L-014

ボランティアコンピューティングにおけるノードの動的クラスタリング

Dynamic Node Clustering for Volunteer Computing

福士 将 Masaru Fukushi †
菅原 雅也 Masaya Sugawara ‡
堀口 進 Susumu Horiguchi †

1 まえがき

近年，インターネット上の遊休計算機資源を活用して，大規模な並列分散処理環境を構築するボランティアコンピューティング（Volunteer Computing: VC）が研究され，実際に運用されている．VC では，参加者が計算プロジェクトに自主的に参加し，自身のパーソナルコンピュータの遊休計算資源（アイドル CPU サイクルなど）を無償で提供する．大型のスーパコンピュータなどと比較すると，VC における個々の計算資源の性能は格段に低い．しかし，SETI@home[1] や Folding@home[2] の実用例に見られるように，インターネット上の膨大な数の計算資源を集めることで，スーパコンピュータを凌ぐほどの計算力をほぼ無償で入手できることが VC の大きな利点である．

これまで VC で扱われてきた計算は，高い並列性があり，ノード間通信をほとんど必要としない“単純並列処理”が可能な大規模並列計算である（例えば，[1]–[5]）．これは，現在の VC 環境で簡単に実行でき，多数のノードで独立に計算させることによる台数効果が得られやすいこと，などが理由として挙げられる．しかし，科学技術計算など全ての大規模並列計算が単純並列処理が可能なわけではなく，VC で扱える計算問題が限定されることは望ましいことではない．

そこで本稿では，通信を含む大規模並列計算を効率よく処理させるための VC のシステム基盤として，ノードを階層的にクラスタリングして管理する手法を提案する．具体的には，下記 2 点を提案する．

1. ノード間で通信を含む場合の VC の並列計算モデル
2. ノードの動的な参加，離脱に対応するための動的クラスタ再構成法

1 については，従来のマスタ・ワーカモデルを拡張し，同期的な通信を含むタスクをノード群（クラスタ）に割り当てる計算モデルを提案する．また，通信オーバヘッドを考慮するために，ノード間のネットワーク距離に応じた簡易通信コストのモデル化を行う．2 については，ノードの動的な参加，離脱が発生する VC 環境下で，ネットワーク距離の近いノードを階層的にクラスタリングし，サイズと個数が任意のクラスタを動的に生成する手法を提案する．インターネット上のノードをクラスタリングする手法は多く提案されているが [6], [8]–[11]，本稿では，Banerjee ら [6] の階層型クラスタ構造を基に，参加ノードが自律分散的にクラスタの再構成を行うために必要となる処理手続きを定義する．シミュレーションによる性能評価により，通信を含む大規模な並列計算問題を処理する VC 環境において，本提案手法により処理効率が向上可能であることを明らかにする．

†東北大学 大学院情報科学研究科, Graduate School of Information Sciences, Tohoku University
‡日本アイ・ビー・エム株式会社, ITS ソリューション・センター, ITS Solution Center, IBM Japan,Ltd.

2 VC の計算モデル

2.1 既存の並列計算モデル

現在運用されているほとんどの VC では，計算モデルとして図 1 に示すマスタ・ワーカモデルを採用している．以下にその概要を示す．

- 処理対象である計算プロジェクトは複数の独立なタスクから成り，各タスクは複数の独立なサブタスクから成る．
- VC システムはマスタと複数台のワーカから成る．
- マスタは，あるタスクの各サブタスクをワーカに対して割り当て，処理を依頼する．
- 各ワーカは，割り当てられたサブタスクを計算し，計算終了後に結果をマスタに返却する．

全てのサブタスクの結果が返却された段階でそのタスクの計算が終了し，同様に全てのタスクが終了すると計算プロジェクトの終了となる．

2.2 通信を含む並列計算モデル

VC の適用範囲を拡大するために，ノード（ワーカ）間の通信を考慮した並列計算モデルを新たに定義する．提案する計算モデルの概要を図 2 および以下に示す．

- 計算プロジェクトは複数の独立なタスクから成り，各タスクは互いに通信を行う複数のサブタスクから成る．
- VC システムはマスタと複数台のワーカから成り，各ワーカはタスクの処理単位として D 個のクラスタに分類される．
- マスタは，あるタスクのサブタスクを，そのタスクを担当するクラスタ内のワーカに割り当てる．
- ワーカは，他のサブタスクを処理するワーカと通信を行いながら，割り当てられたサブタスクを計算し，結果をマスタに返却する．

