クラウドソーシングと計算機による処理の統合記述方式

林　真史† 田島　敬史†† 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
†† 京都大学大学院情報学研究科, 〒 606-8501 京都府京都府京都市左京区吉田 36 番地 1 号
††† 筑波大学 知的コミュニティ研究センター, 〒 305-8550 茨城県つくば市春日 1-2
E-mail: †masafumi.hayashi.2016b@mlab.info, ††tajima@i.kyoto-u.ac.jp, †††mori@slis.tsukuba.ac.jp

あらまし　近年，クラウドソーシングと計算機処理を組み合わせた処理が注目を集めている．本稿では，これらを組み合わせた実行プランを段階的・包括的に記述可能な手法の提案を行う．これにより，人と計算機を組み合わせた処理の最適化や，最初は人からスタートして徐々に計算機に置き換えると言った処理を体系的に議論することが容易になる．本稿では，このような実行プランの記述方式が満たすべき条件を示し，それらを満たす記述方式を提案する．また，人だけでなく計算機によるワーカ (アルゴリズムワーカ) を想定したシナリオを題材に，提案方式による実行プラン記述を用いたコスト見積りや最適化などを行う際の課題について議論する．

キーワード　クラウドソーシング，問い合わせ処理

1. はじめに

近年，問題解決のために必要な作業（以下，タスク）を，ネットワーク上の群衆（以下，ワーカ）に依頼するクラウドソーシング [1] が注目を集めている．

クラウドソーシングは，計算機のみでは解決困難な問題に対し有効な手法として知られているが，単純な計算など，計算機の方が有効な手法となる場合も多い．そのため，単純な計算と計算機のみでは解決困難な問題の双方を含む問題に対しては，二つの手法を組み合わせて問題解決を行うことが重要である．

本稿では，人と計算機の組み合わさった処理を包括的に記述可能な「実行プラン」の記述方式を提案する．ここでいう実行プランとは，クラウドソーシングの依頼者 (リクエスタ) が，クラウドソーシングのタスクや計算機処理を組み合わせた複雑な処理を記述するものである．これにより，人と計算機の処理を組み合わせた複雑なクラウドソーシングの設計や処理の効率化を議論する事が容易になる．

本稿では，まず，提案する実行プランの記述方式が満たすべき条件と，それらの条件を満たす記法を提案する．また，その記述の意味を定義する．次に，人だけでなく計算機によるワーカ (アルゴリズムワーカ) を想定したシナリオを題材に，提案方式による実行プラン記述を用いたコスト見積りや最適化などを行う際の課題について議論する．

本稿の貢献は次の通りである．

(1) 複雑なクラウドソーシング記述のための汎用的な実行プラン記述方式の提案. これまで，クラウドソーシング処理の実行プランについてはいくつか提案されてきているが [2] [3]，本稿で提案する実行プラン記述方式は，それらよりも記述力が高いものである．

(2) 人間のワーカとアルゴリズムワーカを組み合わせた処理に関する議論. 本稿では，提案する実行プランを用いて，人だけでなく計算機によるワーカ (アルゴリズムワーカ) を想定したシナリオを題材に，提案方式による実行プラン記述を用いたコスト見積りや最適化などを行う際の今後の課題について議論する．クラウドソーシングでアルゴリズムワーカが存在するという可能性自体はこれまでも広く認識されてきた．本稿では，実際にアルゴリズムワーカをオープンコールする際のコスト見積り等についての具体的な課題を議論する．

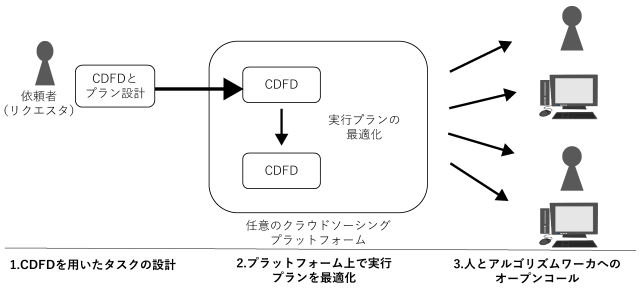

図 1　CDFD を用いたタスク発行までの流れ

本稿の構成は次の通りである．まず，2. 節では人と計算機による処理を記述するのに求められる条件について述べる．3. 節では，関連研究を述べ本稿の位置づけを明らかにする．4. 節では，人と計算機のよる処理の実行プランを書くための図である Crowd Data Flow Diagram について述べる．5. 節では，伝統的な実行プランとクラウドソーシングにて扱われる実行プランの性質の違いについて述べる．6. 節では，我々の提案する評価式を用いた実行プランの見積もりについて述べる．7. 節では，本稿で提案する実行プランを活用したシナリオの提示とその課題について述べる．

2. 人と計算機による処理の記述に求められる条件

クラウドソーシングを通じて，人と計算機による処理を共に

オープンコールし，これらによる処理を表す実行プランの設計・処理に利用する記法は，次の条件を全て満たす必要がある．

（1） 扱うデータを記述可能である．

（2） 行う処理を記述可能である．

（3） 処理を束ねた “タスク” の単位を記述可能である

（4） 処理と，処理を行う主体 (人間，計算機) の独立性を保つ

（5） 処理を行う主体の条件を記述可能である．

通常のデータベース処理などで利用される実行プランでは，(1), (2) に関しては記述可能であるが，(3)-(5) の条件が満たされれない．(3) は，クラウドソーシングにて，タスクの単位と処理の単位が異なることから，必要な条件である．(4) は，クラウドソーシングに人もしくは計算機に依存した処理が存在しないことを表すために必要な条件である．この条件から，どんなタスク・処理も人間か計算機に割り当てることができるようになる．(5) は，処理の内容に応じて，どういったワーカに委託するべきかが，プラン設計に重要な要素であることから，必要な条件である．また，以上の性質に加え，実行プランの設計に利用する事を考えた場合，簡略的な処理から順次詳細化するような設計で利用可能な仕組みである必要がある．

