クラウドソーシングによるデータ収集のためのタスク生成手法の提案

青木　秀人† 　森嶋　厚行††,††† 　三津石智巳†

† 筑波大学大学院 図書館情報メディア研究科　〒 305–8550 茨城県つくば市春日 1–2
†† 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
††† JST さきがけ
E-mail: †{aoki,mori,tomomi}@slis.tsukuba.ac.jp

あらまし　近年，Web 上での多くのサービスにおいてクラウドソーシングによるデータ収集・整理・処理等が広く行われている．一般に，クラウドソーシングの依頼者は，ワーカが処理するタスクを用意する必要があるが，このタスクの設計はクラウドソーシングの結果に大きく影響を与える．本論文では，データ収集を行うクラウドソーシングに焦点を当て，データ収集の範囲を限定したタスクを数多く用意して結果の再現率と作業の並列性を向上する手法を提案する．ここでの問題は，必要なタスクの数が多くなる上に，具体的なタスクは問題依存であるため，あらかじめプログラマが用意する事は煩わしい事である．したがって，提案手法では，必要なタスクの生成に，ワーカ自身を参加させるというアプローチをとる．具体的には，まずシステムにタスク生成プランを与える．次に，システムは，ワーカが行なったタスクの結果とタスク生成プランを照らし合わせ，順次新たなタスクを生成する．実験により，提案手法ではデータ収集の再現率が向上したことが確認できた．

キーワード　クラウドソーシング

1. はじめに

近年，Web 上のクラウドソーシングによる情報の収集・整理・処理等が広く行われている．特に，人の力を使ってデータを収集する処理は多くのアプリケーションで広く行われている．例えば，Wikipedia などの辞典作成や食べログ [1] では，人の力を使って事象やお店の名前などを収集している．クラウドソーシングにおける重要な演算の一つに Human-powered Search がある．これは，Web 上に存在するがアルゴリズムでは探索が困難なデータを人手で発見したり，計算機上に存在しない情報を人に求めたりするものである．典型的な例としては Yahoo 知恵袋などの Q&A サービスが挙げられる．

Human-powered Search の問題は，結果が一つであるような場合には単純な方法でうまくいく場合もあるが，結果が複数個あるような場合に必ずしも単純な方法は向いていないことである．例えば，つくば市にある病院や公共施設をできるだけ多く求めたい場合，単純な方法のタスク（図 1）では必ずしもうまくいかない．このとき，ある対象を**できるだけ多く**求めたい，すなわち，**再現率が重要**な場合を本論文では**データ収集**と呼ぶ．

一般に，クラウドソーシングの依頼者は，ワーカが処理するタスクを設計する必要があるが，このタスク設計は，クラウドソーシングの結果に大きく影響を与える．例えば，クラウドソーシングによるデータ収集における単純な方法は，図 1 のようなタスクを用いてデータを集める事である．このタスクでは，ワーカは現在集まっている情報を見ながら，新たな情報を追加する．このタスク設計では，結果が増えてくるにつれてどの情報が既に入力されているか把握することが困難になるため，タスクが完了できない可能性や再現率が低い状態でタスクが完了する可能性がある．

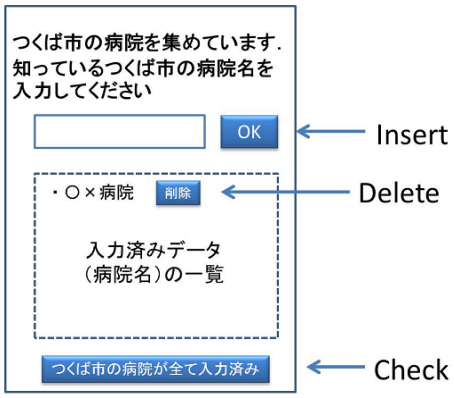

図 1　データ収集を行うための単純な入力フォーム

クラウドソーシングによるデータ収集を行いやすくする方法としては，問題を分割統治する事が考えられる．すなわち，扱う問題をより小さくしたタスクを複数用意することが考えられる．例えば，つくば市にある病院を多く求めたい場合には，つくば市を「竹園」「天久保」といった地域に細分化し，その地域ごとに病院を求めるタスクを作成するといった事である．しかし，このようなタスクの数は一般に多くなる上に具体的なタスクは個々の問題に依存するため，あらかじめ全てのタスクをプログラマが用意する事は煩わしい．

