パイプライン・クラウドソーシングにおける報酬制御を用いたスループットの向上

水澤　健† 田島　敬史†† 天笠　俊之††† 森嶋　厚行††††

† 筑波大学大学院システム情報工学研究科　〒 305-8573 茨城県つくば市天王台 1-1-1
†† 京都大学大学院情報学研究科　〒 606-8501 京都府京都市左京区吉田本町
††† 筑波大学計算科学研究センター　〒 305-8577 茨城県つくば市天王台 1-1-1
†††† 筑波大学 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: †ken.mizusawa.2016b@mlab, ††tajima@i.kyoto-u.ac.jp, †††amagasa@cs.tsukuba.ac.jp,
††††mori@slis.tsukuba.ac.jp

あらまし　クラウドソーシングにおいて，問題を複数種類のタスクに分割してシーケンシャルなワークフローを形成し，パイプライン処理を用いて依頼することで，処理の効率化や，結果の品質向上を図ることがある．しかし，単純なパイプライン処理では，ワーカによるタスクの選択に違いが現れることから，各タスクの処理量に偏りが現れ，パイプライン処理全体のスループットが低下するという問題が生じることがある．本研究ではこの問題に対し，タスクに設定する報酬を動的に操作することで，全体のスループットを向上させる手法を提案する．予備実験では報酬操作の有効性について，報酬操作を行わない場合と，小さく行う場合，大きく行う場合に処理時間に関してどのような影響をあたえるかを確認した．また，予備実験をふまえ，未処理のタスク数をベースとした報酬操作手法を提案し，実験によって，報酬操作を行わない場合に対して，スループットが約 30%向上する可能性があることを確認した．

キーワード　クラドソーシング，シーケンシャルワークフロー, パイプライン処理，処理最適化

1. はじめに

クラウドソーシングとは，ネットワーク上にいる不特定多数の人々(以下，ワーカ) に作業を委託することを指す [1] [2]．このクラウドソーシングにおいて，ワーカに委託する作業の単位を一般に「タスク」と呼ぶ．特にワーカにとって作業負担が小さく，特別な能力を必要としないようなタスクを「マイクロタスク」と呼ぶ．また，作業の依頼者 (以下，リクエスタ) は，自身の作業をクラウドソーシングを利用して依頼する場合、一般的にはクラウドソーシングサービスを提供するプラットフォームを通して行う．プラットフォーム上でリクエスタは作業に対する報酬として賃金を設定し，ワーカに掲示する．ワーカはタスクを処理する作業を行うことで，作業結果をリクエスタに納め，報酬として賃金を得ることができる．

リクエスタが解決したい問題に対して，どのようなタスクを形成して依頼する方法は，様々な種類が考えられる．その方法の 1 つに，複数の種類のタスクを解くことで問題を解決できるようなシーケンシャルなワークフローを形成するというものがある．図 1 はそのシーケンシャルなワークフローを用いた先行研究の例である．これは英語長文の校正を,「Find (発見する)」「Fix (修正する)」「Verify (検証する)」という 3 つの連続したタスクに分割して依頼するというものが先行研究では行われている [3]．この場合，まず長文校正を 3 種のタスクに分割しマイクロタスク化することで，一人あたりの作業量を削減したり，複数人の作業を通すことで翻訳文の品質を向上させるというような効果をもたらしている．さらにこのようなシーケンシャルなワークフローで構成されたタスクを，パイプライン処理することを考え，これをパイプライン・クラウドソーシングとして本研究で扱う対象とする．

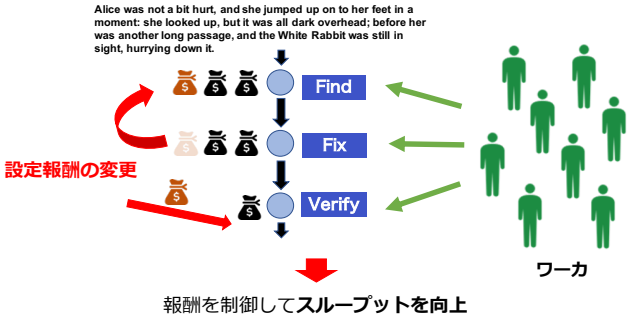

図 1　パイプライン・クラウドソーシングと報酬操作

しかし，単純にパイプライン処理を行うと，あるタスクにワーカの作業が集中，あるいは，全く作業が行われないために，他のタスクの生成に必要なデータが得られず，パイプライン処理が遅延する場合がある．これは，各タスクを処理した際に得られる報酬や作業の難易度によって，ワーカが作業対象として選択するタスクに偏りが現れてしまうことに原因があると考えられる．

そこで本論文は，複数タスクを用いたパイプライン処理をクラウドソーシングで行う場合に起こる，処理の遅延問題に取り組む．パイプライン処理におけるスループットを向上させることを目的に，図 1 の左部分に示すような，動的な報酬の変更操

作を行うことによる制御手法について提案する．予備実験では，報酬を変更せずに依頼した場合と，動的に変更した場合で複数タスクの処理時間にどのような影響をあたえるのかを確認し，実験として報酬操作アルゴリズムを用いた場合の結果について報告する．

本論文の構成は以下の通りである．第 2. 節では，報酬操作が実際に処理に与える影響を確認するために行った予備実験について報告する．第 3. 節では，予備実験より更に大きく報酬操作を行った場合の結果について報告する．第 4. 節では，予備実験を元に，報酬の値を操作する手法について提案する．第 5. 節では，予備実験の結果を元に行った実験結果について述べる．第 6. 節では，関連研究について述べる．第 7. 節では，まとめと今後の課題について述べる．

