グループローテーション型クラウドソーシングの制御手法の提案

熊井　克仁† 　白石　優旗†† 　張　建偉†† 　森嶋　厚行†††

† 筑波大学 情報学群情報メディア創成学類　〒 305–8550 茨城県つくば市春日 1-2
†† 筑波技術大学産業技術学部　〒 305–8520 茨城県つくば市天久保 4 丁目 3-15
††† 筑波大学図書館情報メディア系/知的コミュニティ基盤研究センター　〒 305–8550 茨城県つくば市春日 1-2
E-mail: †katsumi.kumai.2015b@mlab.info, ††{yuhkis,zhangjw}@a.tsukuba-tech.ac.jp,
†††mori@slis.tsukuba.ac.jp

あらまし　多くの人々で分担して通訳などを行う際に，複数のグループにわけて，グループ毎に順次通訳を行うような仕組みが考えられる．これを本稿ではグループローテーションと呼び，この考え方を取り入れたクラウドソーシングをグループローテーション型クラウドソーシングと呼ぶ．本クラウドソーシングにおける問題は，参加者の出入りが激しいため，グループの構成を動的に変更しなければならないことである．しかし，好き勝手に構成を変更すれば，参加者が戸惑うことになる．本稿では，このような点を考慮したグループローテーション型クラウドソーシングにおけるグループ再構成の問題を提案し，複数の手法で比較を行う．

キーワード　クラウドソーシング，リアルタイム，割り当て制御

1. はじめに

近年，計算機ネットワーク技術の発達により，ネットワーク上の多くの人々へのアクセスが容易になった．このような背景のもと，注目を集めているのがクラウドソーシングである．クラウドソーシングとは，計算機ネットワークを通じて不特定多数の人々（ワーカ）に業務 (タスク) を委託することである．そのクラウドソーシングの形式の一つに，マイクロタスク型クラウドソーシングがある．マイクロタスクとは一つ一つのタスクが小さく，短時間でこなせるものである．

一方，クラウドソーシングに限らないが，複数人数でマイクロタスク型の作業を行う作業の形態の一つに，本論文で**グループローテーション**と呼ぶものがある．これは，図 1 のように，人々のグループを複数用意し，タスクを順にグループに割り当てていくものである．複数人が各グループに所属するのはタスク結果の品質向上のためであり，例えば多数決などを用いて，グループ毎にタスクの作業結果を求める．

本論文では，クラウドソーシングを用いてこのようなグループローテーション型の作業を行う際に生じる問題について議論する．そのようなクラウドソーシングの例としては，次のようなものが考えられる．

例 1: 通訳.　この場合，各タスクでは，講演などの音声を聞きながら，1 文ずつ通訳を行う作業を行う．そして，次の文になると，次のグループにタスクを割り当てる (図 1)．突然タスクが割り当てられて戸惑わないように，自分の番でない場合には，タスク画面上に「あと 10 タスク後にあなたが通訳を行います」などの案内を出す．各グループのワーカによる通訳結果は，何らかの形で統合され，出力される．

例 2: スポーツ中継のシーン抽出.　この場合，各タスクでは，中継動画を一定時間で区切り，その動画で例えば反則などと思われるタイミングでボタンを押してもらう．一定時間ごとにグループへのタスク割り当てを変更していく．例 1 と同様に，タスク画面上には，自分がタスクを行うまでの残りタスク処理数を表示する．グループ内の多数決などで，グループ毎の結果を計算し，シーンを抽出する．

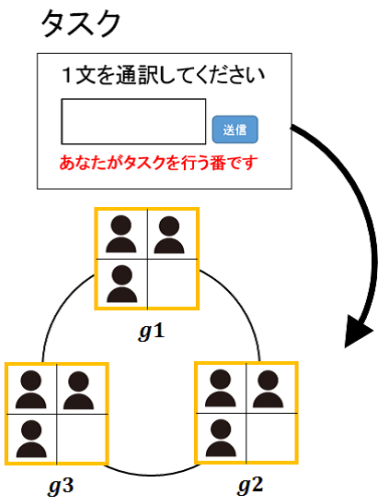

図 1　グループローテーションモデル

このようなグループローテーションにおいては，次の条件を必ず満たす必要がある．

条件 1:　グループ内のワーカの人数が，常に d 人以上である (d はアプリケーションによって決定)．

本論文で扱う問題. グループローテーション作業をクラウドソーシングで行う場合には，ワーカの出入りを制御できないため，人数の増減が避けられない．したがって，これを考慮することが必要となる．まず，ワーカの人数が非常に多くなった場合，固定したグループ数でタスクを行うと，グループ内の人数

は増えるものの，各人の負担は変わらない．もしグループ数を増やすことができれば，ワーカの負担を軽減することができる．一方，ワーカの人数が少なくなり，d 未満になった場合には，アプリケーションによる要求を満たさなくなってしまう．その場合には，グループを統合したり，人が余っているグループから足りないグループに移動する必要がある．

したがって，グループローテーション型クラウドソーシングでは，次の条件を満たす必要がある．

条件 2: ワーカの人数に応じて，グループ数の増減や人の移動を行う．

この問題は，B 木の問題と似ているが，B 木と異なり，移動するのはデータではなく人間である．例えば，あと 10 文後に自分の番だと思っていたワーカのタスク割り当ての順番が，突然あと 1 文後になってしまうなどすると，ワーカは戸惑うことになる．したがって，次の条件を満たす必要がある．

条件 3: ワーカが戸惑うような変化を避ける．

本論文では，これらの条件 1,2,3 を全て満たすグループローテーション型のクラウドソーシング制御という問題に取組むことを提案し，複数のアルゴリズムの提案，比較実験を行う．特に，条件 2 と条件 3 は互いに直感的にトレードオフの関係にあり，これらをある程度両立する制御手法が存在するかどうかは自明ではない．