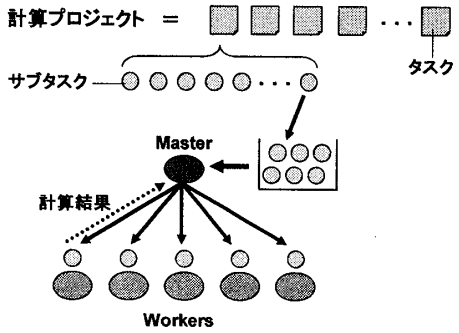

図 1　既存のマスタ・ワーカ型計算モデル

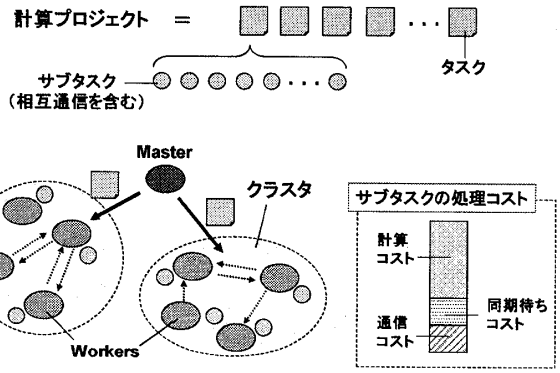

図 2　タスク内に通信を含むマスタ・ワーカ型計算モデル

本計算モデルでは，サブタスク間に通信を含むタスクが D 個並列に計算される．この点が既存のマスタ・ワーカモデルとは大きく異なる点である．

次にサブタスクの処理コストをモデル化する．図 2 に示されるように，サブタスクの処理コストとして，計算処理そのものに要するコスト（計算コスト）に加え，通信が開始可能になるまでの待ち時間コスト（通信待ちコスト）と通信処理に要するコスト（通信コスト）を定義する．一般的な並列計算では，計算の途中で複数回の通信が発生する．通信相手と通信回数は対象とするアプリケーションに大きく依存する．本稿ではモデルの簡略化のために，複数回の通信を 1 つにまとめた形で，通信待ちコスト，通信コストをそれぞれ 1 つのコストとして取り扱う．

各ノードは通信待ちの間に新たなサブタスクをマスタから受け取り，その計算処理を行うことができるものとする．このとき，通信待ちのサブタスクの通信が開始可能になったら，計算中のサブタスクの計算処理を一旦中断し，通信処理を再開できるものとする．ワーカが VC 環境から離脱した場合は，計算および通信の進行状況に関わらず，所持していたサブタスクの計算結果は全て破棄されるものとする．本計算モデルでは，サブタスクに定義された 3 つのコスト分の処理を完了した時点でそのサブタスクの処理が終了し，全サブタスクが終了した時点でそのタスクが終了する．

3　動的クラスタ再構成法

3.1　ノードのクラスタリング

通信を含むタスクを並列に計算するためには，タスクの処理単位および通信範囲を規定するために，ノードをクラスタリングする必要がある．この際，通信オーバヘッドを削減するために，なるべく近いノード同士をクラスタリングするほうが望ましい．さらに，ノードの参加・離脱やタスクの粒度の変化に対応するために，任意数のノードから成る任意数のクラスタを生成する必要がある．

インターネット上のノードをクラスタリングする手法は，これまでに多く提案されている．従来のクラスタリング手法には，大きく分けて，ある特定の情報源やインフラに依存したクラスタリング手法 [9, 10]，ある特定のノードをランドマーク（目印）としてクラスタリングを行う手法 [8]，P2P 分散型のクラスタリング手法 [6, 11] などが存在する．また，これらの手法は，クラスタが階層型か非階層型かによっても分類することができる．

一般的に，多数のノードからなる大規模な分散処理環境では，スケーラビリティの観点から階層的なクラスタリングが適している．Banerjee ら [6, 7] や上田ら [11] の手法では，通信用の物理ネットワークの上にノード管理用の論理的な階層型クラスタ構造を構築し，ノードの新規参加や離脱などの状況変化に応じて，これを動的に更新する．しかしながら，ネットワーク距離に応じたクラスタリングは可能であるものの，任意数のクラスタの生成やクラスタ内ノード数の調整機能がないため，そのままの形で提案する計算モデルに適用することは困難である．