本論文では，まずシステムに**タスク生成プラン**を与え，次にシステムがそのプランにしたがって，ワーカをタスク生成に参加させる手法を提案する．提案手法により，次の事が期待される．(1) タスクあたりの処理時間が短くなる．(2) タスクの並列性が高くなる．(3) データ収集の精度・再現率が高くなる．(4) 具体的なタスク生成はワーカに任せるため，プログラマがあらかじめ全てのタスクを用意する必要が無い．

関連研究. クラウドソーシングにおけるデータ処理においては，しばしば選択 (filtering)，結合，ソートなどのデータベース演算の処理が行われることが知られており，これらの演算についての効率化の議論が行われてきた [2] [3] [4] [5]． これらのなかで，選択，ソート，結合等の効率化については提案されているが，我々の知る限り，データ収集の処理を明示的に論じた研究は存在しない．

本論文の構成は次の通りである．2 節ではクラウドソーシングにおけるタスクとタスクの評価尺度について説明する．3 節ではタスク処理のモデルと単純な手法について説明する．4 節では提案手法について説明する．5 節では実験について説明する．6 節はまとめと今後の課題である．

2. タスク処理モデルと単純な手法

本節では，まず，タスクの処理モデルについて説明する．次にタスク表現方法について説明する，最後に，単純なデータ収集とその問題について説明する．

2.1 タスク処理モデル

本論文では，クラウドソーシングにおいてワーカに割り当てられる作業の一つ一つをタスクインスタンス（もしくは単に**タスク**）t と呼ぶ．例えば，図 1 のような画面で処理を行う作業は一つのタスクである．

タスクを行う人々をワーカ集合 $W = \{w_1, \ldots, w_n\}$ で表す．ここで，w_i は個々のワーカを表す．

このとき，本論文では，タスク処理を次のようにモデル化する (図 2)

（1） ワーカに依頼したいタスクがタスクプール T に格納されている．T の初期値は，手続き `initialize` によって生成されたタスクが，T に格納されたものである．

（2） ワーカ w がタスクを行う際には，関数 `GetTask()` によってタスク $t \in T$ を入手する．w が t を処理したという事実は，テーブル $TaskExecution(t, w) \in T \times W$ に記録される．

（3） ワーカ w がタスク t を行った結果 $result$ は，手続き `Performed(t, result)` によってデータベースに格納される．

2.2 タスクの表現

本論文では，データ収集のためのタスクを，$EntryTask(scope, item)$ と表現する．ここで，$scope$ はデータ収集の範囲，$item$ は収集対象を表す．このタスクの画面には，現在まで収集されたトピックが表示されており，ワーカは，データ入力，データ削除，入力確認の作業を行うことができる．

例えば，筑波大学の研究トピック (Topic) のデータ収集と，つくば市の病院 (Hospital) のデータ収集のためのタスクは，それぞれ次の様に表現する．

$EntryTask(\text{"筑波大学"}, Topic)$

$EntryTask(\text{"つくば市"}, Hospital)$

図 1 は，$EntryTask(\text{"つくば市"}, Hospital)$ の画面である．このタスクでは，ワーカは次のいずれかを行う．

- これまで入力されていない病院を入力し Insert ボタンを押す．
- これまでに間違って入力されている病院を選択し Delete ボタンを押す．
- これまでに全てデータが収集されていることを確認し Check ボタンを押す

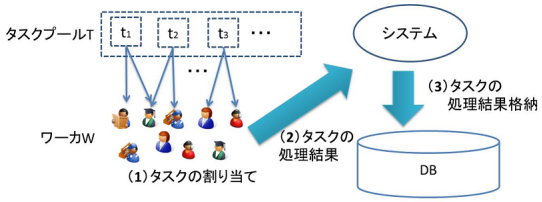

図 2 タスク処理のモデル

```
Initialize(scope, item){
    T.insert(EntryTask(scope, item));
}
```