2. 予備実験 1

依頼中のタスクに対して報酬操作を加えると，処理速度にどのような変化が起き，スループットにどのような影響を与えられるかを確認するという目的で行った予備実験について述べる．本論文における予備実験を含めた実験の全ては Amazon Mechanical Turk(以下，MTurk) 上で行い，MTurk が提供している API を通じてタスクの作成や報酬の変更などを行った．依頼開始時間は全て日本時間の AM11:00 と PM10:30 の 2 つの時間帯を用いた．MTurk 上では図 2 のように掲載されているタスクの一覧から，ワーカは作業を行う前に，タスクの内容や報酬の値を確認することができる．

2.1 タスク設計

実験で用いたタスクの設計について説明する．ここでは与えられた文章の中から名詞を取り出して，その名詞について分類を行うという問題を想定した．この問題を，「文章から名詞を取り出す」「取り出した名詞が文章中に存在するか確認する」「取り出した名詞について分類をする」という 3 種類のタスクをパイプライン処理するワークフローを用いて行うことを想定する．以下この 3 種類のタスクをそれぞれ「Select」「Check」「Class」タスクとする．この流れを図 3 に示す．ただし，予備実験では 3 種類のタスクについて，予めワーカに全てのタスクを依頼できるように結果を用意しておき同時に依頼する．タスクの依頼数はそれぞれ 50 で，依頼してから 8 時間がワーカに提示される時間とした．

2.2 報酬設定

予備実験 1 で設定した報酬の値について説明する．設定は次の 2 つの指標を元に行う．

- パイプライン処理をしている複数タスクの中で，スループットが低いタスクの報酬を上げ，高いタスクの報酬を下げることで，処理全体としてのスループットのバランスを取る．
- 複数タスクに対し予め決められた基準の報酬から，見積もり可能な報酬の総額に近い値になるように，報酬の操作を行う．

今回は 3 種類の固定報酬と，動的な報酬操作を用いた場合の 4 種類の報酬の与え方を用いて依頼した．固定報酬については，0.02$，0.05$，0.08$の 3 種類を用いた．

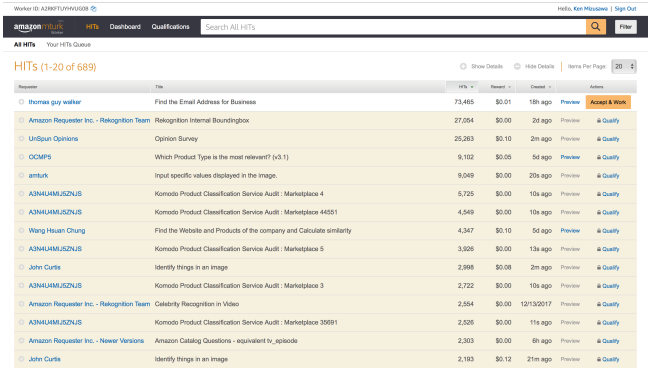

図 2　実際の MTurk 上でのワーカ側のタスク一覧画面

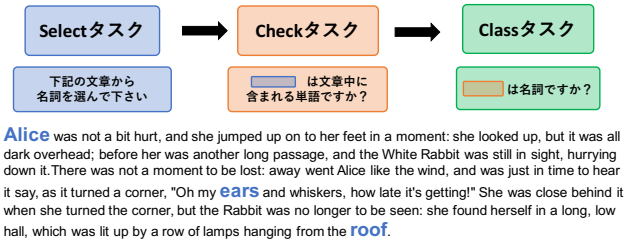

図 3　実験で用いるタスクとパイプラインの流れ

報酬操作について

報酬操作について次の基準を用いて実行した．

- 報酬操作はそれまでに 3 種類のタスクについて処理タスク数を元に設計する
- 基準値として報酬は 0.05$に設定し，3 種類のタスクについてそれぞれ 0.05$で依頼を開始する．それぞれのタスクの処理量に差がなければこの基準値を報酬の設定値として用いる．
- 依頼開始してから 2 分周期でタスクの処理量について監視し，各タスクの処理量に基準値以上の差が出た場合，報酬について変更する．
- 報酬の変更は 0.02$から 0.08$までの範囲で行う．3 種類のタスクに設定した報酬の総額が 0.15$になるように設計する．これは基準値が 0.05$で 3 種類のタスクを発行することから設定した値である (0.05$ × 3 = 0.15$)．

予備実験ではこの報酬操作に基づいて行った．例えば，最初は 0.05$で依頼を開始し，Class タスクだけ処理が速く進んでいる場合には Select，Check タスクはそれぞれ 0.06$，Class タスクは 0.03$に報酬を変化させる (0.06$ × 2+0.03 = 0.15$)．更に，Select タスクが Check タスクよりも処理が進行せず，処理数に差が出た場合は Select タスクを 0.08$，Check タスクを 0.05$，Class タスクを 0.02$に変化させるというような操作になる (0.08$+0.05$+0.02$ = 0.15$)．

2.3 結　　果

予備実験 1 では 4 種類の報酬の与え方と 2 つの時間帯での組み合わせにより 8 種類の依頼形式を用いた．表 1 は 3 種類のタスクを同時に依頼した際にかかった報酬の総額と，10 タスク終了，25 タスク終了，50 タスク全てが終了までに経過した時

