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

2008－ARC－177 (11)
2008－HPC－114 (11)
2008／3／5

Spot-checkingによるボランティアコンピューティング実行時間の最小化

渡邊 寛 福士 将 堀口 進
東北大学 大学院情報科学研究科

概 要

本稿では，誤った計算結果を返す妨害者が存在するボランティアコンピューティング (VC) 環境において，VCの実行時間を最小にするようなspot-checkingの実行頻度を，計算開始前に推定する手法を提案する．妨害者の検出手法であるspot-checkingはVCの計算信頼性を高めるために必要であるが，その実行頻度であるチェック率 q には，実行時間を最小とするような最適値 q_{opt} が存在する．本手法は，実行時間の推定値を q の関数として表現することで，計算開始前に q_{opt} の推定値を得ることができる．シミュレーションにより，実行時間の推定から得られた q_{opt} の推定値を用いることで，実行時間を誤差1 %以内で最小化できることが分かった．

Spot-checking to minimize the Computation Time in Volunteer Computing

Kan Watanabe, Masaru Fukushi, Susumu Horiguchi
Graduate School of Information Sciences, TOHOKU University

Abstract

This paper proposes an optimization technique for spot-checking to minimize the computation time of volunteer computing (VC) systems with malicious participants who return erroneous results. There is an optimal value for the spot-check rate q. The key idea is to represent the mathematical expectation of the computation time as a function of rate q to obtain an estimate of the optimal spot-check rate q_{opt} before the computation. It is shown by Monte Carlo simulations that the proposed technique can always obtain an approximate estimate of q_{opt} and minimize the computation time with an uncertainty of 1 %.

1. はじめに

ボランティアコンピューティング（VC）は，多数のボランティア参加者による計算資源の提供に基づく，大規模並列計算環境の実現方法である．VCでは，インターネットに接続されたPCなどを持つ一般の参加者から，計算資源の遊休CPUサイクルを無償で提供してもらうことで，高性能な並列計算環境を安価に構築することができる．VCが世界規模で運用されている計算プロジェクトの例として，SETI@home[1]やdistributed.net[2]などが挙げられる．

VCでは，ボランティアにより提供される各計算資源（ワーカ）が，小規模に分割された計算（ジョブ）をそれぞれ独立に実行し，計算結果を管理ノード（マスタ）に返却する．参加者数が多いほどVCシステム全体の計算性能は高くなるが，ボランティアの参加に制限を設けない場合は，誤った計算結果を故意に返すような，悪意ある者（妨害者）が参加者の中に混じることがある．妨害者の数が多い場合は計算結果の信頼性が著しく低下してしまうため，何らかの妨害者対策を行う必要がある．

基本的な妨害者対策手法として，VCミドルウェアBOINC[3]で用いられているvotingがある．これは，同じジョブを複数のワーカに計算させて，冗長に計算結果を集める手法である．VCミドルウェアBayanihan[4]では，Sarmentaにより，妨害者の検出手法であるspot-checkingを用いることでvotingの効率を改善した手法[5]が提案されている．この手法では，spot-checkingの通過回数を用いて各ワーカの信頼度を評価する．信頼度をもとに各ジョブの終了に必要な冗長度を動的に算出することで，冗長なジョブをなるべく実行しないようにすることができる．

Sarmentaの手法[5]では，妨害者対策のために信頼度を導入したことで，spot-checkingを行う頻度であるチェック率 q が，全体の実行時間に大きな影響を与えることに注意しなければならない．しかし従来，どのような q の値を用いるべきかについて十分な議論がなされていなかったため，ワーカ数やジョブ数といったパラメータによらず，$q = 0.1$ などの固定値が用いられていた[5]．その結果，パラメータによっては，q の

最適値 q_{opt} を用いた場合と比べて，全体の実行時間が必要以上に大きくなってしまうという問題点があった．

本稿では，spot-checking を用いた妨害者対策[5] の実行時間を最小化するために，与えられたパラメータに応じたチェック率 q の最適値 q_{opt} を推定する手法を提案する．本手法は，実行時間の推定値を，与えられたパラメータと q の関数として表現することで，計算開始前に q_{opt} の推定値 q_{est} を得ることができる．

以下，2 節で本稿が扱う VC のモデルと妨害者対策について述べ，3 節で実行時間の推定値 $T_{expect}(q)$ と q_{est} の導出方法を示す．4 節でシミュレーションにより $T_{expect}(q)$ と q_{est} の精度評価を行う．最後に，5 節で本稿の結論を述べる．