図 3 単純なデータ収集：初期化

```
GetTask(w){
    return T.getOne();
}
```

図 4 単純なデータ収集：タスクの割当て

```
Performed(t, result){
    Let t be TaskEntry(scope, item)
    DB.insert(item, result);
}
```

図 5 単純なデータ収集：タスク実行後の処理

2.3 単純なデータ収集

筑波大学の研究トピックを集めるための単純なデータ収集では，唯一つのタスク $EntryTask(\text{"筑波大学"}, Topic)$ を利用する．先のモデルにおいては，次の手続きを利用すればよい．

（1） タスクプールの初期化を行う手続き `Initialize("筑波大学", "Topic")` (図 3)．ここでは，データ収集のタスクがただ一つ生成され，タスクプールに追加される．

（2） タスクの割り当てを行う `GetTask(w)` (図 4)．ただし，タスクプールには一つのタスクしかないので，全てのワーカに同じタスクが割り当てられる．

（3） タスクの結果を格納する `Performed(t, result)` (図 5)．手続き定義における `DB.insert(item, result)` は，`item` と同名のリレーション (例えば Topic) に，`result` を格納する事である．

2.4 単純なデータ収集の問題

再現率の向上という観点から見た場合， 収集対象のデータ数が大きくなると，単純なデータ収集のクラウドソーシングでは，ワーカが次のような困難に直面すると考えられる．(1) 何が未入力かわかりにくいので，入力時 (Insert 時) に，再現率をあげるような適切なデータの入力がされない可能性が増加する．(2) 間違っているかどうかの判定も難しいため，本来 Delete ボタ

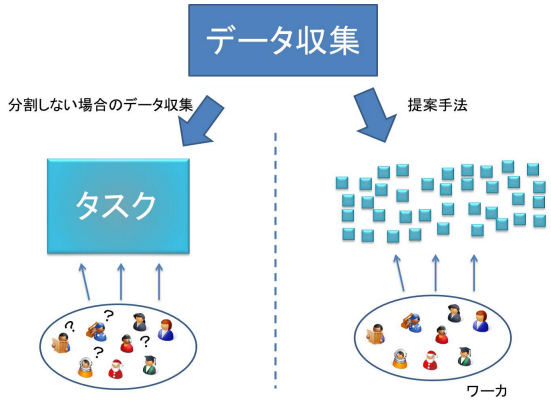

図 6　単純なデータ収集と提案手法の比較イメージ

ン押さなければならない状況で押されない可能性が増加する．(3) 全部入力されたかどうか (Check ボタンを押すか否か) の判定が難しいため，間違ってデータ収集完了の判断をする可能性が増加する．以上の様な理由から，ワーカがタスク処理に悩み，タスクが完了しない可能性やデータが全て入力されない (再現率が低い) 状態でタスクが完了する可能性がある．

3. 提 案 手 法

3.1 概　　要

提案手法のアイデアは，(1) データ収集を分割統治すること，および (2) ワーカ自身が処理の分割を行う事，である．下記でそれぞれについて説明する．

3.1.1 データ収集の分割統治

単純なデータ収集では，ただ一つのタスクを全てのワーカで処理したが，それとは異なり，提案手法ではデータ収集の範囲を小さく限定したタスクを多数作り，それを分担して処理する．(図 6)．このように小さなタスク生成することで，タスクあたりの処理時間が短くなり，かつデータ収集の精度・再現率が高くなることが期待できる．