表 1　依頼形式ごとの報酬の総額と，10 タスク，25 タスク，および，50 タスク処理における各タスクの処理時間

依頼開始時間		AM 11:00				PM 10:30			
報酬設定		0.02	0.05	0.08	change	0.02	0.05	0.08	change
支払った報酬の総額		3.00	7.50	12.00	**7.61**	3.00	**7.50**	12.00	**7.66**
10 タスク終了	Select	0:07:08	0:12:40	0:10:23	0:08:42	0:29:35	0:05:55	0:07:37	0:04:14
	Check	0:08:03	0:16:05	0:03:06	0:09:28	0:20:07	0:07:02	0:02:39	0:02:46
	Class	0:05:46	0:04:53	0:05:18	0:08:03	0:04:25	0:01:59	0:04:50	0:02:27
25 タスク終了	Select	0:34:23	1:35:33	0:45:49	0:31:20	2:28:49	0:24:03	0:28:25	0:10:04
	Check	0:37:52	1:54:35	0:19:41	0:36:19	2:37:25	0:24:45	0:10:58	0:09:48
	Class	0:18:37	1:34:28	0:41:27	0:33:28	0:38:21	0:08:31	0:14:08	0:10:56
50 タスク終了	Select	6:27:33	Expired	3:13:25	**1:33:47**	Expired	**1:42:24**	1:40:02	**0:38:59**
	Check	7:11:55	Expired	1:23:39	**1:44:08**	Expired	**2:14:48**	0:50:02	**0:37:29**
	Class	2:45:08	Expired	2:08:04	**1:33:41**	6:45:15	**0:48:16**	1:11:21	**0:34:31**

間ついてまとめたものである．また，報酬操作による制御を用いて依頼したものを「Change」と報酬設定の部分では表す．また，今回はタスクの掲示時間を 8 時間としたため，8 時間以内に完了しなかった場合，Expired と表記した．今回は AM10:00 に 0.05$固定で依頼した場合，また，AM10:30 に 0.02$固定で依頼した場合で Expired が発生した．報酬操作を行った依頼形式では AM 11: 00 の時には 7 回，PM 10:30 の時には 3 回の報酬操作を行う結果となった．

また，PM10:30 における「0.05$固定」と，AM11:00 における「Change」の場合の 50 タスク全てが終了するまでの，タスクの処理数と経過時間の推移について図 4 に示す．

固定の場合には Check と Class で処理時間に 1 時間半の差がついた．つまり，スループットに大きな偏りがあるといえる．対して変化させた場合には，3 つのタスクは約 10 分以内の差で全てのタスクが終了したことが分かる．

2.4　考　　察

結果より，報酬が高くなるにつれて全体的に処理が速く進む傾向が確認できた．一方で，3 つのタスクそれぞれについて見ると，Select，Check，Class の 3 つのタスクの処理終了時間の差が大きくなる傾向も同時に確認できた．

今回の報酬操作は，各タスクにおけるタスクの処理量を周期的に監視し，基準を用いて相対的に報酬を操作するということのみ行った．したがって，3 つのタスク全体のスループットが低下した場合の制御については行っていないことになる．表 1 の 10 タスク終了時点での経過時間を見ると，報酬の値にかかわらず，最初の段階でのスループットは高い．しかし，処理されるタスク数が増えるに連れて処理時間は上昇している．今回の実験では全てのタスクが 50 タスクで依頼されていたために Expired に陥るケースも少なかったが，問題の規模によってはパイプライン処理が完全に途中で停止してしまう可能性が高くなると考えられる．

2.4.1　報酬の高さが与える影響

クラウドソーシングで作業をするワーカにとって，簡単な作業で多くの報酬を得られるほうが好ましい．このことから，報酬が高いタスクの方がワーカによって選択されやすいということは理解しやすい．報酬が高いほどタスクの処理が速く進み，

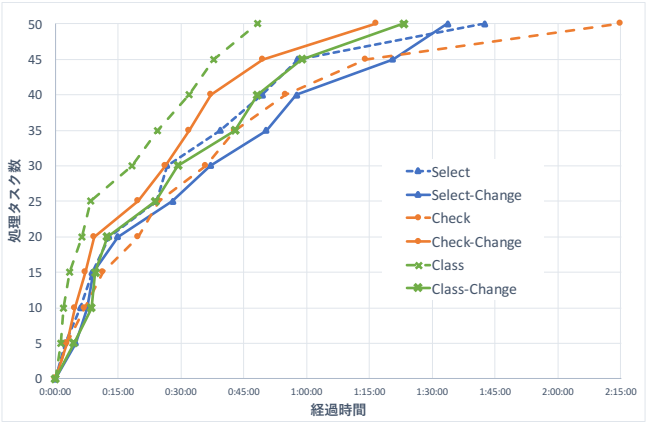

図 4　0.05$固定と 0.05$基準で 0.02$～0.08$の変化を用いた場合の処理タスク数の時間推移

スループットが高くなる．しかし，リクエスタ側の立場からはワーカに支払う報酬は低いほうが好ましい．よって報酬を単純に上げるのではなく，タスク完了までに最終的にかかる報酬の総額などを考慮した上で報酬操作を行う必要があると考えられる．

2.4.2　依頼開始からの経過時間が与える影響

プラットフォーム上には，様々なリクエスタから新しいタスクが依頼されていくため，タスクが依頼されてから時間が経過するほど，タスクの処理速度は低下するという傾向がある．タスクを依頼してから時間が経過するにつれて，古いタスクはワーカからの興味が少なくなるなど，依頼したタスクが未処理のまま残り続けないようにする必要があると考えられる．

3.　予備実験 2

予備実験 1 と同じ設計のもと追加で行った予備実験について述べる．使用したデータや環境は同じものを使用した．予備実験 2 では同じく同時に依頼した 3 つのタスク処理について，報酬操作について変化の度合いを大きくしたものを用いる．予備実験 1 では報酬の値を 0.02$ から 0.08$ の間で変更を行ったが，予備実験 2 では次の条件に従って変更を行った．

- 依頼中のタスクについてタスクの処理数を計算し，ス

ループットを算出，また，その平均値を計算する．

- タスクのスループットが平均値を下回る場合は報酬の値を 0.08$ に設定し，上回る場合は報酬の値を 0.02$ に設定するこれにより，報酬は 0.02$ あるいは 0.08$ のどちらかに設定される．例えば 3 つのタスクのスループットがそれぞれ，1.0, 1.2，1.5 になった場合，最初の 2 つは報酬を 0.08$，3 番目は 0.02$ に設定する．この報酬変更を，予備実験 1 と同じく 2 分周期で行った．