3. 関 連 研 究

伝統的なデータベースでは，リレーショナル代数による問合せ最適化が行われてきた．実行プラン最適化における，Logical Plan では，リレーショナル代数で表され，Phisical Plan は実装レベルの詳細を記述した演算子が利用されてきた [4]．本稿では，Logical Plan にあたる記述が可能であり，また Phisical Plan にあたる実装に近い詳細も記述できる記述方式を提案する．

また，近年クラウドソーシングを組み込んだデータベースシステムが提案されている [3] [2] [5]. Deco では，伝統的な問合せ木に，Fetch Rules や Resolution Rules を導入したものを実行プランの議論に利用している．人が行う処理は Fetch Rules に書かれている内容に限定されており，Fetch Rules に書かれている操作は計算機にはできないという仮定が置かれている．本稿で提案する記述方式は，処理とその処理が行う主体がそれぞれ独立していることから，人が行う処理も計算機が行う処理もまとめて 1 つのルールに記述し議論できる．

本稿で提案する実行プランは，人と計算機による処理を共にオープンコールする事を想定している．seti@home [6] では，計算機による処理をオープンコールしているが，オープンコールするものは計算機資源だけであり，アルゴリズムは依頼者が与える．本稿では，人と計算機による処理を共にオープンコールする事を想定している．言い換えれば，計算機だけでなく人もタスクに取り組むことができ，そのタスクを解くためのアルゴリズムもワーカが自由に選定してよい．Generalized Task Market [7] では，計算機もワーカとしてモデル化し，人による処理と計算機による処理を区別しないというアイデアが提案されている．しかし，そのアイデアは翻訳システムの実装として実装されており，汎用的な仕組みは構築されていない．

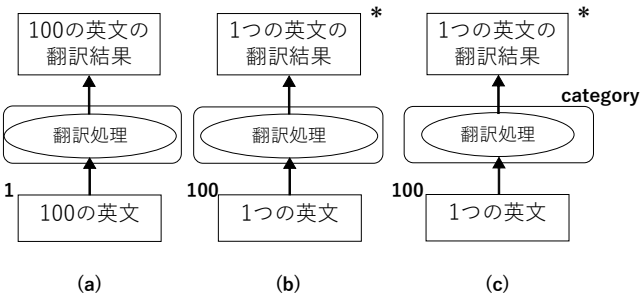

図 2 自然言語の CDFD：具体例 1

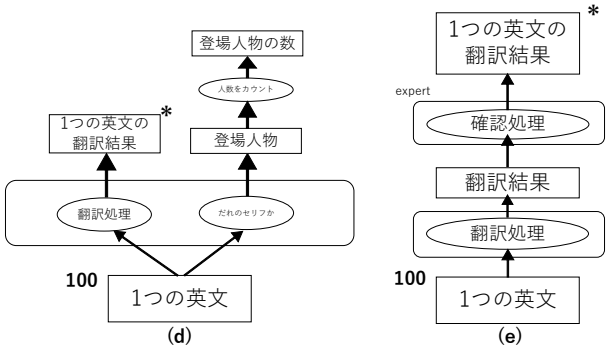

図 3 自然言語の CDFD：具体例 2

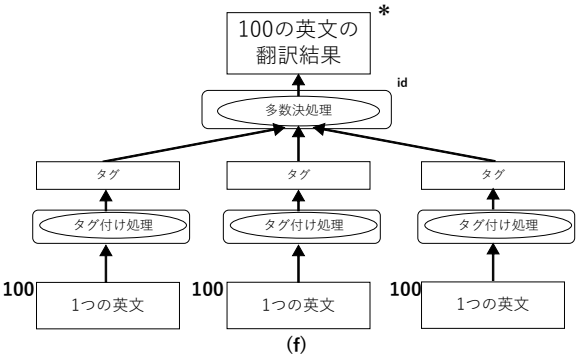

図 4 自然言語の CDFD：具体例 3

4. Crowd Data Flow Diagram

本節では Crowd Data Flow Diagram (以下，CDFD) について説明する．CDFD は人と計算機による処理の実行プランを書くための図であり，2 節で説明した条件をすべて満たす．まず，CDFD の例について説明した後 CDFD の意味を定義する．

4.1 CDFD の例と構成要素

CDFD を利用してタスクを記述した例（図 2〜図 4）をもとに，構成要素の説明を行う．なお，タスクは英語を日本語に翻訳するものを用いる．

4.1.1 ノ ー ド

図 2(a) は「100 の英文を翻訳するタスク」からなるクラウドソーシング設計を表す CDFD である．CDFD はデータ，処理，タスクの 3 つのノードからなる．

- データノード： 図中にて四角で表されるデータノード

は，問題の解そのものや問題の解を生成するのに必要なデータを表す．全てのデータはその構造を表すリレーションスキーマを保持する．図 2(a) では，“100 の英文” が，問題の解を生成するのに必要なデータであり，“100 の英文の翻訳結果” が，問題の解を表すデータである．