本研究の貢献は次の通りである．

(1) 制御手法の問題提案．我々の知る限り，本論文は，グループローテーション型クラウドソーシングにおけるワーカの出入りの問題に注目し，その制御を行うという問題を提起した初めての論文である．目標となる具体的な条件を示し，これらを満たす手法を開発することを提案している．

(2) 問題の形式化．本論文では，グループローテーション型クラウドソーシングにおけるワーカの出入りの問題を形式化する．

(3) 複数手法の提案．本論文では，ワーカが出入りする際のグループ割り当ての制御の手法を複数提案する．

(4) シミュレーション評価による比較．本論文では，提案した複数の手法について，様々にパラメータを変更して比較評価を行った結果を示す．それにより，条件 2 と 3 をある程度両立する優れた手法が存在することを示す．

本論文は次のように構成される．2 章では，関連研究について述べる．3 章では，グループローテーションモデルの定義について述べ，問題を形式化する．4 章では，具体的なワーカ参入時，離脱時の処理について述べる．5 章では，提案手法を用いたシミュレーションについて述べる．6 章では，まとめと今後の予定を述べる．

2. 関連研究・関連システム

本論文では，グループローテーション型クラウドソーシングにおける制御手法ついて提案するが，グループローテーションを用いて，文字おこしを行う研究 [1] が存在する．この研究において，Lasecki らは SCRIBE という文字おこしのためのアプリケーションを提案している．SCRIBE では，リテイナーシステム [2] というリアルタイムなクラウドソーシングに特化したアイデアが採用されている．リテイナーシステムとは，あらかじめワーカに報酬の一部を支払うことで，指定した時間にタスクを行うことをワーカと契約するというアイデアである．リテイナーシステムを提案した論文 [2] では，2 秒以内にワーカからのレスポンスが期待できると記されている．また，リテイナーシステムを用いた論文 [3] [4] において，ワーカが契約通りにタスクに協力するという信頼性が証明されている．そのため，SCRIBE ではワーカが増減することを想定していない．本研究では，リテイナーシステムによらないグループローテーション型クラウドソーシングについて考察する．雇用関係のないワーカに対してグループの割り当てを行うことを想定し，ワーカの増減が起こってもワーカの戸惑いが大きくないような制御手法を提案する．

また，本研究の分割や統合，移動の手法は B 木 [5] の分割や統合，移動の考え方に似ている．しかし，本研究では移動するのはデータではなくワーカであるため，できるだけワーカの戸惑いが少ないようにグループ所属を変更する必要がある．

3. グループローテーションモデルの形式化

ここでは，グループローテーションモデルを定義する．そして，グループローテーション型クラウドソーシングにおけるワーカの出入りを問題として形式化する．

3.1 グループローテーションモデルにおける状態

グループローテーションモデルを定義するにあたり，まず，グループローテーションモデルの**状態**について述べる．グループローテーションモデルの状態とは，グループローテーションモデルを用いた作業順序のスナップショットである．図 1 は，グループローテーションモデルの状態の例である．形式的には，次のように定義される．

［定義 1］ グループローテーションモデルにおける状態 S は，5 項組 $S = (W, G, p, Succ, Assign)$ である．ただし，

- $W = \{w_1, w_2, \cdots, w_n\}$ は，現在参加しているワーカの集合である．図 1 では $|W| = 9$ である．
- $G = \{g_1, g_2, \cdots, g_m\}$ は，現在存在するグループの集合である．図 1 では $G = \{g_1, g_2, g_3\}$ である．
- $p \in G$ は，現在タスク割り当て中のグループである．図 1 では $p = g_1$ である．
- $Succ : G \to G$: あるグループ $g_x \in G$ が与えられたとき，g_x の次にタスクを割り当てられるグループ $g'_x \in G$ を返す関数である．図 1 では，$Succ(g_1) = g_2$, $Succ(g_2) = g_3$, $Succ(g_3) = g_1$ である．
- $Assign : W \to G$: あるワーカ $w_x \in W$ が与えられたとき，所属するグループ $g_x \in G$ を返す関数である．

ここで，W_{g_x} を，グループ g_x に所属するワーカの集合 $\{w | Assign(w) = g_x\}$ とする．状態 S においては，全てのワーカが必ず g_x のいずれか一つに属し，グループ内のワーカ

の人数は 1 以上 でなければならない．すなわち，次の条件を必ず満たさなければならない．

（1） $W_{g_1} \oplus W_{g_2} \oplus \cdots \oplus W_{g_m} = W$

（2） $\forall g_x \in G\ (|W_{g_x}| \geqq 1)$

3.2 グループローテーションモデル

グループローテーションモデル $\mathcal{M}$ とは，ワーカの集合の列 $W_1, W_2, \ldots, W_t$ (ただし $|W_{i+1} - W_i| \leqq 1$) を与えると，グループローテーションの状態の列 $S_1, S_2, \ldots, S_t$ を生成するものであり，次のように定義される．

[定義 2] グループローテーションモデル $\mathcal{M}$ は 3 項組 $\mathcal{M} = (d, S_0, Next)$ である．ただし，

- d は，グループ内のワーカの最低人数である．
- $S_0 = (W_0, G_0, p_0, Succ_0, Assign_0)$ は，グループローテーションモデルの初期状態である．ただし，$\forall g \in G_0(|W_g| \geqq d)$ である必要がある．
- $Next$ は，状態 S_i，W_{i+1}，d が与えられたとき，次の状態 $S_{i+1} = Next(d, S_i, W_{i+1})$ を求める関数である．