3.1 結　　果

予備実験 2 の結果を表 2 に示す．操作を行った場合を 2 回，行わずに 0.05$ で一定の場合を 1 回行った．操作を行った 2 回の実験についてはそれぞれ，45 分，24 分程度で 3 つのタスクが終了している．対して操作を行わなかった場合については Class タスクだけが 14 分と他の 2 つのタスクの半分以下の時間で終わっている．支払った報酬の総額に着目すると, 行わない場合の 7.50$ に対して，8.48$，8.28$ となった．報酬を変更しない場合に比べて，約 0.70$ から 0.90$ 増加している．予備実験 1 ではそれぞれ，約 0.10$ の増加であったことと比べると，報酬の変更を大きく行うことで，結果的に報酬総額が高くなる傾向が見られた．

3.2 考　　察

報酬の変更を大きく行うことを考え，報酬の値を範囲内で緩やかに変更するのではなく，設定した最小値か最大値のどちらかに設定するという手法を用いた．その結果，全体的に処理が早く終了した一方で，報酬を変更しない場合についても，予備実験 1 の場合よりも処理が早く終わるという結果を得られた．加えて，報酬操作を行った場合の支払った報酬の総額が，予備実験 1 の結果と比べて上がっており，割合で見ると，10%の増加になっている．予備実験 1 では報酬総額について考慮していたため，変更を加えてもおよそ 1%の増加に留める事ができていた．これらから報酬変更は予め定めた予算の中で大きく変更するのが適切であると考えられる．

4. 提 案 手 法

予備実験の結果を元に提案する報酬操作手法について述べる．提案手法ではスループットではなく処理すべき残りのタスク数をベースとした報酬操作手法について提案する．予備実験 1 では処理タスク数をベースに，予備実験 2 ではタスク処理のスループットをベースにした．これらを踏まえて提案手法では報酬の総額 (=予算) を考慮しつつ，残りタスク数をベースにしたものを考える．

次から手法を説明するためのパイプラインモデルの設計と報酬操作のアルゴリズムについて説明する．

4.1 設　　計

図 5 に提案手法におけるパイプライン処理のモデルを示す．

パイプライン処理において扱うタスクをタスク列 T として $T = [t_1, t_2, \ldots, t_i, \ldots, t_N]$ $(1 \leqq i \leqq N)$ と表す．

パイプライン処理全体で使うことのできる報酬の総額を $Total\_Budjet$ とする．パイプライン処理における依頼タスク数を $B \in \mathbb{N}$ とする．

表 2　依頼形式ごとの報酬の総額と，10 タスク，25 タスク，および，50 タスク処理における各タスクの処理時間 2

実験回数		1 回目	2 回目	3 回目
報酬制御		あり	あり	なし
支払った報酬の総額		8.49	8.28	7.5
10 タスク終了	select	0:07:26	0:05:21	0:05:23
	check	0:03:27	0:05:59	0:05:06
	class	0:07:25	0:03:13	0:02:49
25 タスク終了	select	0:18:18	0:11:30	0:10:55
	check	0:18:19	0:10:29	0:11:37
	class	0:15:44	0:09:04	0:08:03
50 タスク終了	select	0:49:37	0:24:27	0:35:34
	check	0:45:36	0:24:01	0:30:08
	class	0:43:33	0:23:46	0:14:08

各タスクの最初の設定報酬の値を $Initial\_Reward$ とし，全てのタスクの報酬として均等に設定されるとする．したがって条件から次のように定義できる．

$$Initial\_Reward = \frac{Total\_Budjet}{B \times N} \tag{1}$$

また，提案手法の中で各タスクの報酬として設定できる最小の値を Min_R, 最大の値を Max_R とする．このとき，$Initial\_Reward$ は

$$Min\_R \leqq Initial\_Reward \leqq Max\_R \tag{2}$$

を満たす必要がある．

また，パイプライン処理を行うと，T の中の最初の列から全てのタスクの処理が終了していくと考えられる．タスク列 T の中で，依頼中であるタスクのうちの先頭のものを $i = x$ とし，末尾のものを $i = y$ とする．

タスク t_i において，最初のタスク (t_1) の依頼を開始してからの経過時間 (min) を j とする．また，期間 $[0, j]$ までの間にタスク t_i において処理されたタスク数を $\#tasks_{t_i}$ とする．各タスクにおける時刻 j の報酬の設定額を r_{t_i} とする．

これらを用いて $Initial\_Reward$ から r_{t_i} に報酬を変化させながらパイプライン処理を依頼する．このとき変更した後の r_{t_i} に対して，次のように報酬に変更を加える．

$$r_{t_i} = \begin{cases} Min\_R \ (r_{t_i} < Min\_R) \\ Max\_R \ (r_{t_i} > Max\_R) \end{cases} \tag{3}$$

また，変化の度合いを強めるパラメータとして p $(\geqq 1)$ を設定する．p の値が大きくなるほどパイプライン中のタスクに設定される報酬の値に差が現れるようになるように設定し用いる．

4.2 残りタスク数を元にした報酬変更

依頼中の残りタスク数を元にした報酬変更手法について述べる．時刻 j における，タスク t_i における残りタスク数は定義より $B - \#tasks_{t_i}$ と表すことができる．ここで残りタスク数を元にした報酬を変化させる関数を $Change\_Reward$ として，次のように定義する．

$$Change\_Reward(T, B) \tag{4}$$

$Change\_Reward$ は，各タスクについて次のように報酬を設定する．

$$r_{t_i} = Initial\_Reward \times (y - x + 1) \times \{(\frac{B - \#tasks_{t_i}}{\sum_{i=x}^{y} B - \#tasks_{t_i}})^p\}.normalize \quad (5)$$