例えば，筑波大学の研究トピックのデータ収集を行う場合，筑波大学の研究トピックは非常に多いため，単純なデータ収集では困難な事が予想される．しかし，筑波大学の組織構造に着目すれば，例えば次のように研究科単位に収集範囲を限定したタスクをタスクプール T に登録するという方法が考えられる．

$$
\begin{aligned}
&T = \{ \\
&EntryTask(\text{"物理研究科"}, Topic), && (\mathrm{t}_1) \\
&EntryTask(\text{"文学研究科"}, Topic), && (\mathrm{t}_2) \\
&\vdots \\
&\}
\end{aligned}
$$

t_1 は物理研究科の研究トピックのデータ収集をするタスクであり．t_2 は文学研究科の研究トピックのデータ収集をするタスクである．このように収集範囲を限定した複数のタスクを用意する事で，筑波大学の研究トピックを漠然と収集するのではなく，各ワーカが特定の範囲のデータ収集に集中できるようにする．

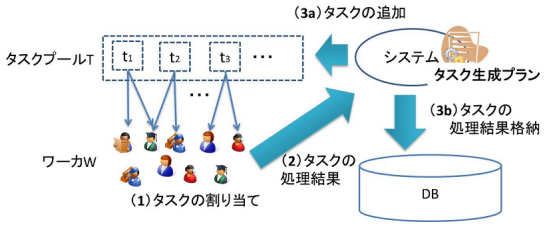

図 7　提案手法のタスク生成の流れ

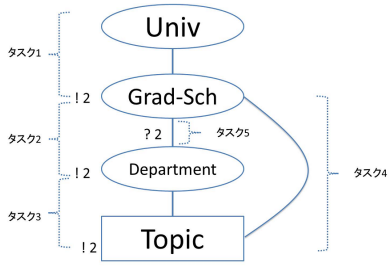

図 8　タスク生成プランのイメージ

3.1.2 タスク生成プランを用いたワーカによるタスク生成

前節で述べたようなタスク収集の分割統治は再現率の向上に有効であると予想されるが，具体的にどのようなタスクを用意するかは収集対象に依存しかつ数も多いため，問題毎にプログラマが全てのタスクをあらかじめ用意しておくことは煩わしい．

そこで，提案手法ではこの作業自体をクラウドソーシングする．ただし，ある程度のタスク生成の枠組みは与えておく．このタスク生成の枠組みのことを**タスク生成プラン**と呼ぶ．

図 7 は，タスク生成プランを用いた処理の流れを示す．基本的には，4 章で説明したタスク処理モデルと同じであるが，次の点が異なる．

（1） あらかじめ，システムにタスク生成プランを与えておく

（2） タスクの処理結果 (result) が与えられると，タスク生成プランに従って，システムが新たなタスクを生成し，T に追加する

以降では，まず，タスク生成プランについて説明し，次に，タスク生成プランを用いたワーカによるタスク生成 (T へのタスク追加) について説明する．

3.2 タスク生成プラン

本論文におけるタスク生成プランとは，データ収集プロセスにおいて，動的に必要なタスクを生成するための枠組みを簡潔に表記したものである．具体的には，収集対象範囲の階層を記述しており，先の例における，大学の研究トピックを研究科毎に収集するといったような事が記述できる．

まず，タスク生成プラン中に記述する情報のイメージを図 8 に示す．これは，大学の研究トピックを収集するために，どのようなタスクを生成すべきかを記述している．末端ノード (Topic) の形だけが異なるのは，これが最終的に収集したいデータであることを示している．

この図において，図中のノードをつなぐ各エッジが，生成されるタスクに対応する．すなわち，このタスク生成プランで

```
[Univ{Tsukuba University}<uname>,
   Grad-Sch[!2, *]<gname>,
   Department[!2, ?2]<dname>,
   *Topic[!2]<topic>]
]
```

図 9　タスク生成プラン

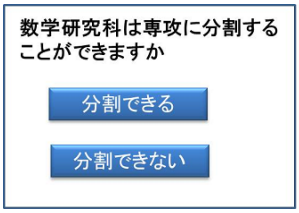

図 10　(2)DivideTask のインタフェース

は次の 4 つのデータ収集タスク (図 8) が生成される．(1) 大学 (Univ) の研究科 (Grad-Sch) を収集するタスク (タスク 1)，(2) 研究科の専攻 (Department) を収集するタスク (タスク 2)，(3) 専攻の研究トピック (Topic) を収集するタスク (タスク 3)，(4) 研究科の研究トピックを収集するタスク (タスク 4)．