3.3 *Next* 関数

$Next(d, S_i, W_{i+1})$ は次のように動作する．

- $|W_i - W_{i+1}| = 0$ の場合，S_{i+1} は，S_i と同じであるが，タスク割り当て中のグループを次に進めた状態になる．具体的には，S_i における $Succ_i$ に従い，現在タスク割り当て中のグループ p を次にタスクを割り当てるグループ $Succ_i(p)$ に変更する．したがって，$S_{i+1} = (W_{i+1}, G_i, Succ_i(p), Succ_i, Assign_i)$ となる．
- $|W_{i+1} - W_i| = 1$ の場合，S_{i+1} は，S_i に対してワーカが増加した状態になる．タスク割当てグループは移動しない．$Next$ 関数 は，新ワーカ $W_{in} = W_{i+1} - W_i$ を所属させるグループを決定してそのグループにワーカを追加し，必要に応じてグループの分割を行う (4.1 節)．その結果を新たな状態 S_{i+1} とする．
- $|W_i - W_{i+1}| = 1$ の場合，S_{i+1} は，S_i に対してワーカが減少した状態になる．タスク割当てグループは移動しない．$Next$ 関数 は，離脱ワーカ $W_{out} = W_i - W_{i+1}$ をグループから離脱させ，それにしたがい，ワーカの移動やグループの統合を行う (4.2 節)．その結果を新たな状態 S_{i+1} とする．

上記において，ワーカを所属させるグループの決定にはいくつかの方法が考えられる．本論文では，4.1.2 節で 3 種類の方法を説明する．実験結果が示すとおり，ワーカ挿入時のグループの選択方法の違いが，グループ数の増減と分割と統合の回数に大きな影響を与える．

グループの分割，ワーカの移動，グループの統合に関しては，合理的な方法はそれぞれ一つに決まるため，それらを説明する．

4. *Next* 関数の処理

4.1 ワーカ参入時の処理

本節では，$Next$ 関数におけるワーカ参入時の処理について説明する．

4.1.1 ワーカ参入時の処理とグループ分割

ワーカ参入時には，ワーカを所属させるグループを決定し，

Algorithm 1 ワーカの挿入の疑似コード

Input: d, S_i, W_{i+1}
Output: S_{i+1}

```
1: w_in = W_{i+1} − W_i
2: g_x = pickup(G_i)
3: if |W_{g_x}| < max then
4:     insert w_in to g_x
5: else if |W_{g_x}| == max then
6:     insert w_in to g_x
7:     divide g_x
8: end if
```

そのグループに所属させ，そのグループのサイズが規定値 max を越えたらグループを分割する．

具体的な制御手法を Algorithm 1 に示す．これは次のように動作する．

（1） ワーカを挿入すべきグループ g_x を決定する．(2 行目)

（2） もし，g_x に所属しているワーカ数 $|W_{g_x}|$ が max 未満ならば，g_x にワーカを挿入して終了する．(3〜4 行目) それ以外の場合は，(3) にすすむ．

（3） g_x に所属しているワーカ数 $|W_{g_x}|$ が max の時は (5〜7 行目)，グループの分割を行う．分割の処理を具体的に説明する．まず，新たなグループ g_{new} を確保する．そして，挿入するワーカと g_x に所属するワーカを合わせた $max + 1$ のワーカの半分を g_x に，残り半分を g_{new} に所属させる．

分割のためのパラメータ

グループローテーションでは，グループ内のワーカの最大人数 max の決め方によって，分割の頻度を調節できると考えられる．例えば，$max = 3d$ とすると，$max = 2d$ の場合よりも分割が生じにくくなる．また分割の際，グループは 2 分割されるため，max を $2d$ よりも大きくすると，グループの分割後の 2 つのグループ内のワーカ数が d よりも十分大きくなると考えられる．グループ内のワーカ数が d よりも十分多い場合統合が生じにくくなる．そこで，グループのワーカの最大人数 max を様々な値に変化させることで結果の違いを考察する．max が大きくなればなるほど，分割や統合が起こりづらいが，グループ数は増えづらくなると考えられる．シミュレーションでは，max のパラメータを変え，結果の違いを考察する．

4.1.2 グループの選択方法

先述したように，ワーカ参入時のグループの選択方法の違いが，グループ数の増減と分割と統合の回数に大きな影響を与える．そこでワーカ挿入時のグループ選択として 3 つの手法を提案する．

単純追加法． 単純追加法は，タスクが終わった直後のグループに新しいワーカを所属させる手法である．この手法では，タスクが終わった直後のグループにワーカを所属させることで，ワーカの挿入により分割が起きたとしても，ワーカの戸惑いが小さい．図 2 に手法のイメージ図を示す．