- 処理ノード: 図中にて楕円で表される処理ノードは，データをもとにデータ自体を変更したり，新たなデータを生成する．図 2(a) では，“翻訳処理” が，100 の英文を翻訳する作業を表す処理である．
- タスクノード: 図中にて角丸四角で表されるタスクノードは，クラウドソーシングのタスクとして行われる処理ノードをまとめたノードである．一つのタスクで複数の処理を行うことが可能である．図 2(a) のタスクでは，100 文を翻訳する処理を行う必要がある．

4.1.2 データ集合と集約

図 2(b) は「100 の英文を 1 文ずつ翻訳し，100 文まとめて出力するタスク」の設計となる．(a) との違いは 100 の英文をまとめて翻訳していたところを，1 文ずつ翻訳している点と最終的なデータをまとめて出力している点である．データノードの左上にある番号は，データの個数を表す．また，データノードの右上には，データの集約の条件を指定することが出来る．この例では，“1 つの英文” が 100 個存在し，“1 つの英文の翻訳結果” を全て集約したものが，最終的なデータとなる事を指定している．*でなく，属性が書かれた場合には，その属性の異なる値毎に集約が行われる．

4.1.3 タスクにおける作業の集約

図 2(c) は「100 の英文をカテゴリごとに翻訳し，100 文まとめて出力するタスク」の設計となる．(b) との違いは 1 タスクの粒度がカテゴリーごとになっている点である．例えば英文がある物語の台本とし，人物をカテゴリとする．この場合，人物ごとのセリフを英文翻訳するタスクを設計できる．このように，タスクの右上に集約条件を記述した場合には，その条件を満たす処理を集めたタスクを行う事を示す．一方で，処理の右上に直接集約条件を記述した場合には，その条件を満たすデータをまとめて処理することを表す．

4.1.4 複数の処理を含むタスク

図 3(d) は「100 の英文を 1 文ずつ翻訳と，その英文がどの人物のセリフかを入力し，翻訳結果と，人物別のセリフ数を出力するタスク」を表す CDFD である．このように，全く別の処理をまとめて 1 つのタスクとしてもよい．また，一つのデータノードから，複数の処理ノードに矢印があっても良い．

4.1.5 他のタスクの結果に依存したタスク

図 3(e) は「100 の英文の中から 1 文を翻訳，その後翻訳が行われた文が正しいか確認を行い，確認されたもののみを出力するタスク」を表す CDFD である．(d) のようにタスクを 1 つにまとめることも (e) のように 2 段階に分けることも可能である．

4.1.6 タスクをする主体の条件

図 3(e) の確認タスクは，(e) の翻訳処理と異なり，英語翻訳の上級者 (expert) のみが行うタスクとなっている．このように，タスクノードの左上にはタスクを割り当てる主体の条件を

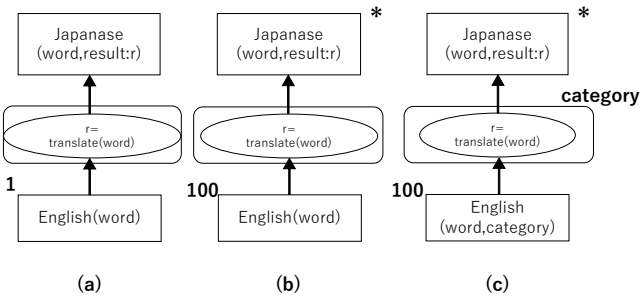

図 5　フォーマルな CDFD : 具体例 1

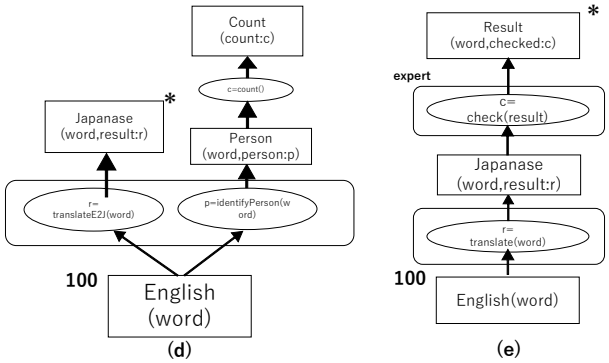

図 6　フォーマルな CDFD : 具体例 2

指定することが可能である．条件の例としては，人間ワーカのみ，アルゴリズムワーカのみ，特定のスキルを持つ人のみ，正答率 80%以上のワーカのみ，ブラックリストに載っていない人のみ，等が挙げられる．

4.1.7 結　　合

図 4 は「100 の英文の中の 1 文に対し，誰のセリフかというタグ付けを 3 人のワーカに行ってもらい，その中で多数決を行って最終的なデータを出力するタスク」を表す CDFD である．複数ノードの矢印が一つのノードに合流 (この場合は，複数の “タグ付け処理” の結果が “多数決処理” に合流) する場合，合流する組合せを指定 (ここでは，“同じ id を持つデータの組み合わせ”) する．特に指定が無い場合には，全ての組合せ (直積) が渡される．

4.2 フォーマルな CDFD

CDFD 自体は自然言語で書いて良いものであるが，我々は「フォーマルな CDFD」を定義する．フォーマルな CDFD では，ノード内に定められた形式の式を書く必要がある．図 5〜7 は，図 2〜4 の CDFD に対応するフォーマルな CDFD である．これを例に，説明を行う．

4.2.1 データノード

データノードには，リレーションスキーマを記述する．ここで，リレーションスキーマとは，リレーション名の後ろに，属性名を並べたものである．図 7 においては，`EnglishTag(id, word)` は，翻訳前の英語タグを格納するリレーションを表す．また，`JapaneseTag(id, result)` は，翻訳後の日本語タグを格納するリレーションを表す．また図 7 において，