2. VC システム

2.1 計算モデル

VC の計算モデルとして，ワークプールを用いるマスタワーカモデルを仮定する．これは，VC やグリッドコンピューティングなど，インターネットを用いた並列計算システムで広く用いられるものである．図 1 に，想定する VC 環境を示す．このモデルの詳細は次の通りである．

- マスタ 1 台と，ボランティアにより提供されるワーカ W 台で，計算プロジェクトを実行する．
- 計算プロジェクトは，N 個の独立したジョブより成る．
- マスタはアイドル状態のワーカに対してジョブを 1 つ配布し，その計算結果（リザルト）を回収する．この際，ワーカ間の通信は発生しない．

全ワーカ W のうち，妨害者存在率を f として，$f \times W$ 台が妨害者であるとする．妨害者が誤ったリザルトを返す確率を，妨害率 s とする．s はマスタにとって未知の値である．

マスタは，ジョブの実行管理を行う．各ジョブは，ワーカからリザルトが返却され，それがマスタによって受け入れられた場合に終了する．N 個の全てのジョブの終了をもって計算プロジェクトの終了とし，それまでに要した時間を実行時間 T とする．また，誤ったリザルトによって終了したジョブを誤りジョブと呼ぶ．計算プロジェクトの終了時において，N 個のジョブのうち，誤りジョブの割合を誤り率 ϵ と定義する．

妨害者対策を行わない場合，1 つのジョブに対して 1 台のワーカでリザルトを 1 つ生成するだけなので，リザルトの誤りが直接ジョブの誤りとなってしまう．各ワーカの性能やジョブの計算量が均一の場合，誤り率 ϵ は $\epsilon = (N \times f \times s)/N = f \times s$ となる．すなわち，妨害者の数が多いほど誤り率 ϵ が大きくなり，計算結果の信頼性は著しく低下してしまう．

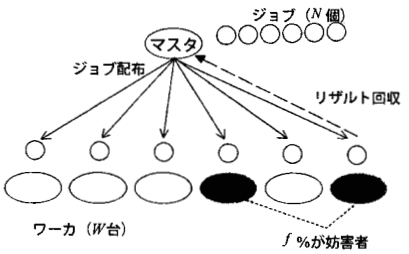


図 1　想定する VC 環境

表 1　基本的な妨害者対策手法

voting	1 つのジョブに対して複数個のリザルトを集め，多数決などを行う
spot-checking	予め正しい答えが分かっているチェック用ジョブをワーカに配布して，対象ワーカが妨害者かどうかチェックを行う
backtracking	妨害者と判断されたワーカがそれまでに生成したリザルトを全て無効化する

2.2 妨害者対策

1 つのジョブに対して複数のワーカからリザルトを集めて冗長性を持たせたり，妨害者を検出し排除するなどして，誤り率 ϵ を小さくすることを妨害者対策と呼ぶ．基本的な妨害者対策手法を表 1 に示す．

Sarmenta[5] は，より効率よく妨害者対策を行うために表 1 に示す手法を組み合わせ，信頼度評価に基づく妨害者対策と呼ばれる手法を提案した．この手法では，ワーカやリザルトなどの VC 環境内の各要素に対して，その要素の正しさを表す条件付確率である，信頼度という指標を導入する．信頼度を用いることで，各ジョブの終了に必要な冗長計算回数を，ジョブの信頼度が閾値 θ に達するまでと定め，冗長に生成するリザルト数をなるべく小さくすることができる．また，閾値 θ は任意に設定することが可能であるため，誤り率 ϵ の期待値を，任意の許容誤り率 ϵ_{acc} 以下に保証することができる．本稿では，誤り率 ϵ を減らすために，信頼度評価に基づく妨害者対策手法を用いる．

信頼度評価に基づく妨害者対策を行う際の，信頼度の計算方法を説明する[5]．ワーカ w の信頼度 $C_W(w)$ は，spot-checking に通過した回数 k を用いて，式 (1) で表される．

$$C_W(w) = \begin{cases} 1 - \frac{f}{k} & \text{if } k \neq 0 \\ 1 - f & \text{if } k = 0 \end{cases} \quad (1)$$