タスク列の中で報酬を設定する必要がある分の $Initial\_Reward$ をタスクがどれだけ処理されているかに比例して再分配するという処理を行っている．この比率にはパラメータ p が影響し，値が大きいほど比率が大きくなるため，変化が大きくなる．また，報酬を変更した際に最終的にかかる総額が変更しない場合に比べて大きくずれが発生しないように，算出した比率に関して 1 に正規化 (normalize) する．

この正規化を行うため，$Change\_Reward$ 関数の実行後の r_{t_i} について

$$\sum_{i=x}^{y} r_{t_i} = 1 \quad (6)$$

を満たしていることになる．

$Change\_Reward$ を用いた報酬設定例

$Initial\_Reward = 0.04$，$B = 50$，タスク t_1, t_2, t_3 のタスク処理数 $\#tasks$ がそれぞれ $30, 20, 10$ であるとする．$p = 1$ の場合で報酬操作を行った場合は，それぞれ報酬操作を行うと $r_{t_1}, r_{t_2}, r_{t_3}$ はそれぞれ 0.0267\$，0.040\$，0.0533\$ になる．これを $p = 3$ でおこなった場合は，それぞれ 0.0096\$，0.0327\$，0.0775\$ になる．

MTurk の制約を考慮した変更

しかし，本実験では MTurk を使用しており，例のような値をそのまま反映させることはできないので次の操作を加える．MTurk ではタスクに対して設定できる報酬の最小単位が 0.01\$ であるため，報酬を変更した後，この制約を満した形で再設定する必要がある．したがって，算出した値に対して切り捨て計算を行い，報酬の最小値と最大値を考慮した上で設定する．前の例で説明すると，$p = 1$ の場合にはそれぞれ 0.02\$，0.04\$，0.05\$，$p = 3$ の場合には 0.01\$，0.03\$，0.07\$ という値に最終的になる．

4.3 アルゴリズム

第 4.2 で説明した関数を用いた報酬変更アルゴリズムを Algorithm 1 に示す．全てのタスクの処理が終わるまで，一定周期でタスクの処理数を計算し，$Change\_Reward$ を用いた報酬変更操作を繰り返す．この一定周期を繰り返す間隔を $Round\_Time(min)$ とする．

1 行目から 3 行目までは $Total\_Budjet$ を元に全てのタスクについて初期設定報酬額である $Initial\_Reward$ を設定する．4 行目からは全てのタスクが終了するまで $Round\_Time$ ごとに処理が行われているタスクについてスループットを計算し，報酬を変更する．5 行目から 7 行目は全てのタスクの処理タスク数を計算する．8 行目から 10 行目までは，次の計算までに設定するタスクの報酬の値について報酬変更関数 $Change\_Reward$ を用いて決定する．11 行目では処理実行の待ち時間になる．

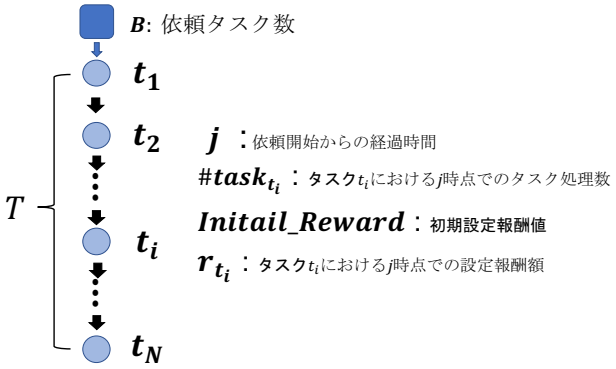

図 5　提案手法におけるモデル

Algorithm 1 報酬操作アルゴリズム

Input: T, $Total\_Budjet$, , $Round\_Time$
Output: $ResultofTask$

```
for i = 0 .. N do
    Setting Initial_Reward_{t_i}
end for
while #tasks_{t_N} == B do
    for i = 0 .. N do
        CalcurateSubmitTaskNumberof t_i
    end for
    for i = 0 .. N do
        Change_Reward
    end for
    WaitforNextRoundTime
end while
```

5. 実　　験

第 4. 節で説明した報酬操作のアルゴリズムを使った実験について述べる．この実験で用いたタスクの内容や設計，使用した実験環境などは第 2. 節などで行った，予備実験と同じものを使った．

5.1 設　　定

実験では「報酬を変更しない場合」および「提案手法を $p = 1$ で用いた場合」「提案手法を $p = 3$ で用いた場合」の 3 つの場合について行った．以下これらのケースをそれぞれ No_Change，$Change\_p1$，$Change\_p3$ と表記することにする．

$B = 50$ で実行し，各場合について 5 回実験を行った結果についてまとめる．報酬は $Initial\_Reward = 0.04$\$ となるように設定した．つまり，報酬を変更しない場合について $Total\_Budjet = 6.00$\$ となるように設定した．$Round\_Time = 2\ (min)$ で $Change\_Reward$ を用いる．各タスクについては t_1 が Select，t_2 が Check，t_3 が Class に相当する．

5.2 結　　果

表 3 は各試行において Selcet,Chack,Class の 3 つのタスクを全て完了するまでの，10 タスクごとに処理した時点での経過時間および支払報酬内訳・総額をまとめたものである．時間は最初のタスク，つまり，Select タスク (t_1) が開始してからの経過時間 ($= j$) を基準としている．また，報酬の単位は\$である．