本稿では，Banerjee ら [6] の提案した階層型クラスタ構造を基にして，クラスタへの新規参加，サブクラスタの分割，リーダ選出などのアルゴリズムを明確に体系化する．また，これに加えて，クラスタ間で所属ノードの移動を行うアルゴリズムを定義することで，クラスタ間で処理性能の調整を可能にする動的クラスタ再構成法を提案する．

3.2　階層型クラスタの基本構造

図 3 に，ノード管理のために本稿で採用する階層型クラスタ構造 [6] を示す．図 3 に示されるように，全てのノードはネットワーク上での距離（ホップ数など）に応じて階層的にクラスタリングされる．各階層ではサブクラスタが形成される．サブクラスタにはリーダノードが存在し，各リーダは最上位層を除く全階層のサブクラスタにおいて，1 階層上のサブクラスタのメンバになっている．図 3 に示されるように，VC に参加する全ノードは最下位層のいずれかのサブクラスタのメンバであり，リーダノードのみが $t \geq 2$ の上位階層のメンバになる．

サブクラスタ内のノード（ネイバーノードと呼ぶ）は Heartbeat 信号と呼ばれる制御信号を互いに送信し合い，距離情報などを交換する．各ノードはネイバーノードの情報をクラスタ表として保持し，随時更新する．Heartbeat

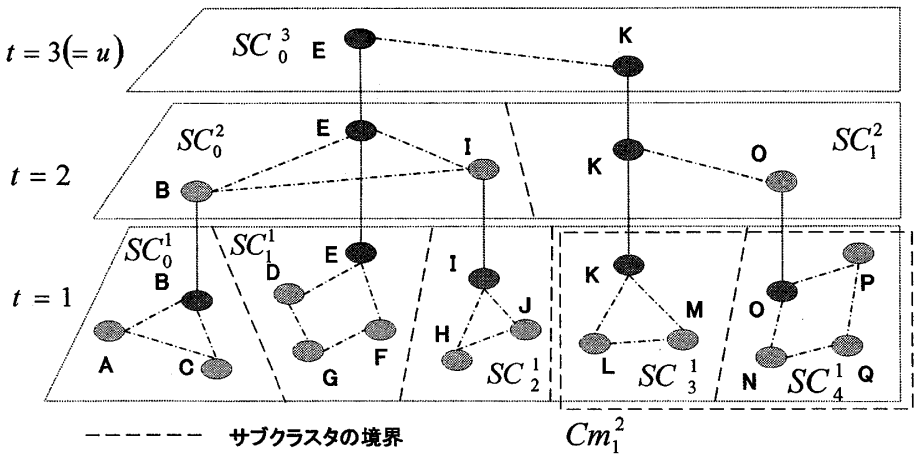

図 3　階層型クラスタの基本構造

信号の応答が一定時間返ってこない場合には，そのノードが離脱したとみなし，該当するクラスタ表から離脱したノードの情報を削除する．

以降の説明の便宜上，以下の記号を定義する．

- SC_i^t ：クラスタ C_i における t 階層目のサブクラスタ
- L_i^t　：SC_i^t のリーダノード
- Cm_i^t：SC_i^t より下層の全てのノードの集合（コミュニティと呼ぶ）

ここで，最下位層を第 1 層，最上位層を第 u 層とし，同一階層のコミュニティを区別するために便宜的な順序 (i) を与える．

3.3　クラスタ再構成アルゴリズム

VC の参加ノードが自律分散的にクラスタの再構成を行うために必要となる各種アルゴリズムを定義する．

3.3.1　新規ノード参加アルゴリズム（Join）

VC に新規に参加するノード n_j は，あるクラスタの最下位層サブクラスタに所属する．n_j は，図 4 に示す Join アルゴリズムにより，クラスタの最上位層から順に，距離的に近いノード n_r を探索し，各階層で再帰的に探索を行うことで，所属先のサブクタスタを選択する．近いノード同士が同一のクラスタに所属することで，以降に新規に参加するノードも同様に近いサブクラスタに所属することができる．