ただし，spot-checking によって検出された妨害者は，新たなワーカとして VC システムに戻ってくるものと

仮定する．

リザルト r の信頼度 $C_R(r)$ は，そのリザルトを生成したワーカ w の信頼度 $C_W(w)$ と等しいものとする．

$$C_R(r) = C_W(w) \tag{2}$$

ジョブ j に対するリザルトは，値の等しいリザルトごとに複数のグループ（リザルトグループ）に分けられる．ジョブ j のリザルトが g 個のリザルトグループ $G_1, G_2, \ldots, G_a, \ldots, G_g$ に分けられたとすると，リザルトグループ G_a の信頼度 $C_G(G_a)$ は，式 (3)-(5) で計算される．

$$C_G(G_a) = \frac{P_T(G_a)\prod_{i\neq a} P_F(G_i)}{\prod_{i=1}^{g} P_F(G_i) + \sum_{n=1}^{g} P_T(G_n)\prod_{i\neq n} P_F(G_i)} \tag{3}$$

$$P_T(G_a) = \prod_{r\in G_a} C_R(r) \tag{4}$$

$$P_F(G_a) = \prod_{r\in G_a} (1 - C_R(r)) \tag{5}$$

ジョブ j の信頼度 $C_J(j)$ は，$G_1, \ldots, G_g$ の中で最大の信頼度を持つグループ G_x の信頼度と定義される．

$$C_J(j) = C_G(G_x) = \max_{1\leq a\leq g} C_G(G_a) \tag{6}$$

ジョブの信頼度が閾値 $\theta = 1 - \epsilon_{acc}$ を超えた場合，マスタは G_x のリザルトをジョブ j のリザルトとして受け入れ，ジョブ j を終了する．

2.3 チェック率 q

妨害者対策を行う場合，1 つのジョブに対して複数のリザルトを集めたり，spot-checking のためにチェック用のジョブを実行するので，一般に，実行時間 T が妨害者対策を行わない場合と比較して大きくなる．そのため，計算結果の信頼性を保証しつつ，適切なジョブスケジューリングによって実行時間 T を小さくすることが求められる．

信頼度評価に基づく妨害者対策では，アイドル状態にワーカに対して，チェック選択とジョブ選択の 2 つのプロセスによりジョブスケジューリングが行われる．チェック選択は，ワーカに対して spot-checking を行うかどうかを決定するプロセスである．ここでは，一定確率 q で spot-checking を行うものとする．spot-checking を行わない場合（確率 1-q）は，計算プロジェクトの計算を進めるために，ジョブ選択によってどのジョブを割り当てるかを決定する．

チェック率 q の値は任意に決定することができる．しかし，q の値はジョブの実行回数に影響を与えるため，注意が必要である．q の値が大きくなるほど，各ワーカは頻繁に spot-checking を受けることになり，ワーカやリザルトの信頼度は上がりやすくなる．リザルトの信頼度が上がれば，ジョブの終了に必要なリザルト数は減少する．しかし，q が大きすぎると，チェック用ジョブの実行にかかる時間が非常に大きくなってしまう．

実行時間 T を最小化するためには，q の値として，最適値 q_{opt} を用いる必要がある．最適値 q_{opt} は，ワーカ数や許容誤り率といったパラメータに依存するが，従来 q の値は，これらパラメータによらず $q = 0.1$ などの値が用いられていた[5]．その結果，最適値 q_{opt} を用いた場合と比べ，実行時間 T が必要以上に大きくなってしまうという問題点があった．

3. Spot-checking の最適化

3.1 手法の概要

信頼度評価に基づく妨害者対策の実行時間を最小化するためには，計算プロジェクトの開始前に，最適値 q_{opt} を知る必要がある．しかし，実行時間は妨害者の挙動によっても変化するため，計算の開始前に q_{opt} の値を求めることは困難である．本稿では，実行時間の推定値 T_{expect} を q の関数として表現することで，q_{opt} の推定値である q_{est} を，計算の開始前に求める手法を提案する．

ただし，実行時間は q の値だけでなく，ジョブ選択手法によっても変化する．ジョブ選択手法としては，Sarmenta により用いられたラウンドロビン手法[5] があるが，この手法はあまり効率的でない．実行時間の最小化を目的とした，より効率的なジョブ選択手法として，見込み信頼度に基づくジョブ選択手法[6] がある．この手法は，s や f の値によらず，ラウンドロビン手法より高速であることが分かっている[6]．本稿では，見込み信頼度に基づくジョブ選択を行うものとする．