表 3　各手法における 10 タスク処理時点での処理経過時間の平均値と支払報酬内訳・総額

使用手法	No_Change			Change_p1			Change_p3		
タスクの種類	Selelct	Check	Class	Selelct	Check	Class	Selelct	Check	Class
10 タスク終了	0:15:36	0:24:24	0:30:00	0:09:36	0:24:48	0:32:48	0:13:12	0:29:12	0:34:24
20 タスク終了	0:47:12	1:02:00	1:11:12	0:23:12	0:53:36	1:02:24	0:39:12	0:51:36	1:02:00
30 タスク終了	1:30:24	1:48:00	2:06:00	0:51:36	1:33:36	1:42:00	1:09:36	1:25:36	1:35:36
40 タスク終了	2:50:48	3:01:12	3:30:00	1:34:24	2:02:00	2:11:36	2:00:24	2:30:00	2:40:00
50 タスク終了	4:26:24	5:02:24	5:39:12	2:29:36	2:42:24	3:26:00	2:58:00	3:16:24	4:04:48
報酬内訳	2.00	2.00	2.00	1.23	1.90	2.33	1.05	1.82	2.72
報酬総額	6.00			5.47			5.59		

表 4　各試行におけるタスク完了時間と支払報酬総額

使用手法		No_Change			Change_p1			Change_p3		
タスクの種類		Selelct	Check	Class	Selelct	Check	Class	Selelct	Check	Class
1 回目	終了時間	4:38:00	4:50:00	6:14:00	1:12:00	1:28:00	1:34:00	4:26:00	5:00:00	5:28:00
	報酬総額	2.00	2.00	2.00	1.15	1.80	2.47	1.31	1.72	2.58
2 回目	終了時間	6:50:00	6:56:00	7:10:00	2:24:00	2:34:00	4:02:00	1:42:00	1:50:00	2:34:00
	報酬総額	2.00	2.00	2.00	1.09	2.01	2.38	0.80	1.97	2.77
3 回目	終了時間	3:34:00	5:22:00	6:24:00	2:42:00	2:48:00	3:26:00	3:28:00	3:40:00	3:46:00
	報酬総額	2.00	2.00	2.00	1.21	1.94	2.43	1.13	1.86	2.64
4 回目	終了時間	3:44:00	4:28:00	4:42:00	4:18:00	4:40:00	4:50:00	3:40:00	3:44:00	3:54:00
	報酬総額	2.00	2.00	2.00	1.48	1.81	2.09	1.08	1.83	2.81
5 回目	終了時間	3:26:00	3:36:00	3:46:00	1:52:00	2:02:00	3:18:00	1:34:00	2:08:00	4:42:00
	報酬総額	2.00	2.00	2.00	1.24	1.95	2.29	0.92	1.71	2.80

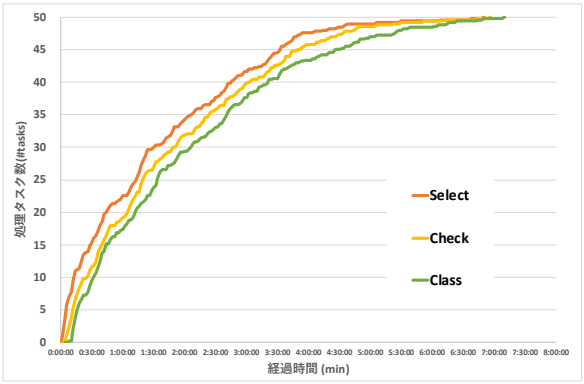

図 6　'No_Change' の場合の処理時間におけるタスク処理数の平均値

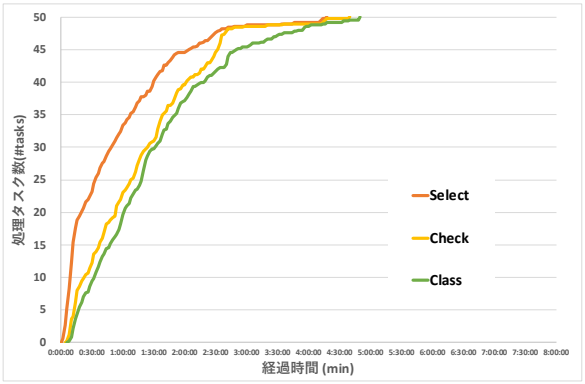

図 7　'Change_p1' の場合の処理時間におけるタスク処理数の平均値

表 4 は，各試行におけるタスクが完了した時間と支払報酬総額についてまとめたものである．

図 6 は報酬操作を行わない，No_Change の場合の，処理時間におけるタスク処理数の平均値を示したグラフである．同様にして，図 7，図 8 はそれぞれ Change_p1，Change_p3 を用いた場合の，処理時間におけるタスク処理数の平均値を示したグラフである．

全 5 回試行おこなった結果の平均を見ると，NO_Change に対して，報酬操作を行った Chanege_p1，Changeg_p3 が処理が早く終了したことを示している．支払報酬総額に関しても，基準とする 6.00$ を超えない範囲で実行することができている．各試行ごとの結果を見ても，全ての試行で基準報酬範囲内で終了することができている．また，Change_p1 と Change_p3 に関して，パラメータの影響が報酬に関して現れている．Change_p1 では Select と Check，Check と Class の各タスクの支払報酬総額の差が 0.67$，0.43$ であるに対して，Change_p3 では 0.77$，0.90$ と p1 よりも差が大きくなっている．しかし，最終的な処理時間に関しては約 30 分ほど *Change_p1* の方が速い．