3.3.2　サブクラスタ分割アルゴリズム（Split）

t 階層目のサブクラスタ SC_i^t は，ネイバーノード数が k を超えた場合に，図 5 に示す K-means 法を模した Split アルゴリズムにより，SC_i^t を二つのサブクラスタ SC_a^t と SC_b^t に分割する．分割の際に SC_a^t, SC_b^t のリーダとして新たに選出されたノード n_a と n_b は，第 $t+1$ 層のサブクラスタのメンバとなる．同時に分割前のリーダ L_i^t は第 $t+1$ 層のメンバリストから削除される．また分割がクラスタの最上位層で行われると，クラスタの階層が 1 つだけ増え，n_a, n_b は新しい最上位層サブクラスタのメンバとなる．このため，ノード数の増大に応じて分割が発生することにより，クラスタの階層が適宜増加する．

Join（n_j）

1. $t = u, n_r = L_i^u$ とする．
2. 新規参加ノード n_j は，n_r から SC_i^t のメンバリストを取得する．
3. n_j は取得したメンバリストに記されたノード（一つ下の階層のリーダ L_i^{t-1} に相当する）と自ノードとの距離測定を行い，最も近いノードを新たな n_r とする．
4. $t = 2$ の場合：新規参加ノード n_j は n_r と同じ最下位層サブクラスタに所属する．
   $t \neq 2$ の場合：$t = t - 1$ として手順 2 へ戻る．

図 4　Join アルゴリズム

Split（SC_i^t）

1. L_i^t が任意の 2 ノード n_a と n_b を分割後のサブクラスタ SC_a^t と SC_b^t のリーダ候補として選出する
2. L_i^t は SC_i^t 内のネイバーノードに対し，選出した n_a と n_b を通知する．
3. SC_i^t の各ノードは，n_a, n_b との距離を測定し，それぞれ SC_a^t, SC_b^t の近い方へ所属する．
4. 手順 3 で構成した SC_a^t, SC_b^t のそれぞれにおいて，他ノードとの距離の最大値が最小のノードを選出し，この 2 つのノードを新たに n_a, n_b とする．
5. 手順 2〜手順 4 を一定回数繰り返すことで SC_i^t を 2 つのサブクラスタ SC_a^t, SC_b^t に分割する．

図 5　Split アルゴリズム

3.3.3　サブクラスタ統合アルゴリズム（Merge）

Split アルゴリズムとは逆に，t 階層目のサブクラスタ SC_i^t 内のノード数が一定値 l を下回った場合に，図 6 に示す Merge アルゴリズムにより，サブクラスタを統合さ

Merge（SC_i^t）

1. SC_i^t のリーダ L_i^t はネイバーノードに対し，自身の所属する第 $t+1$ 層のサブクラスタ SC^{t+1} のメンバリストを通知する．
2. 通知を受けた各ノードは取得したメンバリストの中から最も近いノードを探索し，そのノードが所属する第 t 層サブクラスタ SC_j^t（$j \neq i$）のメンバとして所属先を変更する．
3. L_i^t も手順 2 と同様にして，SC^{t+1} のメンバの中から最も近いノードを探索し，第 t 層のメンバとして所属先を変更する．またこの際に L_i^t は SC^{t+1} のメンバから削除される．

図 6　Meargeアルゴリズム

Leader-election（SC_i^t）

1. L_i^t の離脱を感知した段階で，ノードは自身のクラスタ表を参照し，SC_i^t 内の他ノードとの最大距離値が最も小さいノードを新規リーダ候補とみなして，このノードに対してクエリする．
2. 他ノードからのクエリを一定数受けたノードは，新しいリーダノード L_n^t として他のノードへ通知を行う．

図 7　Leader-electionアルゴリズム

せる．Mergeにより，SC_i^t の所属メンバは，L_i^t が所属する第 $t+1$ 層のサブクラスタを経由して，別な t 階層目のサブクラスタへ所属し直す．また，本稿では，最上位層サブクラスタ SC^u 内の所属ノード数が一台だけとなってしまった場合に，SC^u を削除しクラスタの階層を 1 階層分減らすこととした．$u-1$ 階層においてMergeアルゴリズムが発生することにより，所属ノード数の減少に応じてクラスタの階層数を減らすことが可能となる．

3.3.4　新規リーダ選出アルゴリズム（Leader-election）

離脱や故障などにより，サブクラスタ SC_i^t のリーダノード L_i^t が喪失した場合，図 7 に示す Leader-election アルゴリズムにより SC_i^t 内に残された他のノードが自律的に新規リーダノード L_n^t を選出する．なお本稿では，L_n^t の選出に必要なクエリ数を，ネイバーノード数の半数とした．