実行時間の推定を行うために，以下を仮定する．W 台のワーカの性能は均一であり，ジョブ 1 つの実行にそれぞれ単位時間がかかる．チェック用のジョブの実行にも同様に単位時間がかかるので，妨害者は，配布されたジョブがチェック用であるかどうかを判別できない．また，ジョブの配布とリザルトの回収にかかる時間は十分小さく，無視できるものとする．

3.2 ジョブ選択手法

見込み信頼度に基づくジョブ選択手法は，図 2 に示すアルゴリズム[6] に基づき，アイドル状態のワーカに配布するジョブを決定する．この手法では，ジョブを実行中のワーカの存在を考慮して，見込みリザルト数と見込み信頼度という 2 つの指標を用いる．

- 見込みリザルト数 $ENR(j)$
  ジョブ j を実行中のワーカ全てがリザルトを返したと仮定した場合の，ジョブ j のリザルト数．
- 見込み信頼度 $EC_J(j)$

```
Push idle workers in worker queue Q;
Group all unfinished jobs into S_i;
  // S_i(i = 0,1,2,...) は ENR が i と等しいジョブの集合
i = 0;
  // S_0 に属するジョブから順に選択する

while ( Q is not empty ) do
 while ( S_i != φ ) do
  Pop worker w from Q;
  Allocate a spotter job to w with rate q;
  if( w is not allocated any job ) then
   Select job j which has minimum EC_J in S_i;
   Allocate job j to worker w;
   Update EC_J(j);
   S_i = S_i - {j};
   S_{i+1} = S_{i+1} + {j};
  end if
  if ( Q is empty ) then
   exit;
  end if
 end while
 i = i + 1;
end while
```

図 2 見込み信頼度に基づくジョブ選択アルゴリズム

ジョブ j を実行中のワーカ全てが，信頼度が最大のリザルトグループ G_x と同じリザルトを返したと仮定した場合の，ジョブ j の信頼度．

3.3 実行時間の推定値 T_{expect}

最適値 q_{opt} の推定値 q_{est} を計算プロジェクトの開始前に求めるため，実行時間の推定値 T_{expect} を q の関数として表す．見込み信頼度に基づくジョブ選択では，見込みリザルト数 ENR の小さい順にジョブが選択されるので，リザルト数の少ないジョブは他のジョブより先に実行される．そのため，いずれかのジョブが 2 個目のリザルトを得る前に，全てのジョブが 1 個目のリザルトを得る．この特徴を利用すると，T_{expect} を式 (7) のように分割することができる．

$$T_{expect} = T_1 + T_2 + \dots \tag{7}$$

ただし $T_i(i = 1, 2, \dots)$ は，未終了のジョブ全てが i 個目のリザルトを得るのに要する時間の長さである．

図 3 に，T_i の例を示す．計算プロジェクトの開始時 (時刻 $t_0 = 0$) において，N 個のジョブ $J_1, \dots, J_N$ は 1 つもリザルトを持っていない．すなわち，時刻 t_0 において未終了のジョブ数は N であり，N 個のジョブ全てが 1 個目のリザルトを得るまでに T_1 単位時間がかかる．ここで，時刻 t_0 から t_1 の間に，各ワーカは最大 T_1 回の spot-checking を受けるので，いくつかのワーカの信頼度は閾値 θ に達する．これらのワーカが生成したリザルトを持つジョブ $J_1, \dots, J_{T1}$ は時刻 t_1 までに終了する．残りのジョブは未終了のままなので，次に時刻 t_1 から t_2 の間に，これら未終了のジョブに対して 2 個目のリザルトが生成される．一般に，時刻 t_{i-1} から t_i の間に，時刻 t_{i-1} にて未終了のジョブに対して i 個目のリザルトが生成される．

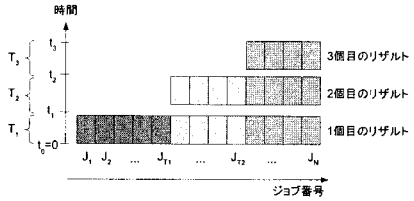


図 3 T_i と i 個目のリザルト

ある妨害者が妨害率 s で誤ったリザルトを返すと，その妨害者は spot-checking によって検出されることがある，妨害者が検出されると，その妨害者によって生成された全てのリザルトは backtracking により無効化される．妨害率 s を任意の値とすると，誤ったリザルトが長い時間無効化されずに残る可能性があり，T_{expect} の計算が非常に複雑になってしまう．本稿では，妨害率 s について，$s = 0$ と $s = 1$ という両極端な場合を考え，T_{expect} を求めるものとする．