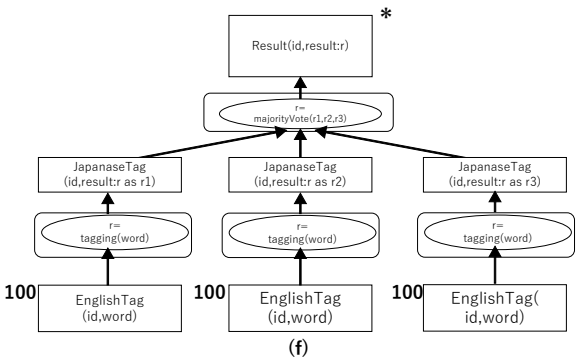

図 7　フォーマルな CDFD：具体例 3

`JapanaseTag(id,result:r as r1 )` とあるが，ここでの “:” は，親ノードの処理結果をそのデータノードの属性に挿入することを表し，“as” は，子ノードで扱う変数として属性の名前を再定義している．

4.2.2　処理ノード

処理ノードには，その処理を表す関数を用いた代入文を記述する．例えば，`result1=tagging(word1)` は，英文 `word1` に，タグ付け `tagging()` を行い，結果を出力先データノードの属性 `result1` に格納することを表す．

4.2.3　結 合 条 件

結合条件は論理式で表す．例えば，`id1==id2 and id1==id3` は，`id1` と `id2` と `id3` が同じ値であることを表す．同じ名前の変数は，同じ値に束縛されるので，もし 3 つのデータノードに全て `EnglishTag(id, word)` と書かれている場合には，条件式を書かなかったとしても，同じタグか否かを結合条件とする．

4.3　フォーマルな CDFD の意味

本節では，フォーマルな CDFD の意味を定義する．まず，フォーマルな CDFD をルール集合に変換し，次に，そのルール集合の意味を定義する．

4.3.1　CDFD ルール言語

CDFD ルール言語は，CyLog に似た言語である．CyLog とは，Datalog [8] 風の構文を持つ，クラウドソーシング用のプログラミング言語である．フォーマルな CDFD の意味は，CDFD ルールの集合として表されるため，ここではまず CDFD ルールについて説明する．

CDFD のルールは，Datalog ルールと似ている．例えば，データベースに格納されている英文を，人で翻訳する処理を考える．次の処理は，図 12(a) を CDFD ルール言語として表した例 (図 8(a)) の一部である．

```
Japanase(word,result:@r) <- English(word)
```

ここで，`English` は英文を表す述語である．これらの述語を用いて記述された `English(word)` などの式を原子式 (Atom) と呼ぶ．原子式は左から評価され，引数に指定された変数が束縛される．ここで@で始まる変数はタスクを通じて人間が値を入力するという指定である．$@r$ の場所には，特別な値 (オープン値) が入っている．特別な原子式として， リレーション `R` の計算が終了している事を表す `R.`，リレーション `R` の全てのタプルを変数 `p` に格納する `R(x,y):p` と行った書き方が許されている．

CDFD ルールでタスクを生成するには，タスクインスタンス毎に 1 タプルを格納する特別なリレーションを利用する．これは，!で開始するリレーションである．次の式は，Japanase でオープン値を含むタプル毎に，タスクを生成する事を表す．[] の中には，タスクの依頼対象を記述する．[*] は誰でも良いことを示す．

```
!Task1(word)[*] <- Japanese(word, result:@w);
```

また，タスク本体を，次のような “タスクテンプレート定義” で定義する．

```
!Task(word) {
  ask @w <- translate(word);
  return Japanese(word, result:@w)
}
```

タスクテンプレート定義の中では，そのタスク中で聞く質問と，結果表すタプルを記述する．複数の異なるタプルを一つのタスク本体で記述することが許される．

4.3.2　ルール集合への変換

図 5〜7 の CDFD をルール集合に変換したものを図 8〜10 に示す．これは，中間データを表す各ノード毎に，次の規則によりルールを導出することにより作成したものである．次のルールでは，矢印でつながるノードの元側を親，先側を子と呼ぶ．また，これらの閉包をそれぞれ祖先，子孫と呼ぶ．親の居ないノードを始端ノード，子の居ないノードを末端ノードと呼ぶ．

以下の説明では，次の記号を用いる．

- CDFD のノードを n と表記する．
- n の祖先ノードのうち，最も n に近いデータノードを $nd_1, nd_2, \ldots, nd_k$ とする．
- nd_i と n に挟まれたタスク外の処理ノードを $np_1, np_2, \ldots, np_k$ とする．
- nd_i と n に挟まれたタスク内の処理ノードを $tp_1, tp_2, \ldots, tp_k$ とする．
- ノード n に書かれている式を $exp(n)$ と表現する．

このとき，始端以外のデータノード n それぞれに対して，次のルールを導出する．

$$exp(n) \leftarrow \quad exp(nd_1), exp(nd_2), \ldots exp(nd_k), \\ [exp(np_1), exp(np_2) \ldots exp(np_m)]$$

ただし，処理ノード np_i がタスクノードに含まれている場合には，$exp(np_i)$ は代入する変数の先頭に@マークを付け，後述するように，タスク本体の定義に含める．このルールでは，そ

(a)
```
Japanese(word, result: @result) ← English(word),
!Task(words)[*] ← English(word), Japanese(word, result: @result)
!Task(word){
 ask @result ← translate(word)
 return Japanese(word, result: @result)
}
```

(b)
```
Japanese(word, result: @result) ← English(word)
!Task(word)[*] ← English(word), Japanese(word, result: @result)
!Task(word){
ask @result ← translate(word)
 return Japanese(words, result: @result)
}
Table(word, results: r) ← Japanese., Japanese(word, result): v, [r ← agg(v)]
```