3.3.5　クラスタ間でのノード移動アルゴリズム（Transfer）

ある特定のクラスタの所属ノード台数が大きく減少すると，このクラスタに与えられたタスクの処理に大きな遅れが生じ，結果的に VC 全体で著しく処理性能が低下する可能性がある．図 8 に示す Transfer アルゴリズムにより，クラスタ C_i からクラスタ C_j（$j \neq i$）へ s 台程度のノードを移動させる．

Transfer（C_i, C_j, s）

1. C_j の最上位層サブクラスタ SC_j^u のリーダ L_j^u は C_i に対し Join アルゴリズムを実行し，最も距離の近い最下位層サブクラスタ SC_i^1 を探索する（ここでは探索のみを行い所属動作は行わない）．
2. 手順 1 で到達した SC_i^1 から上位層に向かって，所属ノード数が s 以上のコミュニティ Cm_i^h を順次探索する．
3. SC_i^h の所属ノード L^{h-1} が C_j に対してそれぞれ Join アルゴリズムを実行する．
4. Cm_i^h の他のノードは，第 $h-1$ 層のリーダが手順 3 で到達した最下位層のサブクラスタへ移動する．

図 8　Transferアルゴリズム

3.4　動的クラスタ再構成法

前節で述べたアルゴリズムを用いて，VC 環境上の性能管理手法として，動的クラスタ再構成法を提案する．提案する手法は，ワーカの参加・離脱，タスクの粒度の変化に対応するために，クラスタ間でノードを移動し，動的に再構成を行う．ここで，全参加ノード数を n，クラスタ数を D として手法の説明をする．簡略化のために，各タスクを構成するサブタスク数は一律に S とし，1 個のサブタスクは R 台のワーカで冗長に計算される冗長計算を仮定する．

1. 階層型クラスタ管理構造への登録
   計算プロジェクトに参加している全ノードを図 3 で説明した階層型クラスタ構造に登録する．
2. クラスタ数 D の決定
   次式により，クラスタ数（同時に処理可能なタスク数）を決定する．

$$D = \left\lfloor \frac{n}{S \times R} \right\rfloor$$

3. クラスタ再構成
   定期的に，最も所属ノード数の多いクラスタ C_i から最も所属ノード数の少ないクラスタ C_j に対して Transfer アルゴリズムを実行する．Transfer アルゴリズムの引数である移動台数 s は，以下の式によって決定する．

$$s = \min\{\frac{1}{2}(x - S \cdot R), 2(S \cdot R - y)\}$$

   ただし，x, y はそれぞれクラスタ C_i, C_j の所属ノード数である．

3.5　新規参加ノードの所属クラスタの決定

VC に新規に参加するノード n_j は，図 4 で示した Join アルゴリズムにより，あるクラスタ内において距離的に近いサブクラスタに所属するが，まず最初に，所属するクラスタを決定する必要がある．ここで，各クラスタに Join アルゴリズムを適用し，最下位層のリーダとの距離

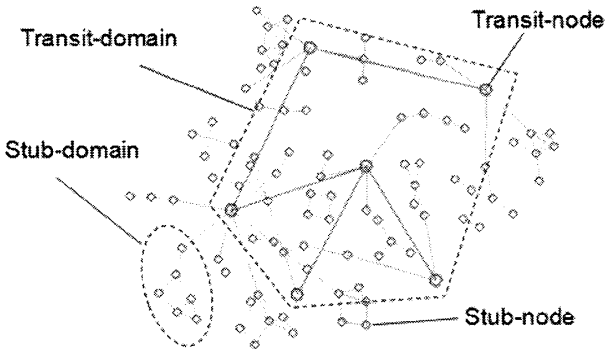

図 9　Transit-Stub 型ネットワークトポロジ

が最も近いクラスタを選択する方法が考えられる．しかし，クラスタ内の全ノードとの距離を考慮しているわけではないため，クラスタの物理的なネットワークトポロジによっては，最下位層のリーダとの距離が最短であっても，距離的に遠いノードが存在する可能性もある．本稿では，n_j との距離が最も近い最下位層のリーダ n_r と最も遠い最下位層のリーダ n_f の 2 つのノードを考慮することで，この問題に対処する．n_j が所属するクラスタを決定する際に，Join アルゴリズムで n_r を，Join アルゴリズムと同様のアルゴリズムで n_f を探索し，次式で定義される近接度が最も小さいクラスタに所属する方法をとる．