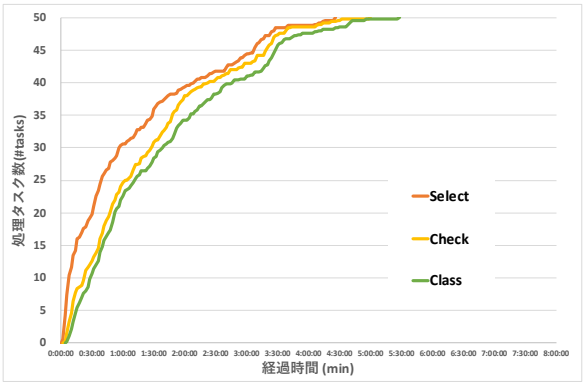

図 8　'Change_p3' の場合の処理時間におけるタスク処理数の平均値

各手法を用いた場合の経過時間におけるタスク処理数の平均値のグラフを見ると，どの手法においても後半になるつれて処理が遅くなる傾向が見られる．(1)t_1 あたる Select の最初の 10 タスクが終了するまでの処理時間 (2)t_3 にあたる最後の 10 タスク (41 タスク目から 50 タスク) の処理時間を比較してみる．(1) は手法にかかわらず 20 分以内，(2) は操作を行った Change_p1 ,p3 では 1 時間，行わなかった No_Change は 2 時間以上を必要とした．

5.3 考　　察

全 5 回の実験の結果について考察する．まず，実行時間帯，報酬支払方法にかかわらずタスクの処理は前半のほうが速く，後半になるにつれて遅くなる傾向があることが分かる．図 9 は各手法における，タスク列の最後尾，ここでは Class タスクに関して，処理時間におけるタスク処理数の平均値を比較したものである．全体的に緩やかな曲線を描いており，最初から最後まで一定のスピード，つまりスループットが一定には処理が進まなかったことが言える．本論文で提案した残りタスク数をベースに用いた報酬操作は，後半のスループットの低下を和らげることはできたが，維持，あるいは向上させるにはいたらなかった．

ここで，支払った報酬の総額について着目すると，報酬を変更しない No_Change に対して，Change_p1,p3 は約 0.5$ 低い．本研究が目的とするのは報酬の削減ではなくスループットの向上であるため，仮にこの設定予算に満たない 0.50$ を有効活用することでよりよい結果を得られるのならば，そちらのほうがよい．提案手法では確かに報酬操作関数 *Change_Reward* を用いて各タスクに対して報酬を設定しているが，計算した結果，結局のところ報酬が変わっていない時間帯の方が長い試行も見られた．図 10 は Chnage_p1 を用いた場合の報酬変更がどのように行われたか，図 11 は Chaneg_p3 を用いた場合の報酬変更がどのように行われたかをある 1 回の試行を取り上げて示したものである．前者は，報酬操作があまり行われず，例えば t_1 に該当する Select タスクの報酬は最初に 0.02$ まで低く設定された後，ほぼ最後まで変わらず，また，t_2 に該当する Check タスクにおいては初期設定である 0.04$ からほとんど変更が行われていない．これに対し，Change_p3 では報酬変更が行われた回数が多かったものの，処理時間を見るとこの 2 つの試行では Change_p1 のほうが処理が速く終了している．全体として，操作を加えるほうがパイプライン処理のスループット向上に対して良い傾向は確認できたが，処理全体が停滞したときなど，他のタイミングで加える事ができるか，また，報酬の設定額の変更値についてはさらなる検討が必要であると思われる．

6. 関 連 研 究

タスクの報酬を見積もることや変化させることで，クラウドソーシングでの処理を効率化する研究がこれまで行われてきた [4] [5] [6]．Gao らは [7]，予め与えられた予算とタスク完了期限の条件下で，タスクの処理速度に基準に，報酬を変更することで全体でかかる報酬の総額を削減しつつ処理速度を向上させるアルゴリズムを提案した．本研究では Gao らが取り組んだ

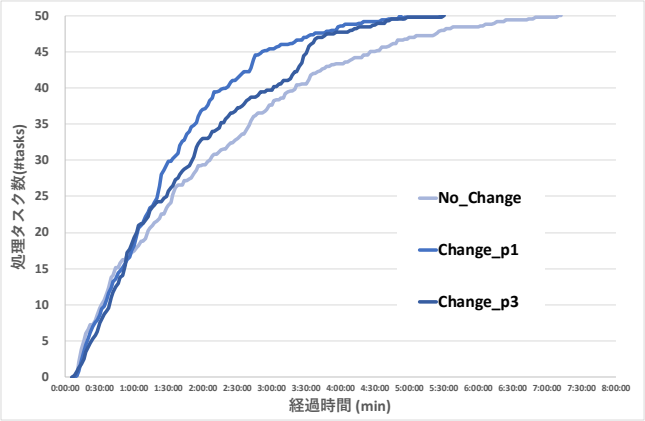

図 9　各手法を用いた場合の処理時間における Class タスク (= t_3) の処理数の平均値

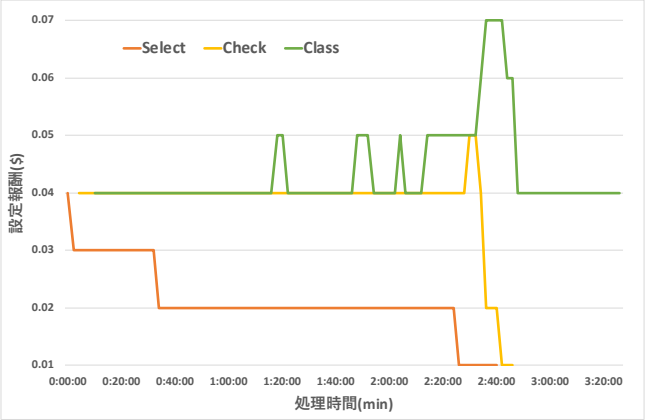

図 10　Chaneg_p1 を用いた場合の処理時間における報酬操作の例

図 11　Chaneg_p3 を用いた場合の処理時間における報酬操作の例

単一のタスクに対しての操作ではなく，複数タスクがパイプライン処理される関係性を持った中での報酬操作による制御に取り組む．Parameswaran らが提案した DECO [8] というシステムでは，SQL のようなクエリ記述言語を元に，クラウドソーシングのタスクを依頼する仕組みを提案した．システムがクエリの最適化を行う中で，予め報酬についても見積もりを立て依頼をすることで報酬の削減をした．本研究では予めの基準をもとに，結果に応じて動的に報酬を操作することを考える．