(c)
```
Japanese(word, result: @result) ← English(words),
!Task(word)[*] ← English(word), Japanese(word, result: @result),
                 distinct English(category)
!Task(word){
 ask @result ← translate(word)
 return Japanese(words, result: @result)
}
Table(word, results: r) ← Japanese., Japanese(word, result): v, [r ← agg(v)]
```

図 8　CDFD の式表現：具体例 1

(d)
```
Japanese(word, result: @result) ← English(word)
Person(word, person: @person) ← English(word)
Count(count: count) ← Person(word, person): m, [count = count(m)]
!Task(word)[*] ← English(word)
!Task(word){
 ask @r ← translateE2J(word)
 ask @p ← identifyPerson(word)
 return Japanese(word, result: @r), Person(word, person: @p)
}
Table(word, result: r) ← Japanese., Japanese(word, result): v, [r ← agg(v)]
```

(e)
```
Japanese(word, result: @r) ← English(word)
Result(word, checked: @c) ← Japanese(word, result: @r )
!Task1(word)[*] ← English(word)
!Task1(word){
 ask @r ← translate(word)
 return Japanese(word, result: @r)
 }
!Task2(result)[expert] ← Japanese(word, result)
!Task2(result){
 ask @c ← check(result)
 return Result(word, check: @c)
 }
Result(word, result: r) ← Result., Result(word, result): v, [r ← agg(v)]
```

図 9　CDFD の式表現：具体例 2

(f)
```
JapaneseTag(id, result: @r as r1) ← EnglishTag(id, word)
JapaneseTag(id, result: @r as r2) ← EnglishTag(id, word)
JapaneseTag(id, result: @r as r3) ← EnglishTag(id, word)
!Task(word) ← EnglishTag(word)
!Task(word){
 ask @r ← tagging(word)
 return JapaneseTag(id, result: @r)
}
Result(id, result) ← JapaneseTag(id, r1), JapaneseTag(id, r2)JapaneseTag(id, r3),
                     [result ← majorityVote(r1, r2, r3)]
Table(id, result: r) ← Result., Result(id, result): v, [r ← agg(v)]
```

図 10　CDFD の式表現：具体例 3

の結果を格納する変数名に@マークを付ける．

例えば，図 6(a) の CDFD からは，次のルールが導出される．

$$Japanese(words, result : @result) \leftarrow \quad English(words)$$

4.3.3　データの集約

データノード nd_i に対して集約が指定されている場合には，次のルールを導出する．

$$S(a, y) \leftarrow \texttt{distinct}\, R(y), R(x, y) : v, [a \leftarrow agg(v)]$$

ここで，$exp(nd_i) = R(x, y)$ とし，集約条件として y が指定されているとする．また，$exp(n) = S(a, y)$ とし，a は集約結果を格納する変数とする．集約条件が*の場合には，`distinct R(y)` は指定されず，次の様になる．

$$S(a, y) \leftarrow R., R(x, y) : v, [a \leftarrow agg(v);]$$

ここで，$R.$ は，R の結果が全てそろうと true を返す述語である．

4.3.4　タ　ス　ク

各タスクノードに対して，次のルールを導出する．ここで，nt_i は，タスクの中に処理ノード，$?exp(nt_i)$ は，タスクの結果が格納されたタプルを表す．

$$!Task_k(id) \leftarrow ?exp(nt_1), ?exp(nt_2), \ldots, ?exp(nt_n);$$

また，タスク本体を定義し，タスク中の処理ノードの全ての質問文とその結果を返すタプルを記述する．

4.3.5　タスクの集約

タスクの集約が指定されているときには，次のルールを導出する．ここで，$R(y)$ は，集約の属性値を提供するデータノードのリレーションであり，y が集約の属性値とする．

$$!Task_k(id, v) \leftarrow Rel(exp(nt_1))., \ldots Rel(exp(nt_2))., \texttt{distinct}R(y)$$

5.　伝統的な実行プランとの性質の違い

提案する記述方式による実行プランを利用した各種コストの見積りや最適化を行う際には，データベースシステムを対象とした伝統的な実行プランを扱う場合と比べて，次の 2 つの大きな違いがある．

5.1　ワーカに関する要素

データベースシステムにて行われる処理は，正しいことが検証されたアルゴリズムによって全て行われることが一般的である．一方で，クラウドソーシングの処理においては，スパムが混じる可能性もあり，スパムでなくても一般に処理結果が正しいとは限らない．また，実際に処理を開始したときに，どれぐらいのワーカが参加可能かは一般には不明である．これらの点も，適した実行プランを最初から選択することを困難にする一因となる．

5.2　データに関する統計情報の有無

一般にデータベースシステムは既にデータが存在し，そこから得られた統計情報を元に実行プランを評価できる．しかし，クラウドソーシングを処理に含む場合，データが処理の初期段階では存在しない場合がある．従って，処理開始時には統計情報が得られず，適した実行プランを選択することが困難である．

6.　実行プランの見積り

本節では，前節で説明した，データに関する統計情報や，ワーカに関する情報は既知として，CDFD で記述された実行プランから，タスク数，費用，実行時間を見積もる方法を検討する．データに関する統計情報などの議論は次節で行う．

ある CDFD が与えられたとき，ここでは次の表記を用いる．

- $T = \{t_1, t_2, \ldots, t_n\}$ をそこに含まれるタスクノードの集合とする．例えば，図 11 の場合，$T = \{t_1, t_2, t_3\}$ である．
- 各タスクノード t_i に入次するデータノードの集合を，$In(t_i) = \{d_{i,1}, d_{i,2}, \ldots, d_{i,n}\}$ と表記する．図 11 の場合，