$$dis(n_j, n_r) + \alpha \cdot dis(n_j, n_f)$$

ただし，dis(n_j,n_r) はノード n_j と n_r 間の距離であり，α は遠いノードの存在をどの程度考慮するかを決定するパラメータである（本稿では $\alpha = 1.0$ としている）．

n_j が VC に参加する際，存在する全てのクラスタに対して近接度を計算すると，メッセージ交換によるコストが増大する．そこで本稿では，n_j が近接度を計算するクラスタの候補を全クラスタのうち所属台数の少ない下位 $\lfloor D \times H \rfloor$ 個に制限する．ここで，$H(0.0 \leq H \leq 1.0)$ はどの程度候補を限定させるかを決定するパラメータである．この方法により，新規参加時のメッセージ交換コストを減少させるだけでなく，参加ノードを処理性能（＝ノード台数）の小さなクラスタに優先的に所属させ，特定のクラスタが肥大化することを防ぐことが可能となる．すなわち，ノードの新規参加時に Transfer アルゴリズムを適用するのと似たような効果が得られる．

4　評価と考察

4.1　シミュレーション実験の概要

提案手法の有効性を検証するために，VC のシミュレーション実験を行い，計算ターン数による評価を行った．物理ネットワークとして，1000 台，5000 台規模の 2 種類の Transit-Stub 型のインターネットトポロジをトポロジジェネレータ GT-ITM[12] により作成した．図 9 に Transit-Stub 型トポロジの例を示す．また，トポロジを作成した際のパラメータを表 1 に示す．

シミュレーション実験で用いたパラメータを表 2 に示す．本研究では全てのタスクは S 個のサブタスクから成るものとし，VC に参加するノードの処理性能は均一であると仮定する．

表 1　生成したトポロジの各種パラメータ

トポロジ内ノード数	1000	5000
Transit-domain 数	1	5
Transit-node 数	25	25
Transit-node 当たりの Stub-domain 数	4	4
Stub-domain 当たりの Stub-node 数	13	13
Transit-domain 間辺生成確率	—	0.15
Transit-domain 内辺生成確率	0.09	0.09
Stub-domain 内辺生成確率	0.15	0.15
最大ホップ数	23	30

表 2　実験パラメータ

トポロジ内ノード数 N	$1000, 5000$
参加離脱率 P	$0.005, 0.01, 0.02$
タスク数 T	$500, 1000, 1500$
サブタスク数 S	$10 \sim 50 (N = 1000)$ $60 \sim 100 (N = 5000)$
サブタスクの計算コスト	20 ターン
サブタスク間の通信コスト	所属クラスタ内の他ノードとの距離（ホップ数）の最大値
冗長度 R	2
サブクラスタ内ノード数の上限 k	6
サブクラスタ内ノード数の下限 l	2
Transfer の実行頻度 TI	20 or 0 (実行せず)
新規参加時の所属先クラスタ制限 H	$0.5, 1.0$

シミュレーションの流れを以下に示す．

1. トポロジ内に存在する全ノードのうち 70% のノードをランダムに参加させ，提案するクラスタ再構成法に従って 1 個の初期クラスタを構成する．クラスタ数が D 個となるまでクラスタの分割または統合を繰り返し，この操作を含めて 100 ターン経過させた状態をタスク配布前の初期状態とする．
2. 手順 1 で構成した初期状態の各クラスタに対して個別のタスクを割り当て処理を開始する．
3. タスクの処理が完了したクラスタに対しては未割当てのタスクを新たに配布し，処理を継続させる．T 個のタスク全てが処理完了した段階で計算プロジェクトの完了とし，シミュレーションを終了する．

なお，タスク配布の前後に関わらず，毎ターンごとにノードの新規参加と離脱をそれぞれ $N \times P$ 台ずつ発生させる．また，毎ターンごとに，必要に応じて各ノードが図 4〜7 のアルゴリズムを実行することで，クラスタ内の構成を動的に更新する．

4.2　離脱の頻度と処理時間の関係

$N = 1000, 5000$ の場合における，参加離脱率と処理時間の関係をそれぞれ図 10, 図 11 に示す．なおサブタスク数はそれぞれ $S = 20, 60$ としている．それぞれの図において，No-Clustering はクラスタリングを全く行わず，