次に，各ノードの横に書かれている $!x$ は**完了同意回数**であり，そのデータの収集が完了したことを x 名のワーカで確認する必要があることを表している．例えば，図 8 においては，研究科の研究トピックが既に全て収集されたかの確認は 2 名のワーカで同意を取る必要がある事が書かれている．

また，エッジの横に書かれている $?y$ は**生成同意回数**，タスクを生成する前に，そのタスクを行う必要がある事を y 名のワーカで同意を得なければならないことを表している．例えば，図 8 においては，研究科単位でなくさらに専攻毎に分割してデータを収集するタスクを生成するかどうか (タスク 5) は，2 名のワーカで同意を取る必要がある事が書かれている．

生成同意を導入する目的は，データ収集分割のための階層構造が必ずしも均一とは限らない場合があるからである．例えば，物理研究科には専攻があるが，社会学研究科には専攻がない場合，全ての場合に専攻に分割すると，社会学研究科に関して不適切なタスク生成を行なってしまう．

記法.　実際には，タスク生成プランは図 9 のように $[n_1, n_2, \ldots, n_m]$ と記述する．これは，基本的には図 8 と同じ情報を保持しており，各 n_i は各ノードを表す．$Topic$ だけは，先頭に*が着いているが，これは末端ノードであることを表す．各ノードの直後の [] の中には，完了同意回数，生成同意回数，末端ノードへのエッジが存在するか (*) を記入する．

さらに，実際のタスク生成プランは，図 8 には存在しない次の 2 つの追加情報を持つ．

（1）　最も上位のタスク (Univ-Grad-Sch) の Univ の初期値 (この例では “筑波大学”) が {...} に書かれている．

（2）　各ノードが表すデータを格納するリレーションの属性名が，’< ... >’ に書かれている (例えば，リレーション $Grad\text{-}Sch$ は属性として $gname$ という研究科名を格納する属性を持っている)．

3.2.1　生成するタスク

提案手法では，そのプロセスにおいて 2 種類のタスクを生成する．本節ではそれらを説明する．

(1) データ収集タスク (EntryTask) このタスク (図 1) では次の 3 つのうち 1 つをワーカが行う．

- ($Insert$) 対象となるデータを入力
- ($Check$) 対象のデータが全て揃っているか判断
- ($Delete$) 既に入力されたデータを削除

例えば，筑波大学の研究トピックのデータ収集を行うタスク $EntryTask(\texttt{"筑波大学"}, Topic)$ の場合 , 具体的にワーカが行うことができる行動は次のである．

- ($Insert$) 筑波大学の研究トピックである「クラウドソーシング」などを入力する．
- ($Check$) 入力されている研究トピックを見て，これらのデータで筑波大学の研究トピックが全て揃っているか判断する．
- ($Delete$) 入力されている研究トピックを見て，間違っている研究トピックや重複してい病院を削除する．

提案手法においては，上位のタスク (研究科の収集等) で漏れがあると最終的なデータ収集に大きな影響を与えるため，$Check$ が正しく処理されるかどうかは特に重要である．

(2) 生成同意タスク (DivideTask) 生成同意タスク $DivideTask(scope, n_i)$ は，$scope$ の収集範囲をさらに n_i 単位のタスクに分割する前に，そのタスクが必要かどうかに関する同意を求めるものである．図 10 は，数学研究科が専攻に分割できるか問い合わせるタスク $DivideTask(\texttt{"数学研究科"}, Department)$ の画面である．ワーカは「分割できる」か「分割できない」の 2 つから 1 つを選択する．

3.3　タスク生成アルゴリズム

本節では，タスク生成プラン $[n_1\{v_1\}, n_2, \ldots, n_m]$ が与えられた時のタスク生成アルゴリズムについて説明する．本アルゴリズムは次の手続き (図 11) から構成される．

（1）　初期化 (initialize)：タスク生成プランの最上位ノード n_1(初期値 v_1) と n_2 から初期タスク $EntryTask(v_1, n_2)$ を生成し T に登録する．例えば，図 9 の場合は，$EntryTask(\texttt{"Tsukuba University"}, Grad\text{-}Sch)$ を T に登録する．