バランス優先法． バランス優先法は，グループ内のワーカ数が最小のグループにワーカを所属させる手法である．もし，グ

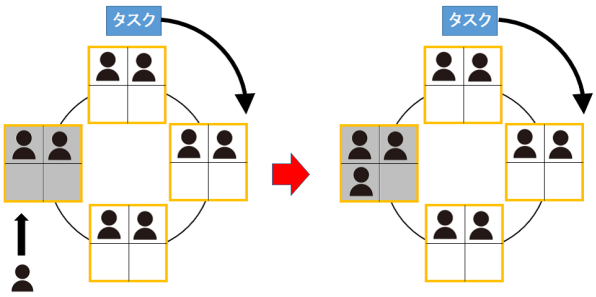

図 2　単純追加法

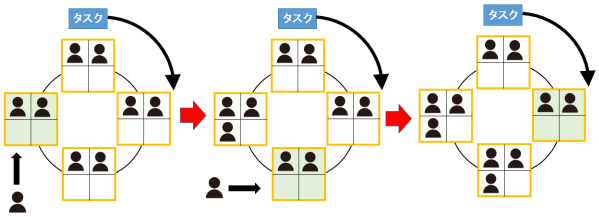

図 3　バランス優先法

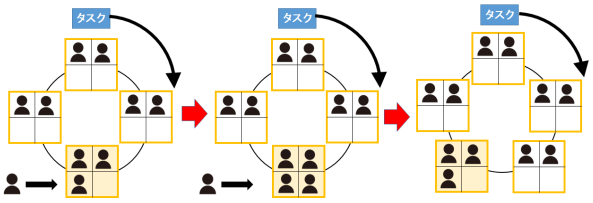

図 4　分割優先法

ループ内のワーカ数が最小のグループが複数ある場合は，その中でタスクを行うまでの残りグループ数が多いグループにワーカを挿入する．この手法では，おおよそ 1 グループのワーカの人数が均等になり，グループの分割，統合が起こりにくくなる．図 3 に手法のイメージ図を示す．

分割優先法． 分割優先法は，グループ内のワーカ数が最大のグループにワーカを所属させる手法である．もし，グループ内のワーカ数が最大のグループが複数ある場合は，その中でタスクを行うまでの残りグループ数が多いグループにワーカを挿入する．タスクこの手法では，グループの分割が早まり，グループ数が増加しやすくなる．図 4 に手法のイメージ図を示す．

4.2　ワーカ離脱時の処理

本節では，$Next$ 関数におけるワーカ離脱時の処理について説明する．ワーカの挿入時と同様に，ワーカが減少した際にも，$Next$ 関数によって，グループローテーションモデルの状態は変化する．ただし，ワーカが離脱する際は，離脱するグループはワーカの意志によって決定されるため，ワーカの挿入時とは異なり，離脱するグループをこちらで制御することが出来ない．このような条件のもと，あるグループ内のワーカ数が d を下回った場合，移動や統合を行い，グループ内のワーカ数が d 以上である条件を満たす必要がある．ここでは，グループローテーションにおいて，条件を満たしつつ，できるだけワーカの戸惑いが少ない移動や統合の手法を提案する．

4.2.1　ワーカの戸惑いが少ない移動や統合の制御

あるグループ g_x のグループ内ワーカ数が条件を満たさなくなった時，グループ間の移動や統合が行われることを考える．ワーカの戸惑いを少なくするためには，移動や統合が起きたとき，あるグループ g_x がタスクを行うまでの残りグループ数（以降 J_{g_x} と表す）の変化が出来るだけ少ないことが望ましい．そこで本手法では，隣接するグループとのみ移動や統合を行う．隣接するグループは，g_x の直後にタスクを行うグループ（g_{back} とする）と g_x の直前にタスクを行うグループ（g_{front} とする）がある．

ワーカの移動

ワーカの移動を行う際，$Jg_{front} < J_{g_x} < Jg_{back}$ より，g_x から g_{back} に移動する場合は，タスクを行う順番が遅くなるため，ワーカの戸惑いが少ない．しかし，g_x から g_{front} に移動する場合は，タスクを行う順番が早まるため，ワーカの戸惑いが大きい．そこで本手法では，グループの移動を行う際は g_x から g_{back} に移動することを優先する．

グループの統合

グループの統合を行う際は，統合が行われたグループよりもタスクを行う順番が遅いグループ全てに対して，タスクを行う順番が早まるという変更が行われてしまう．そのため，g_x と g_{front} が統合するよりも，g_x と g_{back} が統合したほうが全体としてのタスクを行う順番の変更は少なくなる．そこで本手法では，グループの統合を行う際は g_x と g_{back} を統合する．

例外処理

g_x がタスク割り当て中のグループの場合，これまでに説明した手法を適応してしまうと，g_{front} のワーカが，g_x に移動させられることがある．その場合，タスクを行うの順番を示す案内が「あなたがタスクを行う番です」から「あと 10 回であなたがタスクを行う番です」に変わり，その後タスクが発行されると再び「あなたがタスクを行う番です」と変わり，ワーカを戸惑わせる．また，$J_{g_x} = 1$ の場合，同様に手法を適用すると，g_{front} に所属するタスクを処理している最中のワーカが g_x に移動することがある．その場合，タスクを行うの順番を示す案内が「あなたがタスクを行う番です」から「あと 1 回であなたがタスクを行う番です」に変わり，その後タスクが発行されると再び「あなたがタスクを行う番です」と変わり，ワーカを戸惑わせる．そこで，例外的な処理を行う必要がある．本手法では，$J_{g_x} = 0$ の時または，$J_{g_x} = 1$ の時は，g_{front} との移動や統合は行わない．

4.2.2　ワーカ離脱時の具体的な制御

グループローテーションにおいて，グループ内のワーカ数が d 以上であるという条件を満たしつつ，できるだけワーカの戸惑いが少ない移動や統合の具体的な制御手法を Algorithm 2 に示す．また，図 5，図 6，図 7 にワーカ離脱時の具体的な制御の主な動作のイメージを示す．Algorithm 2 は次のように動作する．

（1） 離脱すべきワーカが所属するグループ g_x を取得する．(2 行目)

（2） もし，g_x に所属しているワーカ数 $|W_{g_x}|$ が $d+1$ 以上ならば，g_x からワーカを離脱させて終了する．それ以外の場合は，(3) または (5) にすすむ．(4 行目)