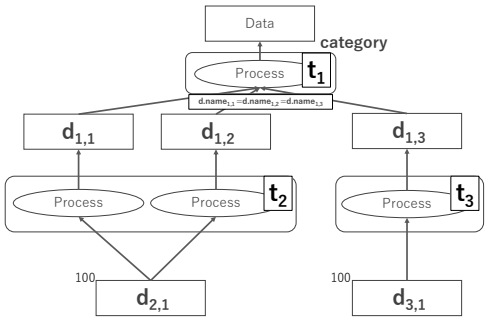

図 11　CDFD に関する見積りで利用する例

$In(t_1) = \{d_{1,1}, d_{1,2}, d_{1,3}\}$ である．

- データノード $d_{i,j}$ が保持するタプル数を $N(d_{i,j})$ と表記する．図 11 の場合，$N(d_{2,1}) = 100$ である．
- タスクノード t_i に入次するデータノードが表すリレーションのうち，結合結果のタプルの数を $\#Join_{t_i}(In(t_i))$ とする．図 11 の場合，t_1 にて，$In(t_1)$ のデータノードのリレーション “name” がそれぞれ一致するタプルの数が $\#Join_{t_i}(In(t_i))$ である．
- タスク t_i の集約条件として属性名が記述されている場合，その属性値を用いて集約を行う．その結果のタスク数 $\#Agg_{attr}(t_i)$ とする．図 11 の場合，t_1 のタスクは，入次されたデータノードのリレーションの中で，属性 “category” のデータに集約を行い，そのデータごとにタスクが行われる．その時のタスク数を $\#Agg_{category}(t_1)$ と表記する．
- タスクノード t_i が表現するタスクを一つ実行するために必要なコスト (費用) を $Cost(t_i)$ と表記する．

6.1　タスク数の見積り

タスクノード t_i が表すタスクのタスク数の見積り $Count(t_i)$ は，タスクノードに集約条件 $attr$ が記述されているかどうかに依存する．

- タスクノードに集約条件 $attr$ が記述されている場合，$Count(t_i) = Agg_{attr}(t_i)$ となる．しかし，$\#Agg_{category}(t_1)$ の数は，データに依存する．データがデータベースにあらかじめ格納されている場合は求めることが可能であるが，クラウドソーシング等で入手する場合は，正確に求めることはできない．

したがってそのような場合には，タスク数は $attr$ を含むリレーションのタプル数と同じであると見積もる．

- タスクノードに集約条件が記述されていない場合，$Count(t_i) = \#Join(t_i)$ となる．リレーションの統計値を用いて見積もることが可能な場合にはそれを利用する [9] が，不可能な場合には，最悪，データの数はそれぞれのタプルの直積となる．したがって，従って次の式となる．

$$Count(t_i) = \prod_{d_{i,j} \in In(t_i)} N(d_{i,j})$$

6.2　費用の見積り

タスク t_i で必要とされる費用 $Totalcost(t_i)$ は次の式で表現できる．

$$Totalcost(t_i) = Count(t_i) * Cost(t_i)$$

6.3　時間の見積り

実行プランの時間を見積もるうえで重要なのはタスクの並列性である．タスクが並列に処理できる場合，かかる時間は並列にこなされるタスクの中で最も処理時間が長いものになる．そこで，次の仮定を置く．

- あるタスクノード t_i に最も近い子孫となるタスクノードの集合を $Pred(t_i)$ とする．図 11 の場合，$Pred(t_1) = t_2, t_3$ である．
- タスク t_i の処理時間は $Time(t_i)$ とする．
- 末端のタスクノードから t_i までを処理するために必要な処理時間を $Totaltime(t_i)$ と表現する．t_i が末端の時は，$Totaltime(t_i) = Time(t_i)$ である．
- $Pred(t_i)$ に含まれるタスクノードのうち，最も長い Totaltime を持つ t_j を $Max(Pred(t_i))$ と表現する．$Pred(t_i) = \phi$ の場合は $T(Pred(t_i)) = nil$ を返す． ここで，$Time(nil) = 0$ と定義する．

以上のとき，末端のタスクノードから t_i までを処理するために必要な処理時間は，次のように再帰的に表すことができる．実行プランの合計処理時間を求めることができる．

$$Totaltime(t_i) = Time(t_i) + Max(Pred(t_i))$$

7.　シナリオ例と課題

本節では，翻訳を行う実行プランを題材に，提案した記述方式の実行プランを活用する際の課題について議論する．

7.1　シナリオ例

本シナリオでは，10 万の英文のうち，観光に関する文を日本語に翻訳する問題を扱う．ワーカは，人であっても，アルゴリズムであってもよいとする．

7.2　実行プランの例

このシナリオを実現するための実行プランの例をいくつか示す．ここでは異なる二つの状況で有効と考えられる実行プランを示す．

第一に，人による作業は信頼できるという状況で有効と考えられるプランは次が考えられる．

- 図 12(プラン 1) は，まず，すべての文にタグをつけ，その後，観光に関する文だけを翻訳する．これらの作業はともに人手で行う．
- 図 12(プラン 2) は，プラン 1 と同じであるが，タグ付けはアルゴリズムで行う．これは，タグ付けを行うアルゴリズムワーカの品質が良い場合に適していると考えられる．
- 図 12(プラン 3) は，タグ付けの結果を人手で確認する実行プランである．タグ付けを行うアルゴリズムワーカの品質があまりよくない場合に適していると考えられる．