（2）　割り当て (GetTask): T 中のタスクの中で最も過去に登録されたタスクを割り当てる．

（3）　タスク処理終了時 (Performed):次の 4 つのケースに分かれる．

- タスクが EntryTask であり Insert ボタンを押された場合: 結果 $result$ を DB に格納する．このときに，同時に，タスク生成プランを見て，一つ下位のタスクを生成する．例えば，処理されたタスクが $n_1 - n_2$ の場合，$n_2 - n_3$ に対応するタスク $EntryTask(result, n_3)$ を生成し，T に挿入する．ただし，タスク生成プランにおいて，新たなタスクを生成するための生成同意が指定されている場合には，$DivideTask(result, n_3)$ を生成し，T に挿入する．

```
Initialize([n_1{v_1},..., n_m]) {
    T.insert(EntryTask(v_1, n_2));
}

GetTask(){
    return T.getOldestTask();
}

Performed(t, result, button, [n_1, ..., n_m]){
  If t is EntryTask(scope, n_i) {
    switch(button)
        case Insert:
          DB.insert(n_i, result);
          if (n_{i+1}.?y==0) {
             T.insert(EntryTask(result, n_{i+1}));
          else T.insert(DivideTask(result, n_{i+1}));
        case Delete:
          DB.delete(n_i, result);
          T.delete(EntryTask(result, n_{i+1}));
        case Check:
          if(完了合意済) {
                T.delete(scope, n_i);
          }
  } else if t is DivideTask(result, n_i) {
          if (分割合意済) {
             T.insert(result, n_i);
          }
  }
}
```

図 11　提案手法のアルゴリズム

- タスクが EntryTask であり Delete ボタンが押された場合: 結果 $result$ を DB から削除し，同時に，T から対応するタスクを除去する．
- タスクが EntryTask であり Check ボタンが押された場合: タスク生成プランの完了同意回数に達した場合には，T から完了したタスクを除去する．
- タスクが DivideTask の場合: $DivideTask(result, n_i)$ の結果が Yes であり，分割同意回数がとれれば，$EntryTask(result, n_i)$ を T に挿入する．

4. 評　　価

本節では我々が行った評価実験の結果を説明する．本実験は，提案手法がデータ収集の再現率を上げるのか，また，生成されたタスクはクラウドソーシングのタスクとして適切かどうか，の評価を行うものである．

4.1 実験方法

収集対象． 本実験では，収集対象として「新潟県の公営 (国営・県営・市営・町営・外部委託を含む) のプール名」を選択した．

```
[ 県{新潟県},
  地方 [!2]<地方名>,
  市町村 [!2,*]<市町村名>,
  区 [!2,?2]<区名>,
  *公営プール [!2]<公営プール名>]
```

図 12　実験に用いたタスク生成プラン P

表 1　再現率・適合率

	再現率	適合率	F 値
単純手法（T_{simple}）	0.68132	0.96875	0.80000
提案手法（T_{divide}）	0.79121	0.92308	0.85207

このデータ収集を選択した理由は，Web 上にこの項目だけをまとめて提供しているページが存在せず (簡単に一覧が手に入らず)，単純には見つからないことが確認できているからである．データ入力者は Web 検索エンジンの利用が許されるため，収集結果がそのまま Web 掲載されているような収集対象は実験目的にはそぐわない．したがって，この条件は重要である．事前に，筆者が Web 検索などを通じて手作業で正解集合を作成した．正解集合のプール名の個数は 91 個である．

ワーカ． 本実験のために，2 つのワーカのグループ A, B (それぞれ 6 人) を用意した．グループ A は，分割を行わない単純な手法（以下，単純手法）でデータ収集を行ない，一方，グループ B は，提案手法（以下，分割手法）を用いてデータ収集を行う．実験中は，ワーカ同士のコミュニケーションを許可しなかった．ワーカには事前にタスクのタイプと操作方法について説明を行なった．