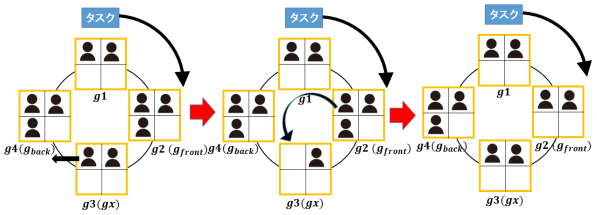

図 5　g_{front} から g_x へ移動時

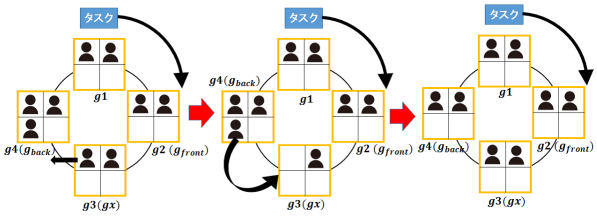

図 6　g_{back} から g_x へ移動時

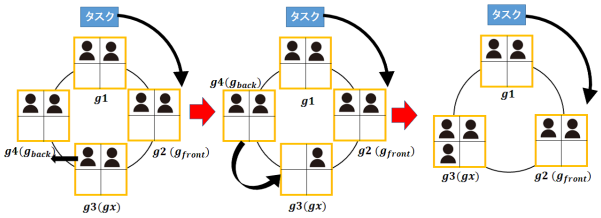

図 7　g_{back} と g_x の統合時

［J_{g_x}=0 または $J_{g_x}=1$ の時］(7〜12 行目)

（3） g_x に所属しているワーカ数 $|W_{g_x}|$ が d の時は，g_x からワーカを離脱させ，g_{back} を調べる．g_{back} に所属しているワーカ数 $|W_{g_{back}}|$ が $d+1$ 以上ならば，条件を満たすように g_{back} から g_x にワーカを移動させる．(8〜9 行目)

（4） g_{back} に所属しているワーカ数 $|W_{g_{back}}|$ が d ならば，g_x と g_{back} を統合する．g_{back} に所属するワーカの所属を g_x に変更する．(10〜11 行目)

［それ以外のグループの時］(14〜22 行目)

（5） g_x に所属しているワーカ数 $|W_{g_x}|$ が d の時は，g_x からワーカを離脱させ，g_{front} を調べる．g_{front} に所属しているワーカ数 $|W_{g_{front}}|$ が $d+1$ 以上ならば，条件を満たすように g_{front} から g_x にワーカを移動させる．(14〜15 行目)

（6） g_{front} に所属しているワーカ数 $|W_{g_{front}}|$ が d ならば，g_{back} を調べる．g_{back} に所属しているワーカ数 $|W_{g_{back}}|$ が $d+1$ 以上ならば，条件を満たすように g_{back} から g_x にワーカを移動させる．(16〜18 行目)

（7） g_{front} に所属しているワーカ数 $|W_{g_{front}}|$ が d かつ，g_{back} に所属しているワーカ数 $|W_{g_{back}}|$ が d ならば，g_x と g_{back} を統合する．g_{back} に所属するワーカの所属を g_x に変更する．(19〜20 行目)

4.3　タスク直前のグループの分割や統合，移動を禁止

本節では，ワーカの戸惑いを削減するためにワーカ参入時，離脱時に共通して行う制御方法について述べる．

グループローテーションでは，あるグループ g_x のタスクが割り当てらるまでの回数 J_{g_x} が小さいほど，グループの所属

Algorithm 2 ワーカの離脱の疑似コード

Input: d, S_i, W_{i+1}
Output: S_{i+1}

```
1: w_out = W_i − W_{i+1}
2: g_x = pickup(G_i)
3: if |W_{g_x}| ≧ d + 1 then
4:     delete w_out from g_x
5: else if |W_{g_x}| == d then
6:     delete w_out from g_x
7:     if J_{g_x} == 0||J_{g_x} == 1 then
8:         if |W_{g_back}| == d + 1 then
9:             move workers from g_back to g_x
10:        else if |W_{g_back}| == d then
11:            merge g_x and g_back
12:        end if
13:    else
14:        if |W_{g_front}| == d + 1 then
15:            move workers from g_front to g_x
16:        else if |W_{g_front}| == d then
17:            if |W_{g_back}| == d + 1 then
18:                move workers from g_back to g_x
19:            else if |W_{g_back}| == d then
20:                merge g_x and g_back
21:            end if
22:        end if
23:    end if
24: end if
```

の変更によるワーカの戸惑いが大きくなると考えられる．そこで，$J_{g_x} < L$(L は自然数) のグループに対しては，分割や統合，移動は行わないという制御を行う．具体的には，$L=1$ とした場合，$J_{g_k}=0$ のグループ g_k に対しては，グループ内の人数 $|W_{g_k}|$ が max を上回っても分割を行わない．また，$|W_{g_k}|$ が d を下回っても統合，移動は行わない．そして，タスクの割り当てが変更され，$J_{g_k}=|G|-1$ となった時，つまり，g_k がタスクから最も遠くなった時に，分割，統合，移動の処理を行う．シミュレーションでは，L のパラメータを変え，比較を行う．

5.　シミュレーション

本章では，4.1.2 節で述べた提案手法を用いて行ったシミュレーションについて述べる．

5.1　概　　要

4.1.2 節で述べたワーカ参入時におけるグループの選択方法である単純追加法，バランス優先法，分割優先法についてシミュレーションを行い，グループ数とタスクを行う順番の変更によるワーカの戸惑いの 2 点からそれぞれの手法の考察を行う．

5.1.1　設　　定

シミュレーションの際には，次のように設定を行った．

- $\mathcal{M} = (2, S_0, Next)$，ただし，初期状態 $S_0 = (W_0, G_0, p_0, Succ_0, Assign_0)$ では，$|W_0|=60$，$|G_0|=20$，および $\forall g_x \in G_0$ $(|W_{g_x}|=3)$
- $Next$ 関数では，3 種類のグループ選択手法をそれぞれ