第二に，人による作業が必ずしも信頼できないという状況でのプランは次のようなものが考えられる．

- 図 13(プラン 4) は，タグ付けを複数人のワーカの多数決によって決定した後，翻訳処理を行うプランである．このプラ

表 1　人間ワーカとアルゴリズムワーカの違い

	人間ワーカ	アルゴリズムワーカ
タスクの処理速度	低速	高速
本人の自信度	信頼できない	信頼できる

ンは，ワーカの品質がそれほど高くない状況でも有効と考えられる．理由は，タグ付けは不正解の数が多く，不正解のもの同士で一致することがすくないからである．

- 図 14(プラン 5) は，図 13(プラン 4) と同じであるが，多数決の一員をアルゴリズムで行う．

7.3　問題 1: アルゴリズムワーカの扱い

まず，オープンコールによって参加するアルゴリズムワーカを評価する仕組みが必要である．アルゴリズムワーカと人間ワーカでは，ワーカとしての性質が異なる (表 1)．異なる性質の一つとして，回答者であるワーカの自信の信頼度がある．人間ワーカの回答とその自信度には相関が無いことが知られている [10]．一方で，アルゴリズムワーカは定められた処理から答えを導き出すため，その自信度には信頼性があるといえる．タスクの種類にもよるが，人間による処理を前提としたクラウドソーシングタスクの多くでは，一般に，アルゴリズムワーカのタスクの正解率は低くなりがちである．したがって，ゴールドスタンダードデータ（既知の解答）を用いたテストなどの成績は必ずしも高くなく，スパムではないにもかかわらず，人間のワーカでなければ採用されない可能性がある．

しかし，必ずしも平均して正解率が低いわけではなく，特定のパターンの場合には得意であるという状況も存在する．例えば，計算機を使って多くの漢字と少数の英数字の混じる画像を 1 文字ずつ文字におこすタスクを考える．平均のタスク正解率が低かったとしても，英数字に限って言えば，認識精度が高いといったケースが考えられる．

また，タスクの処理コストが人間に比べて圧倒的に安いため，少しでも活用できるなら活用してもトータルとしては安上がりになる可能性もある．

したがって，アルゴリズムワーカを活用するには，次を気をつける事が必要と考えられる．(1)　正解率が低くてもスパムとして扱うのではなく，活用出来る機会を検討する．(2) 得意な問題では正解率が高い可能性が高いという性質を利用する．

そこで，上記のような点を考慮した，アルゴリズムワーカの採用判定方法が考えられる．ポイントは次の通りである．(1) 得意なものをアルゴリズムワーカ自身に申告させる．(2) 単純に正解率の高低を問題にするのではなく，選択肢からランダムに解答を行うワーカ（以下，ランダムワーカ）と比べて有意に正解率が高ければワーカとして採用する．

具体的には，次の手順で，判定を行う．

（1） 計算機に既に正解が既に得られているタスクを渡し，その回答を得る．回答のうち，正解率の計算対象として欲しい回答にはフラグを付ける．

（2） フラグがついている回答と，正解を比較し，正解率を求める．

（3） ランダムワーカよりも統計的に有意に高いかどうかを判定し，高ければ採用する．

7.4　問題 2: データ統計情報や単価情報の取得

クラウドソーシングによって入力が行われるデータに関しては，事前に統計情報を得ることは困難である．例えば，このシナリオ例では，10 万文のそれぞれが観光に関する文か否かのタグは既知でないため，どれだけの文を日本語に翻訳する必要があるかわからない．したがって，ランダムサンプリングなどでタスクの一部をワーカに回答してもらい，その結果から全体のデータの推定を行う方法が考えられる．その推定作業を実行プランの一部として含めることが必要と考えられる．

各タスクを行う際の単価についても取得する必要があるが，人間とアルゴリズムでは，コストモデルが異なることにも注意が必要である．人間は，タスク処理数に比例して支払いを行うことが一般的と思われるが，アルゴリズムは一定の開発コストがかかり，実行時のタスク数だけに比例した支払いは必ずしも適切でないかもしれない．これらに関しては，ワーカ側が提案するか，こちら側が提案して納得のいくワーカのみが参加するなど，何らかの調整の仕組みが必要と考えられる．

7.5　問題 3: 実行プランの選択

実行プランを評価するための各種情報の入手や推定が終了したら，次のステップとして，適切な実行プランを選択することになる．本来は，同じ結果をもたらす実行プランを列挙する必要があるが，本稿では，7.1 節で説明した実行プランを対象とする．ここでは下記を仮定する．まず，人のタスク処理の費用と時間を表 2 と仮定する．次に，アルゴリズムワーカはタスクとごとではなく，固定で 1M(M は 10^6 を意味する) の費用を必要とし，タスクの処理時間を 1 とする．また，観光に関する文が 10 万の場合と 10 の場合のそれぞれを検討することとし，実行プランの 3 つの評価軸で比較する．

この仮定の下で，プラン 1〜3 を比較した結果を表 3 に示し，プラン 4〜5 を比較した結果を表 4 に示す．ここで，K は 10^3 を意味し，時間の単位は秒，費用の単位は円とする．

この表からわかるように，費用，時間，人の行うタスク数のすべてにおいて優れたプランは必ずしも存在しない場合がある．例えば，タスクの数が 10 の場合，プラン 1 が最もタスクの費用が少なく，プラン 2 が最も処理時間が短く人間が行うタスク数が少ない．したがって，一概にどちらが良いとは判断できない．