タスク生成プラン． 分割手法では，図 12 に示すタスク生成プラン P を利用した．

提案手法合わせて，単純手法においても 2 回の完了合意が行なわれるとタスクがが完了するように設定し実験をおこなった．

4.2 実験結果

再現率・適合率． 実験の結果，単純な手法の場合は 62 個，分割手法では 72 個のプールがワーカによって入力された．正解集合と比較した場合の再現率・適合率を表 1 に示す．予想通り，再現率は分割手法の方が向上していることがわかる．単純手法の方が適合率が高かった理由は，主に 2 人のワーカによってデータの入力が行なわれて，他のユーザは積極的にデータ入力せずに，データの削除やデータが揃っているかの判断をしために，重複等が生じにくかったと考えられる．

総タスク処理数． 単純･分割手法の実験終了時の $TaskExcection$ テーブル (3.1 節参照) に含まれていたタスク (実際に行われたタスク処理) の集合を T_{simple}, T_{divide} とする．このとき，$|T_{simple}| = 104, |T_{divide}| = 428$ であった．これらは実際に実験を行った結果のタスク処理数であるので，間違った値を入力したり取り消したりするため，理想的な場合に比べると数が増える． 表 1 に内訳を示す．

これとは別に，ワーカがタスクを行なう際に，理想的にタスクを行なった (間違えなかった) 場合の単純．提案手法タスク集合をそれぞれ T'_{simple}, T'_{divide} とする． この場合，

表 2　タスクの内訳

	EntryTask			DivideTask	合計
	insert	check	delete		
T'_{simple}	91	2	0	0	91
T'_{divide}	133	86	0	60	279
T_{simple}	82	5	17	0	104
T_{divide}	179	141	44	64	428

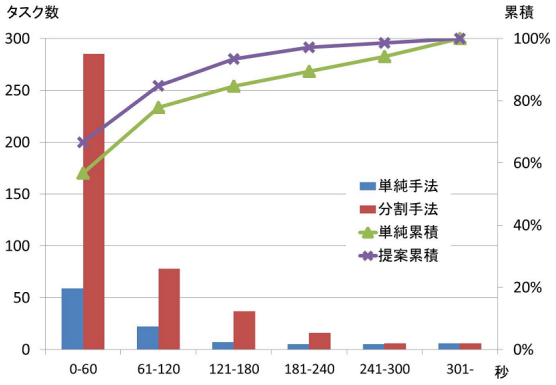

図 13　タスク処理時間の分布

$|T'_{simple}| = 93, |T_{divide}| = 279$ となる．

タスク処理時間とタスク処理時間分布．あるタスク t があるとき，ワーカが t を処理するためのタスク画面を表示してからボタンを押すまでの時間を処理時間 $time(t)$ する．タスク処理時間分布を図 13 に示す．単純手法における 1 タスクあたりの平均処理時間は約 97 秒であり，分割手法では 58 秒であった．分割手法では，タスク処理時間が 3 分以下のタスクが 92 %と短いタスクが多く，クラウドソーシングに向いたマイクロタスクを生成できていると言える．

タスク処理の集合 T が与えられた時，タスク処理時間の単純総和 $Time_{task}(T) = \sum_{t_i \in T} time(t_i)$ を計算すると，単純手法の総タスク処理時間は $Time_{task}(T_{simple}) = 10043$（約 167 分）であり，分割手法の総タスク処理時間は $Time_{task}(T_{divide}) = 24672$（約 411 分）であった．すなわち，タスク処理時間の単純総和に関しては，分割手法は単純手法と比較して約 2.5 倍の総タスク処理時間を必要とした．

しかし，分割手法においては，単純手法と比較してタスク間の依存関係が低くなり，独立に (並列して) 処理可能なタスクが数多く生成される．したがって，そのため，クラウドソーシングのワーカが十分多く確保できる場合には，並列して処理を行う事により，より短い時間で処理を終了可能である．独立作業可能なタスクの処理を並列化した場合をシュミュレーションすると，30 人以上のワーカが存在する場合には，分割手法は 3652 秒（約 61 分）で処理が終了することがわかった．