行い，比較する．

- ワーカ集合の列 $W_1, W_2, \ldots$ については，次の手法で計算する．まず，W_0 からスタートして，$\lambda = 1.5$ のポアソン分布に従いワーカの増減を決定し，増減が行われるたびに新しい W_{i+1} を追加する．それと平行して，一定時間毎にタスクが移動すると仮定し，10 秒毎に $W_{i+1} = W_i$ となるような W_{i+1} を追加する．この列を，タスクが 100 個移動するまで生成する．

5.1.2 ワーカの戸惑い（$PENALTY$）

ここで，分割と統合によってタスクを行う順番が変更されることで生じるワーカの戸惑いを，計算式によってスコアとして数値化する．ワーカの戸惑いはワーカの所属するグループ g_x のタスクまでの残りグループ数 J_{g_x} を用いて次のように表す．

$$PENALTY = \frac{1}{J_{g_x} + 1} \quad (0 \leqq J_{g_x} \leqq |G| - 1) \qquad (1)$$

ワーカの戸惑いは J_{g_x} が小さければ小さいほど大きくなる．(1) 式はこの考え方に従い，J_{g_x} が小さければ小さいほど値が大きくなるように，J_{g_x} の逆数をとり，ワーカの戸惑いを数値化している．

また，分割や統合によりワーカのグループ所属が変更される際，タスクまでの残りグループ数が増える場合にはワーカの戸惑いが少ないが，減る場合にはワーカの戸惑いは大きくなる．例えば，グループ所属の変更により，あるワーカがタスクまでの残りグループ数を 3 から 4 に変更される場合，タスクまでの時間的余裕が生まれ，ワーカにとって大きな戸惑いにはならない．しかし，タスクまでの残りグループ数が 3 から 2 に変更される場合，急にタスクを行うまでの時間が短くなり，ワーカにより大きな戸惑いを与える．このことを踏まえて，タスクまでの残りグループ数が減る場合と増える場合について重みをつける必要がある．本シミュレーションでは，減る場合の $PENALTY$ を 2 倍にして計算を行う．したがって，(1) 式は次のようになる．

$$PENALTY = \begin{cases} \frac{1}{J_{g_x}+1} & (\text{増えるとき}) \\ 2 \times \frac{1}{J_{g_x}+1} & (\text{減るとき}) \end{cases} \quad (0 \leqq J_{g_x} \leqq |G|-1) \qquad (2)$$

5.2 シミュレーション結果の考察

5.2.1 ワーカ参入時におけるグループの選択方法

グループ内のワーカの最大人数を $max = 2d$ とし，それぞれの手法ごとに 100 回のシミュレーションを行った．シミュレーションによって得られたグループ数 (図 8) と $PENALTY$ の合計値 (図 9) の推移を手法ごとに示す．

図 8 に，各手法ごとのある 1 回のシミュレーションにおけるグループ数の推移を示す．ここから分割優先法が最もグループ数が多くなり，ワーカの 1 人あたりのタスク処理回数が少なくなると分かる．しかし，分割優先法では頻繁にグループ数が変化している．この結果から，ワーカの 1 人当たりタスク処理回数は少ないが，グループの分割や統合が頻繁に生じているため，ワーカの順番の変更が多くなり，ワーカの戸惑うことが多いと考えられる．一方で，バランス優先法はグループ数があまり増えず，ある程度一定のグループ数を保っている．このことから，1 人のワーカのタスクの処理回数は多いが，グループの分割や統合はあまり起こらないため，ワーカの順番の変更による戸惑いは少ないと考えられる．

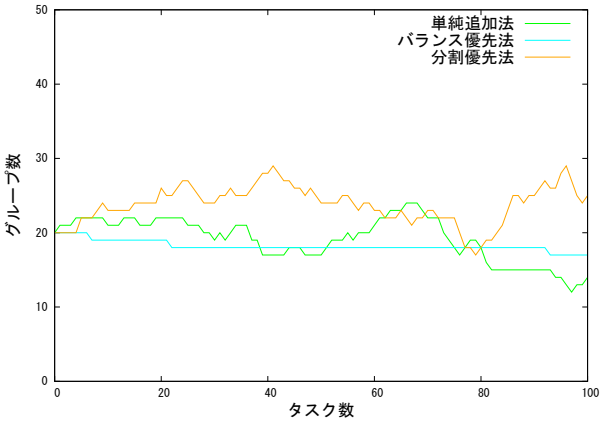

図 8　グループ数の推移

図 9 に，提案手法ごとの $PENALTY$ の合計値の推移を示す．グラフの傾きがが大きければ大きいほど，分割や統合，移動によりタスクを行う順番の変更が起きた回数が多いと言える．または，ワーカの戸惑いが大きい順番の変更が起こったと言える．図から，分割優先法は傾きが大きく，バランス優先法は傾きが小さいことが確認できる．このことから，分割優先法は比較的，分割や統合の回数が多くグループ数が安定しないが，バランス優先法は分割や統合の回数が少なく，グループ数が安定していると考えられる．

図 10 に，100 回のシミュレーションを行った結果を「グループ数の平均」と「$PENALTY$ の合計値」の 2 項目でプロットした散布図を示す．分割優先法ではグループ数を増加させることは実現できたが，$PENALTY$ の合計値が大きくなってしまった．一方，バランス優先法ではグループ数を増加させることはできなかったが，$PENALTY$ の合計値をかなり抑えることができた．

バランス優先法と分割優先法を比較すると，バランス優先法のグループ数の平均は分割優先法より約 30 %の減少となったが，$PENALTY$ の合計値の平均を約 90 %削減した．また，バランス優先法と単純追加法を比較すると，バランス優先法はグループ数の平均を約 19 %の低下にとどめ，$PENALTY$ の合計値の平均を約 84 %削減することができた．これらの結果から，バランス優先法は他の手法よりも相対的に優れた手法であると言える．また，分散を比較するとバランス優先法は分散が小さく他の手法よりも信頼度が高いと考えられる．