本節では，$s = 0$ と $s = 1$ の場合それぞれについて $T_i(i = 1, 2, 3, \dots)$ を求め，最後に $T_{expect}(s = 0)$ と $T_{expect}(s = 1)$ を得る．

3.3.1 T_1

時刻 t_0 から t_1 の間に，W 台のワーカは N 個のジョブそれぞれに対して 1 個目のリザルトを生成する．もし，妨害者が誤ったリザルトを返さないならば $(s = 0)$，各ワーカは計算プロジェクトのジョブを N/W 回と，チェック用のジョブを確率 q でそれぞれ均一に与えられる．各ジョブの実行にはそれぞれ単位時間がかかるので，この場合，$T_1(s = 0)$ は式 (8) で与えられる．

$$T_1(s = 0) = \frac{1}{1 - q} \times \frac{N}{W} \tag{8}$$

また，$s = 1$ の場合，全ての妨害者は spot-checking によって簡単に検出され，誤ったリザルトは backtracking によって無効化される．無効化されたリザルトの代わりのリザルトを生成する分，$s = 0$ の場合より長い時間が必要になる．この場合，妨害者を除く $W \times (1 - f)$ 台のワーカが，N 個のジョブそれぞれに対して 1 個目のリザルトを生成すると考えることができるので，$T_1(s = 1)$ は式 (9) で与えられる．

$$T_1(s = 1) = \frac{1}{1 - q} \times \frac{N}{W \times (1 - f)} \tag{9}$$

3.3.2 $P(i, t)$

$T_i(i = 2, 3, \dots)$ を求めるために，ここで確率 $P(i, t)$ を定義する．$P(i, t)$ は，i 個のリザルトを持つ任意のジョブが，時刻 t までに終了している確率である．まず，時刻 t_0 から t の間に，任意のワーカが k 回の spot-checking を受ける確率 $Pc(k, t)$ は式 (10) で表される．

$$Pc(k,t) = {}_tC_k q^k (1-q)^{t-k} \quad (10)$$

次に，確率変数 $Pu(k_1,...,k_i)$ を次のように定義する．

$$Pu(k_1,...,k_i) = \quad (11)$$
$$\begin{cases} 1 & \text{if } \dfrac{\prod_{k=k_1}^{k_i} Cw(w(k))}{\prod_{k=k_1}^{k_i} Cw(w(k)) + \prod_{k=k_1}^{k_i}(1-Cw(w(k)))} \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

ただし $Cw(w(k))$ は，spot-checking を k 回受けたワーカの信頼度である．$Pu(k_1,...,k_i)$ は，$k_1,\ldots,k_i$ の組み合わせが，ジョブの信頼度を閾値 θ 以上とする場合に 1 となる (ただし $s=0$ 又は $s=1$)．

あるジョブが i 個のリザルトを持っているとき，それらのリザルトがそれぞれ spot-checking を $k_1,\ldots,k_i$ 回受けたワーカによって生成されたものである確率は $\prod_{k=k_1}^{k_i} Pc(k,t)$ である．確率 $P(i,t)$ は，全ての k の組み合わせに対して確率変数 $Pu(k_1,...,k_i)$ を足し合わせることで，式 (12) のように与えられる．

$$P(i,t) = \quad (12)$$
$$\sum_{k_1=0}^{t} \ldots \sum_{k_i=0}^{t} \Big(\prod_{k=k_1}^{k_i} Pc(k,t) \times Pu(k_1,...,k_i) \Big)$$

ここで，$P(0,t)=0$ かつ $P(i,0) \leq P(i,t)$ である．

3.3.3 T_2

時刻 t_1 において未終了のジョブの個数 $Rn(T_1)$ は，確率 $P(i,t)$ を用いて式 (13) で与えられる．

$$Rn(T_1) = N \times (1 - P(1,T_1)) \quad (13)$$

時刻 t_1 から t_2 の間に，ワーカは最大 $Rn(T_1)$ 個のジョブに対して 2 個目のリザルトを生成する．ただし，ワーカが t_2 までの間にさらなる spot-checking を受けることで，$Rn(T_1)$ 個のジョブの一部は，2 個目のリザルトがなくても終了することができる．一般に，T_2 はパラメータ α_1 を使って次のように書ける．

$$T_2(\alpha_1) = T_1 \times (1 - P(1,T_1+T_2(\alpha_1)) + \quad (14)$$
$$\alpha_1 \times (P(1,T_1+T_2(\alpha_1)) - P(1,T_1)))$$