タスク処理に求められる能力の分布．ワーカが持ちうる能力の集合を $Abilities = \{a_1, a_2, \ldots\}$ とする．能力の例としては，英語を理解できる，阪神タイガースに関する知識がある，ある地域を訪ねることができる，などである．新潟県の公営プールの例については，「新潟県の公営プール全体に関する知識」「ある市町村の公営プールに関する知識」などがある．分割手法では，

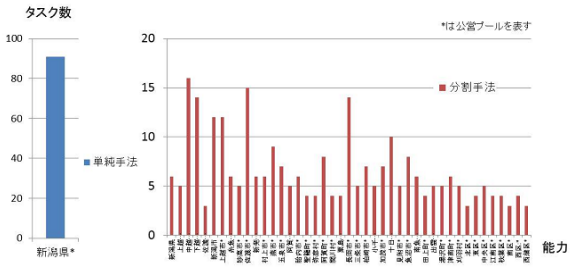

図 14　タスク処理に求められる能力の分布

特に，局所的な範囲における公営プールに関する知識もしくは見つける能力，が求められる．このとき，タスク t_i の処理に必要な能力を関数 $ra : T \to 2^{Abilities}$ で表す．

図 13 は，本実験で実際に行われたタスク処理を対象とし，横軸を，必要な能力集合 $Abilities$ の各項目 a_i とし，縦軸はそれを必要とするタスク処理数 (すなわち，$a_i \in ra(t)$ となる t の数) としたものである．単純な手法では，全てのタスク処理において新潟県の公営プール全体に関する知識 (知っている，見つけることが出来る) が必要とされるのに対し，分割手法では，各タスクの処理に必要な能力はそれぞれごく小さな範囲に局所化されている．これは，本実験においては分割手法の各タスクの処理時間を短くすることに貢献している．一般論としては，必要な能力の局所化度合いは，各タスクの処理時間を短くすることに貢献するほか，クラウドソーシングにおけるワーカへのタスク割当にも大きな影響を与える．

5. おわりに

本稿では，クラウドソーシングによって効率よくデータ収集を行うためのタスクの生成手法について提案した．データ収集の再現率を上げるためには問題を小さく限定したタスクを数多く用意する事が重要であるが，具体的にどのタスクを用意すべきかは問題固有であり，かつ一般にタスク数も多くなることから，個々の問題毎にプログラマがそれらのタスクを用意する事は煩わしい．本論文では，まずデータ収集のタスク生成プランをシステムに与え，ワーカのタスク処理の結果に応じてこのようなタスクを動的に生成する手法を提案した．本論文ではさらに，実験結果について説明し，提案手法によるデータ収集の再現率の向上が可能である事と，クラウドソーシングのタスクとして適切なタスクを生成する事を示した．今後の課題としては，タスク生成プランの作成支援やプランの妥当性の判断手法の検討，他の関連問題 (インセンティブなど) と組み合わせた検証などがあげられる．

謝　辞

ゼミなどにおいてコメントいただきました，筑波大学 杉本重雄教授，阪口哲男准教授，永森光晴講師に感謝いたします．本研究の一部は JST さきがけ「情報環境と人」による．

文　献

[1] 食べログ. http://tabelog.com/.
[2] Marcus Adam, Wu Eugene, Karger David R., Madden Samuel, Miller Robert C. . Human-powered Sorts and Joins.

PVLDB. 2011, vol. 5, no. 1, pp. 13-24.

[3] Marcus Adam, Wu Eugene, Madden Samuel, Miller Robert C. . Crowdsourced Databases: Query Processing with People. CIDR, 2011, pp. 211-214.

[4] Franklin Michael J., Kossmann Donald, Kraska Tim, Ramesh Sukriti, Xin Reynold. . CrowdDB: answering queries with crowdsourcing. SIGMOD Conference, 2011, pp. 61-72.

[5] Parameswaran Aditya G., Polyzotis Neoklis. . Answering Queries using Humans, Algorithms and Databases. CIDR.,2011, pp. 160-166.