5.2.2 分割のためのパラメータを比較

図 11，図 12，図 13 にワーカ参入時におけるグループの選択の各手法において，グループ内のワーカ数の最大値 max を変更し，それぞれ 100 回のシミュレーションを行った結果を「グループ数の平均」と「$PENALTY$ の合計値」の 2 項目でプロットした散布図を示す．図 11 と図 13 より，単純追加法，

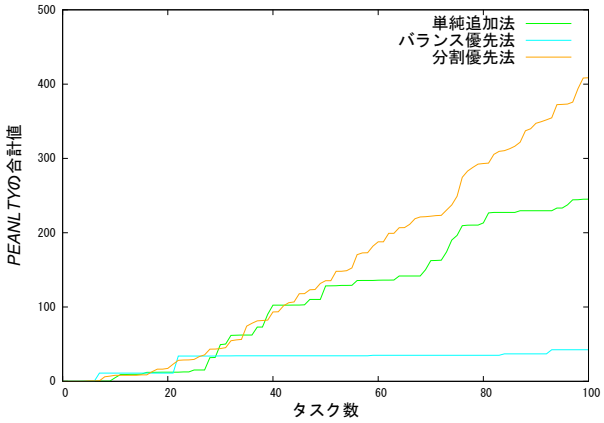

図 9　$PENALTY$ の合計値の推移

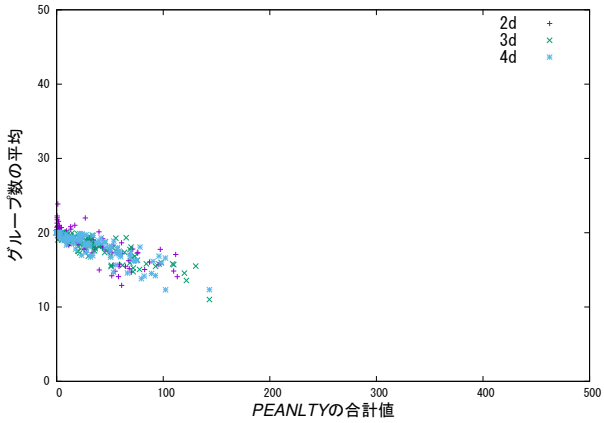

図 12　グループ内のワーカ数の最大値を変更（バランス）

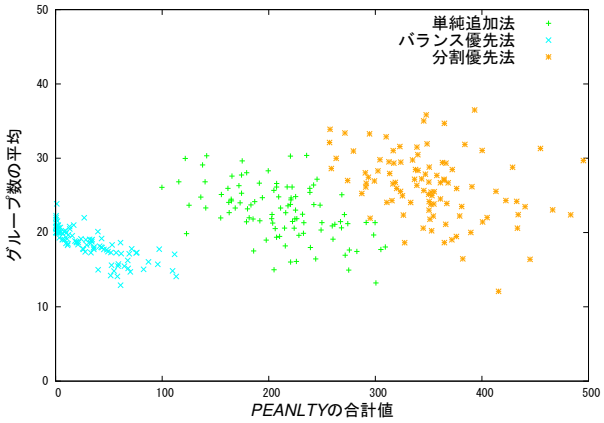

図 10　グループ数の平均と PENALTY の合計値の散布図

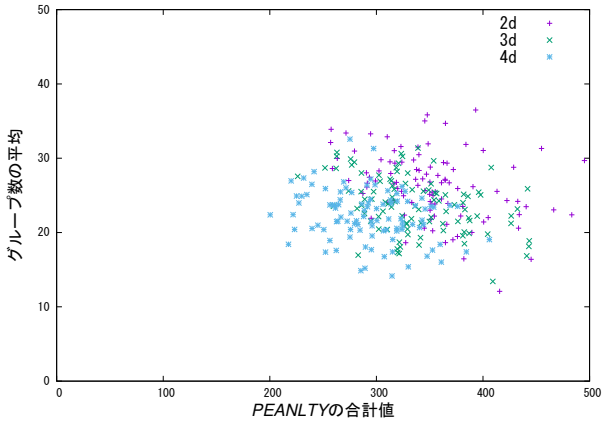

図 13　グループ内のワーカ数の最大値を変更（分割）

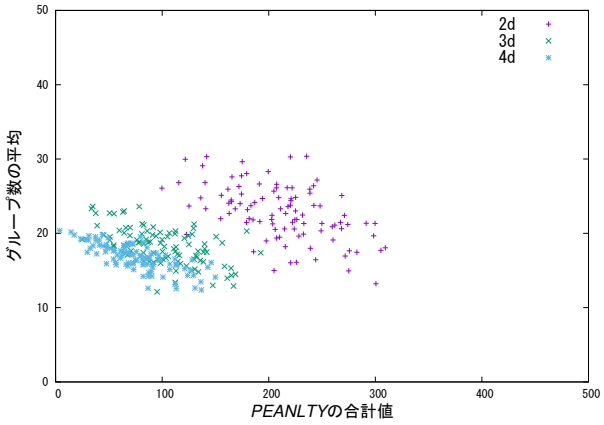

図 11　グループ内のワーカ数の最大値を変更（単純）

分割優先法では，max の値を大きくすると，「グループ数の平均」と「$PENALTY$」の値がどちらも低くなっている．これは，max の値を大きくすることで分割が起こりづらくなったためだと考えられる．一方で，図 12 より，バランス優先法では「$PENALTY$ の合計値」，「グループ数の平均」ともにほとんど変化がなかった．これは，バランス優先法がもともと分割が起こりづらい手法であるため，max の大小があまり影響しなかったのではないかと考えられる．