ただし，α_i $(i=1,2,\ldots)$ は主にジョブ選択手法に依存するパラメータである $(0 \leq \alpha_i \leq 1)$．ここで，α_i は未知であるため，本稿では，理想的な場合である $\alpha_i = 0$ を仮定する．この場合，T_2 は式 (15) で与えられる．

$$T_2 = T_1 \times (1 - P(1,T_1+T_2)) \quad (15)$$

3.3.4 T_i $(i=3,4,\ldots)$

T_3 を求めるために，時刻 $t_2(=T_1+T_2)$ において未終了のジョブの個数 $Rn(T_1+T_2)$ を求める．ここで，時刻 t_1 において終了しているジョブ $J_1,\ldots,J_{T1}$ に対しては，仮に 2 個目のリザルトが生成されても，各ジョブは終了したままである点に注意する．よってこれらのジョブに対しては，2 個目のリザルトが生成されたが，そのリザルトはジョブの終了に不必要なものとして無視された，と考えることで，$Rn(T_1+T_2)$ は式 (16) のように求めることができる．

$$Rn(T_1+T_2) = N \times (1 - P(2,T_1+T_2)) \quad (16)$$

式 (15) を求めた時と同様にして，T_3 は式 (17) で与えられる．

$$T_3 = T_1 \times (1 - P(2,T_1+T_2+T_3)) \quad (17)$$

一般に，T_{i+1} $(i=1,2,3,\ldots)$ は式 (18) で与えられる．

$$T_{i+1} = T_1 \times (1 - P(i, \sum_{j=1}^{i+1} T_j)) \quad (18)$$

3.3.5 T_{expect} と q_{est}

式 (7), (12), (18) を用いて，T_{expect} は次で与えられる．

$$T_{expect} = T_1 + T_2 + \ldots + T_m \quad (19)$$

ただし m は $P(m,0)=1$ を満たす最小の自然数であり，次式を満たす．

$$\frac{(1-f)^{m-1}}{(1-f)^{m-1}+f^{m-1}} < \theta \leq \frac{(1-f)^m}{(1-f)^m+f^m} \quad (20)$$

特に，式 (19) より，$T_{expect}(s=0)$ と $T_{expect}(s=1)$ はそれぞれ次式で与えられる．

$$T_{expect}(s=0) = \quad (21)$$
$$T_1(s=0) \times \sum_{i=0}^{m-1} (1 - P(i, \sum_{j=1}^{i+1} T_j(s=0)))$$

$$T_{expect}(s=1) = \quad (22)$$
$$T_1(s=1) \times \sum_{i=0}^{m-1} (1 - P(i, \sum_{j=1}^{i+1} T_j(s=1)))$$

式 (21) - (22) より，$q_{est}(s=0)$ と $q_{est}(s=1)$ はそれぞれ，式 (21) と (22) を最小とする q の値として得ることができる．

4. シミュレーション

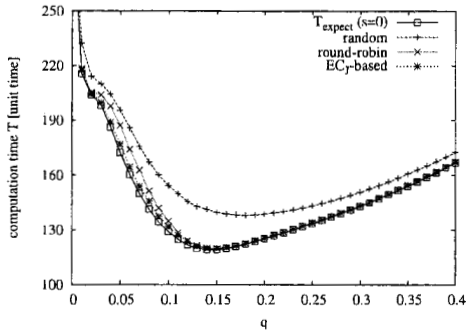
4.1 シミュレーションの仮定

提案した推定式 T_{expect} と q_{est} の精度を評価するため，VC システムの計算シミュレーションを行った．実行時間 T として，3 つの異なるジョブ選択手法 (ランダム，ラウンドロビン[5]，見込み信頼度に基づく手法[6]) それぞれについてシミュレーション 1000 回の平均値を求め，T_{expect} との比較を行った．表 2 にパラメータを示す．