ワークフローを用いたクラウドソーシングでは，Chilton

ら [9] はデータ分類を行うためのワークフローを提案した．Tran ら [10] は，複雑なワークフローを形成するクラウドソーシングにおいて，予算を効率的に割り当てる手法について提案した．Goto ら [11] は，クラウドソーシングのワークフローについて分析し，タスクの種類などを考慮した最適化モデルを提案した．複数タスク処理の仕組みとして，Ambati ら [12] は文章翻訳を行う際に，能力の低いワーカから得られた品質の低い翻訳文を，能力の高いワーカに再び翻訳タスクとして割り当てることで，翻訳結果の品質向上をはかるワークフローを提案した．Peng ら [13] は，報酬を抑えつつタスク結果の品質を向上させるために，AI を利用し，効率的なワークフローの形成をした．Anand ら [14] は，難しいタスクを幾つかのタスクにワーカによって分割させ，タスクの解答精度の向上をはかった．本研究では，ワークフローを元に複数タスクをパイプライン処理することで問題を解決するクラウドソーシングを対象とし，それらのタスクを処理する上での制御問題に取り組む．

7. まとめと今後の課題

本研究では複数タスクをパイプライン処理するワークフローを用いた場合に起こる問題に対し，動的に報酬を変更することでスループット向上させる手法について提案した．２つの予備実験では，同時に依頼した複数タスクに対する相対的な報酬操作が，処理速度に影響を与えてる可能性を確認した．それらを踏まえた上で，残りタスク数をベースとした報酬操作手法を提案した．報酬操作を行わない場合との比較実験では，報酬操作を行ったほうが全体の処理は早く進む傾向があることが得られた．実際に 5 回繰り返し行った試行の平均を比較すると，全体として約 30%ほどスループットが向上する結果が得られた．

また，パイプライン上の全てのタスクに関して処理が行われない時間帯が，特に処理が後半になるにつれて多く存在した．しかし，提案した報酬操作では残りタスク数しか見ていないため，停滞状態に対して変化を加えることができていない．今後は停滞状態が続いた場合に追加で報酬操作を加えるなど，今回提案した操作の他に，処理速度に影響を与え，スループットが向上するような操作について加えられないかを検討していきたい．

謝　辞

本研究の一部は JST CREST (#JPMJCR16E3) の支援による．

文　献

[1] Edith Law, Luis von Ahn, and Luis Von Ahn. Human computation. *2008 Ieee 24th International Conference on Data Engineering, Vols 1-3*, Vol. 5, No. 3, pp. 1–121, 2005.

[2] Jeff Howe. The rise of crowdsourcing. *Wired Magazine*, Vol. 14, No. 06, pp. 1–5, 2006.

[3] Michael S Bernstein, Greg Little, Robert C Miller, Björn Hartmann, Mark S Ackerman, David R Karger, David Crowell, and Katrina Panovich. Soylent: a word processor with a crowd inside. *Communications of the ACM*, Vol. 58, No. 8, pp. 85–94, 2015.

[4] Siamak Faradani, Bjoern Hartmann, and Panagiotis G. Ipeirotis. What's the right price? pricing tasks for finishing on time. In *Human Computation, Papers from the 2011 AAAI Workshop, San Francisco, California, USA, August 8, 2011*, Vol. WS-11-11 of *AAAI Workshops*. AAAI, 2011.

[5] Daniel W Barowy, Charlie Curtsinger, Emery D Berger, and Andrew McGregor. Automan: A platform for integrating human-based and digital computation. *Acm Sigplan Notices*, Vol. 47, No. 10, pp. 639–654, 2012.

[6] Zehong Hu and Jie Zhang. Optimal posted-price mechanism in microtask crowdsourcing. In *IJCAI International Joint Conference on Artificial Intelligence*, pp. 228–234, 2017.

[7] Y Gao and A Parameswaran. Finish them!: Pricing algorithms for human computation. *Proceedings of the VLDB Endowment*, Vol. 7, No. 14, pp. 1965–1976, 2014.

[8] Aditya Ganesh Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: declarative crowdsourcing. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 1203–1212. ACM, 2012.

[9] Lydia B. Chilton, Greg Little, Darren Edge, Daniel S. Weld, and James A. Landay. Cascade: Crowdsourcing taxonomy creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pp. 1999–2008, New York, NY, USA, 2013. ACM.

[10] Long Tran-Thanh, Trung Dong Huynh, Avi Rosenfeld, Sarvapali D. Ramchurn, and Nicholas R. Jennings. Crowdsourcing complex workflows under budget constraints. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pp. 1298–1304. AAAI Press, 2015.

[11] Shinsuke Goto, Toru Ishida, and Donghui Lin. Understanding crowdsourcing workflow: Modeling and optimizing iterative and parallel processes. In *Fourth AAAI Conference on Human Computation and Crowdsourcing*, 2016.

[12] Vamshi Ambati, Stephan Vogel, and Jaime Carbonell. Collaborative workflow for crowdsourcing translation. *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW '12*, p. 1191, 2012.

[13] Peng Dai, Christopher H Lin, Daniel S Weld, et al. Pomdp-based control of workflows for crowdsourcing. *Artificial Intelligence*, Vol. 202, pp. 52–85, 2013.

[14] Anand Kulkarni, Matthew Can, and Björn Hartmann. Collaboratively crowdsourcing workflows with turkomatic. *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW '12*, p. 1003, 2012.