この問題の解決策として考えられるのは，依頼者が何を希望するかを事前に取得する方法である．例えば，依頼者が時間の速さを最重視する場合は，タスク 10 のプランの中で最適なプランはプラン 2 となる．反対に，依頼者が費用の安さを最重視する場合は，最適なプランはプラン 1 となる．

以上から，複数の視点を元にそのシナリオで求められる実行プランを選択することが要求される．また，一般のデータベース問合せと比較して長時間の処理時間が必要になることから，動的な最適化の必要性がより高いと考えられる．

8.　まとめと今後の課題

本稿では，人と計算機による処理を包括的に記述可能な「実行プラン」記述方式を提案した．また，人と計算機の処理をと

表 2　タスクの費用・時間

	費用	時間
翻訳タスク	100	100
タグ付けタスク	60	60
確認タスク	30	30

表 3　実行プラン 1 ～ 3 の評価

	英文の数	タスクの費用	時間	人間が行うタスク数
プラン 1	100K	16M	16M	200K
プラン 2	100K	11M	10.1M	100K
プラン 3	100K	14M	13.1M	200K
プラン 1	10	1.6K	1.6K	20
プラン 2	10	1001K	1.01K	10
プラン 3	10	1001.3K	1.31K	20

表 4　実行プラン 4 ～ 5 の評価

	英文の数	タスクの費用	時間	人間が行うタスク数
プラン 4	100K	23M	16M	300K
プラン 5	100K	28M	16M	400K
プラン 4	10	2.3K	1001.6K	30
プラン 5	10	2.8K	1.6K	40

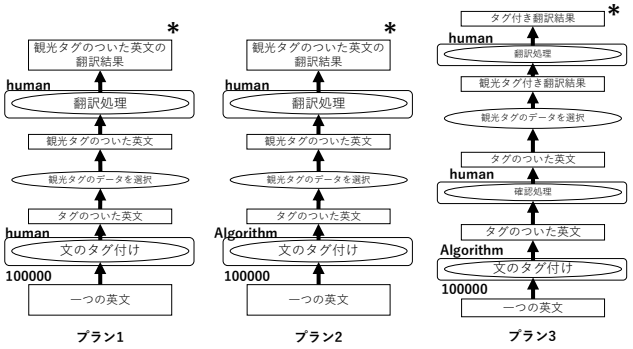

図 12　実行プラン例：1

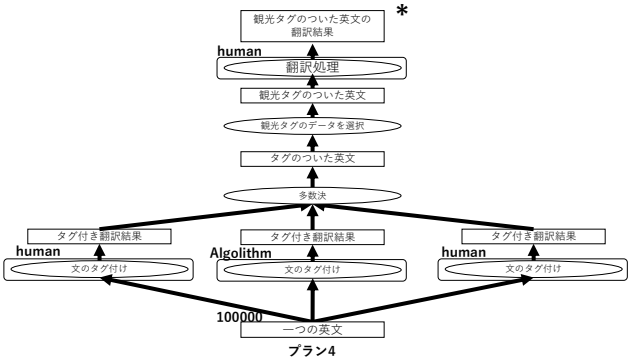

図 13　実行プラン例：2

もにオープンコールして行う場合を想定し，提案した実行プランを用いた処理や最適化について議論を行った．今後は，各種統計情報の推計手法や，同等の実行プランを数え上げる体系的な手法の開発が必要である．

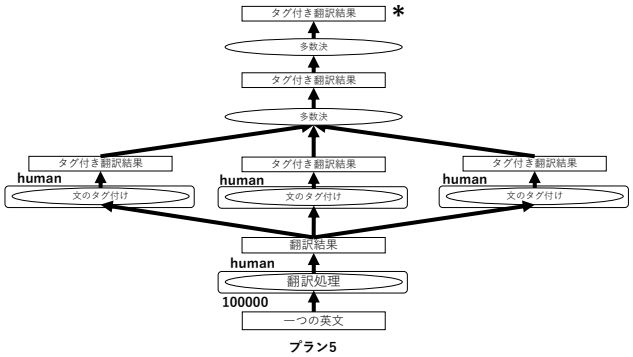

図 14　実行プラン例：3

謝　　辞

本研究の一部は，JST CREST および JSPS 科研費 (#25240012) の支援による．

文　　献

[1] Jeff Howe. The rise of crowdsourcing. *Wired Magazine*, Vol. 14, No. 6, 06 2006.

[2] Aditya Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: Declarative crowdsourcing. Technical report, Stanford University, 2012.

[3] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: Answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pp. 61–72, New York, NY, USA, 2011. ACM.

[4] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice-Hall, 2002.

[5] Atsuyuki Morishima, Norihide Shinagawa, Tomomi Mitsuishi, Hideto Aoki, and Shun Fukusumi. Cylog/crowd4u: A declarative platform for complex data-centric crowdsourcing. *Proc. VLDB Endow.*, Vol. 5, No. 12, pp. 1918–1921, August 2012.

[6] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: An experiment in public-resource computing. *Commun. ACM*, Vol. 45, No. 11, pp. 56–61, November 2002.

[7] Dafna Shahaf and Eric Horvitz. Generalized task markets for human and machine computation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pp. 986–993. AAAI Press, 2010.

[8] Stefan Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE TRANSACTIONS KNOWLEDGE AND DATA ENGINEERING*, Vol. 1, No. 1, pp. 146–166, 1989.

[9] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. *SIGMOD Rec.*, Vol. 14, No. 2, pp. 256–276, June 1984.

[10] Satoshi Oyama, Yukino Baba, Yuko Sakurai, and Hisashi Kashima. Accurate integration of crowdsourced labels using workers' self-reported confidence scores. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pp. 2554–2560. AAAI Press, 2013.