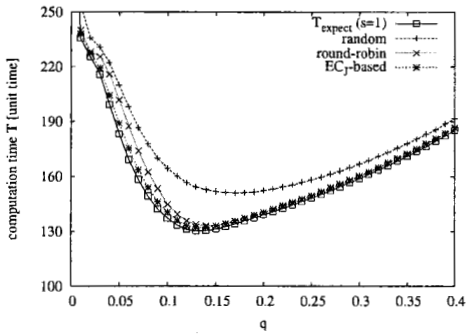
4.2 シミュレーションの結果

4.2.1 実行時間 T と T_{expect} の比較

図 4 に，$\epsilon_{acc} = 10^{-2}$ の場合の q に対する実行時間 T と T_{expect} を示す．この図から，T_{expect} は，見



(a) $s = 0$



(b) $s = 1$

図 4 $\epsilon_{acc} = 10^{-2}$ の場合の実行時間 T と T_{expect}

込み信頼度に基づくジョブ選択を行った場合の実行時間 $T(EC_J\text{-based})$ にほぼ近い値となっていることが分かる．また，例えば図 4(a) の場合，シミュレーション結果から得られる真の最適値 $q_{opt} = 0.14$ に対して，T_{expect} から得られる q_{opt} の推定値 q_{est} も 0.14 となっており，q_{est} が q_{opt} のよい推定となっていることが分かる．

4.2.2 $\mathbf{T}(q = q_{opt})$ と $\mathbf{T}(q = q_{est})$

図 5 に，様々な q の値に対する，見込み信頼度に基づくジョブ選択を行った場合の実行時間 T を示す．$T(q = q_{opt})$ は，q_{opt} を用いた最小の実行時間であり，全ての q に対してシミュレーションを行った結果として得られる．$T(q = q_{est})$ は，計算の開始前に得る

表 2 パラメータ

ジョブ数 N	10000
ワーカ数 W	100
妨害者存在率 f	0.1
妨害率 s	0.0, 1.0
許容誤り率 ϵ_{acc}	$10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$
チェック率 q	0.0 – 0.4

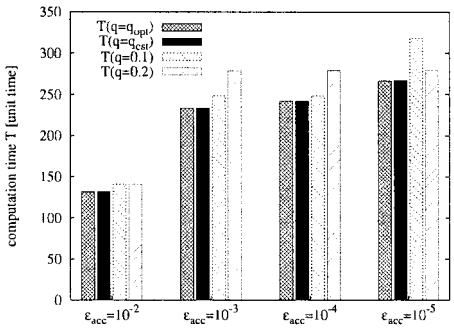


図 5 $s = 1$ の場合の，様々な q の値に対する実行時間 T

ことができる，q_{est} を用いた場合の実行時間である．$T(q = 0.1)$ と $T(q = 0.2)$ は，与えられたパラメータ ϵ_{acc} などによらず，q としてそれぞれの値を用いた場合の実行時間[5] である．この図から，$T(q = q_{est})$ は，ϵ_{acc} によらず常に $T(q = q_{opt})$ に近い値となっていることが分かる．これらのシミュレーションでは，$T(q = q_{opt})$ と $T(q = q_{est})$ の差は最大でも 1 %以下であった．

5. ま と め

本稿では，誤った計算結果を返す妨害者が存在する VC 環境において，実行時間を最小化するために，spot-checking を行う確率 q の最適値を推定する手法を提案した．最適値 q_{opt} の推定値 q_{est} を計算の開始前に求めるため，実行時間の推定値 T_{expect} を q の関数で表現した．シミュレーションにより，T_{expect} から得られる q_{est} を用いることで，真の最適値 q_{opt} を用いた場合とほぼ同程度に実行時間を小さくできることが分かった．

謝辞 本研究の一部は総務省戦略的情報通信研究開発推進制度 SCOPE 特定領域重点型研究開発 (061102002) 助成によって行われた．関係各位に感謝する．

参 考 文 献

1) http://setiathome.ssl.berkeley.edu/
2) http://www.distributed.net/
3) http://boinc.berkeley.edu/
4) http://www.cag.lcs.mit.edu/bayanihan/
5) Luis F. G. Sarmenta, "Sabotage-Tolerance Mechanisms for Volunteer Computing Systems", Future Generation Computer Systems, Vol. 18, Issue 4, pp.561-572, 2002.
6) Kan Watanabe, Masaru Fukushi, Susumu Horiguchi, "Expected-credibility based Job Scheduling for Volunteer Computing Systems", in Proc. of the 2nd International Conference on Advances in Information Technology IAIT2007, pp. 222 – 229, 2007