかが評価可能である．Round-Robin を用いた場合では，クライアントはプロジェクトから与えられたタスクを締め切りまでに全く完了できず，プロジェクトに対して全く貢献できていない．一方，EDF を用いた場合では，Round-Robin を用いた場合よりも多くのタスクを計算している．このことから，EDF を用いた場合のプロジェクトへの貢献度は，Round-Robin よりも大きいといえる．しかし，BOINC 稼働率が 80%以下では，タスク成功数よりもタスク失敗数の方が大きく，プロジェクトへ高い貢献をするためにはクライアントを常時稼働させている必要がある．それに対し提案手法を用いた場合，どの BOINC 稼働率においても，タスク失敗数よりもタスク成功数の方が大きい．そのため，提案手法は他の 2 手法に比べて，与えられたタスクをより確実に計算することが可能であり，プロジェクトへ常に高い貢献をすることが可能である．

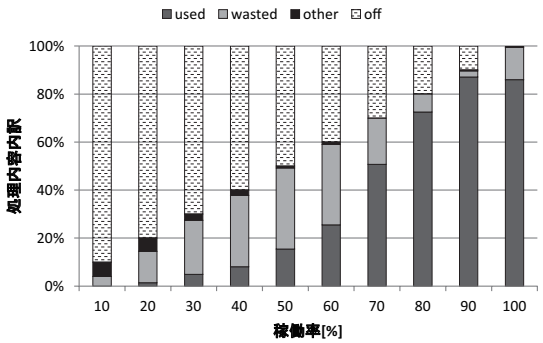
次に，BOINC 稼働率を変化させた場合の計算資源の処理内訳を図 **6** に示す．図の *used* は締切時間までに計算が完了したタスクに費やした計算時間を表し，

*wasted* は締切時間までに計算が完了しなかったタスクに費やした計算時間を表す．*off* は，クライアントが起動していなかった計算時間を表す．また *other* は，クライアントは起動していたが，タスクの計算は行われていなかった時間を表す．すべての手法に共通する傾向として，BOINC 稼働率が高くなるほど，計算資源の利用効率が高くなる．しかし，Round-Robin を用いた場合，BOINC 稼働率が 100%でも計算資源の利用効率が 5%しかなく，EDF を用いた場合でも利用効率は最大で 40%程度である．それに対して，提案手法を用いた場合では BOINC 稼働率が 70 %の時点で 50 %以上の計算資源の利用効率であり，利用効率は最大 80%以上を実現している．このことから，提案手法を用いることにより，ボランティアコンピューティングプロジェクトはクライアントの持つ計算能力を高い効率で回収可能であることがわかる．

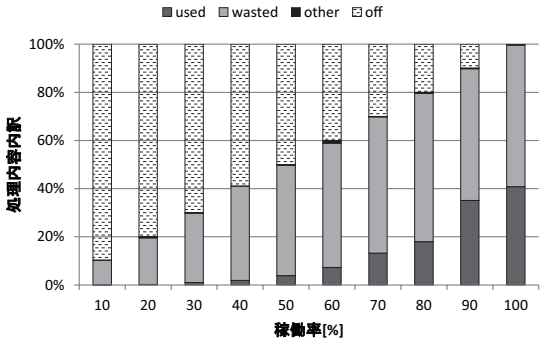
最後に，プロジェクト間での計算資源の used 時間の分配率を図 **7** に示す．資源分配率が 100%の時には，平均タスク計算時間が 1 時間のタスクのみが計算されたことを示す．逆に，資源分配率が 0%の時には，平均タスク計算時間が 10 時間のタスクのみが計算されたことを示す．資源分配率が 50%であれば，クライアントの計算能力は 2 つのプロジェクトに対し均等に分配されたことを示している．EDF を用いた場合には，資源分配率は常に 100%であり，片方のプロジェクトでは計算が全く進まないことが確認できる．一方，提案手法では BOINC 稼働率が 70%以上の時には，資源分配率が 50%に漸近している．これらの結果から，クライアントが複数のプロジェクトに参加する場合，従来手法では計算時間が短いタスクを持つプロジェクトにばかり計算資源を提供してしまう．一方，提案手法は複数のプロジェクトに対して常に公平な資源分配が可能であることがわかる．そのため，クライアントが複数のプロジェクトに参加する環境では，提案手法がより有効な手段であると言える．