5.2.3　分割や統合，移動を禁止するグループ数を比較

図 14，図 15，図 16 にワーカ参入時におけるグループの選択の各手法において，分割や統合，移動を禁止するグループ数を変更し，100 回のシミュレーションを行った結果を「グループ数の平均」と「最終 $PENALTY$」の 2 項目でプロットした散布図を示す．L はタスク作業直前の分割や統合，移動を禁止するグループの個数 L を示す．

今回のシミュレーションでは，分割や統合，移動の禁止がワーカへの戸惑いを軽減することに有効であるかは確認できなかった．おそらく，グループ数が多数あるような場合，タスク直前のたかだか L 個のグループに対して分割や統合，移動を行わないという制御を行ったとしても，必ずしも制御を行う L 個のグループにワーカの挿入や離脱の操作が生じるわけではないため，有効な手段にならなかったのではないかと考えられる．

6.　まとめと今後の課題

本論文では，グループローテーション型クラウドソーシングの制御手法の提案をおこなった．グループの増減やワーカの移動と，それに伴うワーカの戸惑いはトレードオフの関係にあるが，単純追加法，バランス優先法，分割優先法の 3 手法の比較により，バランス優先法がそれらをある程度両立することを示した．

今後の課題としては，(1) 全体のペナルティの削減だけでなく，個々人のペナルティに配慮した手法の検討，および (2) 現実により即したペナルティの計算方法を明らかにするための実

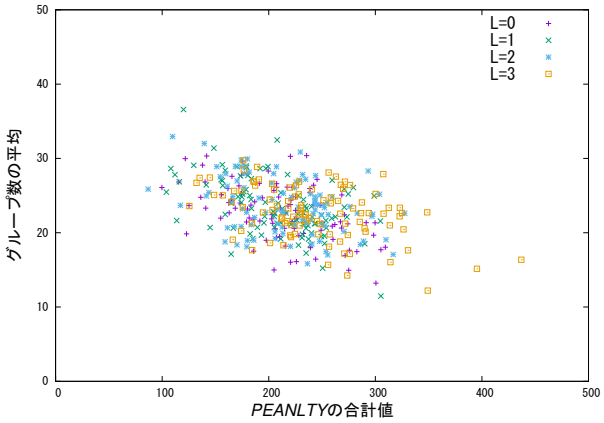

図 14　分割や統合，移動を禁止 (単純)

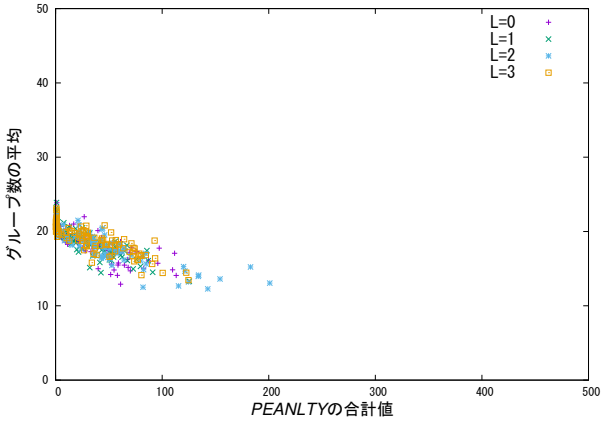

図 15　分割や統合，移動を禁止 (バランス)

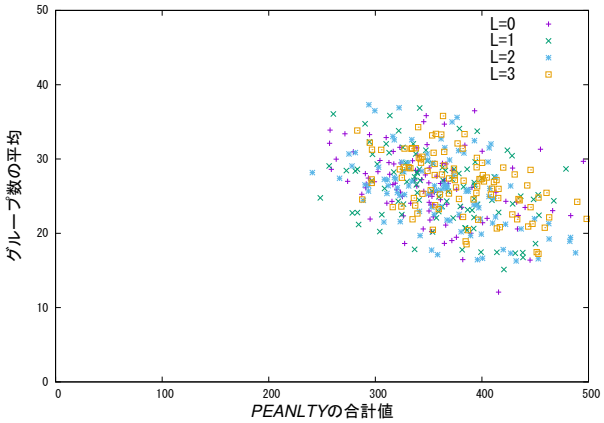

図 16　分割や統合，移動を禁止 (分割)

験，などがあげられる．

謝　　辞

本論文の一部は JSPS 科研費 (25240012, 26870090)，並びに筑波技術大学平成 27 年度学長のリーダーシップによる教育研究等高度化推進事業による助成の成果であり，ここに記して謝意を表すものとする．

文　　献

[1] W.S. Lasecki, C.D. Miller, A. Sadilek, A. Abumoussa, D. Borrello, R. Kushalnagar, J.P. Bigham. Real-time Captioning by Groups of Non-Experts. In Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2012). Boston, MA. p23-34.

[2] M. S. Bernstein, J. R. Brandt, R. C. Miller, and D. R. Karger. Crowds in two seconds: Enabling realtime crowd-powered interfaces. In Proc. of the 24th annual ACM Symp. on User Interface Software and Technology, UIST ' 11, p33-42. 2011.

[3] L. Chilton. Seaweed: A web application for designing economic games. Master' s thesis, MIT, 2009.

[4] L. von Ahn and L. Dabbish. Labeling images with a computer game. In Proc. of the Conf. on Human Factors in Computing Systems, CHI ' 04, p319-326. 2004.

[5] Bayer, R. and McCreight, E. M. : Organization and Maintenace of Large Ordered Indexes. Acta Informatica, 1(3), pp.173-189. 1972.