## 5. おわりに

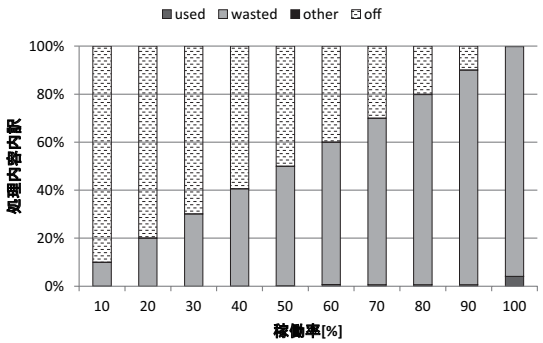
本論文では，クライアントが複数プロジェクトに参加し，タスクを締め切りまでに計算しきれない状況を考慮し，タスクの取得と，ワークキュー内のタスクを適切に選択計算するスケジューリングポリシーを提案した．またシミュレーションによる評価結果より，本手法によって締め切りに間に合わないタスクに計算資源を浪費することを防ぎ，締め切りまでに計算が完了するタスク数を増加させ，ボランティアコンピューティング環境における効率的な資源利用が可能であること




(a) 提案手法




(b) EDF




(c) Round-Robin

**図 6** BOINC 稼働率が変化した時の計算資源の処理内訳

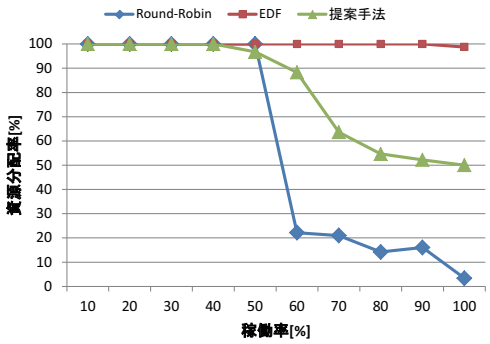



図 7　BOINC 稼働率とプロジェクト間での計算資源分配率

を示した．

今後は，スケジューラによって返却と判断されたタスクをサーバに直接返却するのではなく，クライアント間で直接タスクの移動を行うことを検討する[13),14)]．これにより，サーバへの負荷集中を回避し，より大規模なボランティコンピューティング環境の実現を目指す．

謝辞　本研究の一部は，文部科学省科研費特定領域研究 (18049003)，および総務省特定領域重点型研究開発 (061102002) の支援を受けた．

## 参　考　文　献

1) Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M. and Werthimer, D.: SETI@home: An experiment in Public-Resource Computing, *Communications of the ACM*, Vol. 45, No. 11, pp.56–61 (2002).
2) Folding@home distributed computing: `http://folding.stanford.edu/`.
3) 村田善智，遠藤聡明，滝沢寛之，小林広明：ボランティアコンピューティングの高効率化のためのクライアントレベルスケジューリング，情報科学技術フォーラム，pp.165–172 (2009).
4) Anderson, D. P.: BOINC: A System for Public-Resource Computing and Storage, *5th IEEE/ACM International workshop on Grid Computing*, pp.4–10 (2004).
5) Anderson, D.P., Korpela, E. and Walton, R.: High-Performance Task Distribution for Volunteer Computing, *First IEEE International Conference on e-Science and Grid Technologies*, pp.196–203 (2005).
6) Anderson, D.P., Christensen, C. and Allen, B.: Designing a runtime system for volunteer computing, *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, ACM (2006).
7) Anderson, D. and VII, J.M.: Local Scheduling for Volunteer Computing, *Workshop on Large-Scale, Volatile Desktop Grids (PCGrid 2007) held in conjunction with the IEEE International Parallel & Distributed Processing Symposium (IPDPS)* (2007).
8) Kondo, D., Anderson, D. P. and Vii, J. M.: Performance Evaluation of Scheduling Policies for Volunteer Computing, *e-Science and Grid Computing, International Conference on*, Vol.0, pp.415–422 (2007).
9) Kargahi, M. and Movaghar, A.: Non-preemptive earliest deadline first scheduling policy: a performance study, *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp.201–208 (2005).
10) SourceCode – BOINC – Trac: `http://boinc.berkeley.edu/trac/wiki/SourceCode`.
11) Furnkranz, J.: Round robin classification, *Machine Learning Research*, Vol. 2, pp. 721–747 (2002).
12) Toth, D. and Finkel, D.: Increasing the amount of work completed by volunteer computing projects with task distribution policies, *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp.1–8 (2008).
13) 村田善智，　稲葉勉，滝沢寛之，小林広明：大規模計算環境における分散協調型負荷分散手法，情報処理学会論文誌特集号“新しいパラダイムの中での分散システム/インターネット運用・管理”，Vol.3, No.49, pp.1214–1228 (2008).
14) 村田善智，石杜佑記，滝沢寛之，小林広明：動的負荷分散機能を持つ高性能ボランティアコンピューティングの実現，情報処理学会論文誌，　Vol. 2, No.52, pp.401–414 (2011).