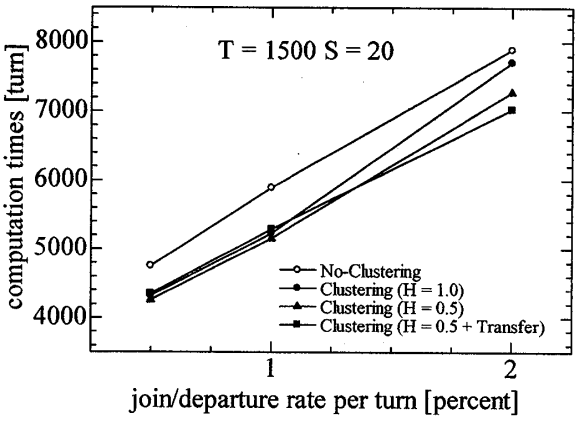

図 10　参加・離脱率に対する計算時間 ($N = 1000$)

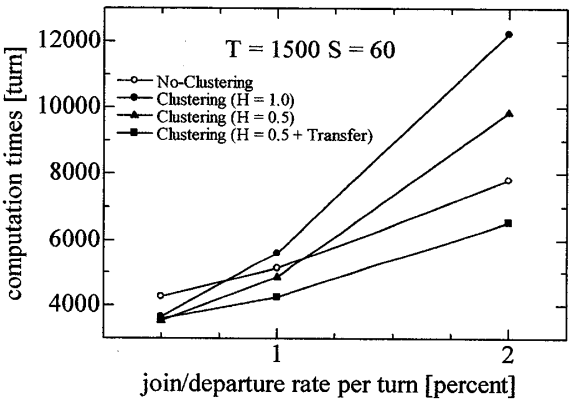

図 11　参加・離脱率に対する計算時間 ($N = 5000$)

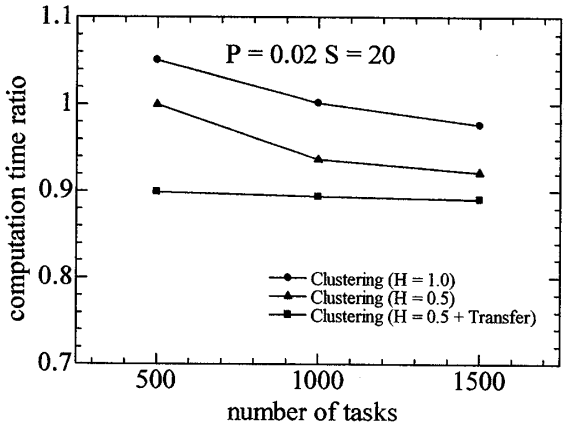

図 12　タスク数に対する計算時間比 ($N = 1000$)

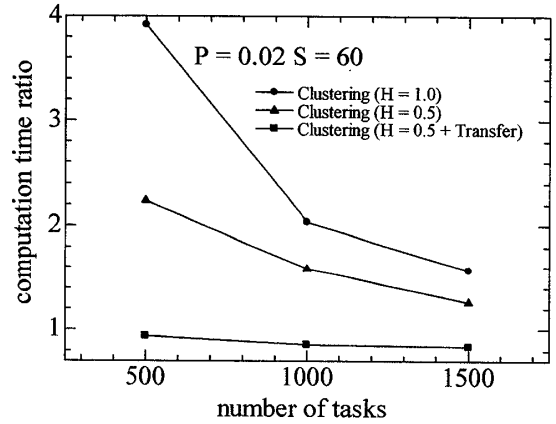

図 13　タスク数に対する計算時間比 ($N = 5000$)

D 個のタスクおよびタスク内のサブタスクをラウンドロビンでワーカに配布する手法であり，Clustering は提案手法を示している．提案手法において，$H = 1.0, 0.5$ は，3.5 節で説明した所属先クラスタの候補数を制限するパラメータ H をそれぞれ 1.0, 0.5 に設定することを意味しており，Transfer は TI ターン毎に Transfer アルゴリズムを実行することを意味している．

これらの結果から，参加離脱の頻度が大きくなるにつれて計算プロジェクトにかかる処理時間が増大していることが分かる．参加離脱の頻度が小さい場合には，クラスタリングを行わない手法に比べて提案手法がより短い処理時間で計算を完了できることが確認された．しかしながら，提案手法において Transfer や新規参加時の所属先クラスタ候補の制限を行わない ($H = 1.0$) 場合は，参加離脱率が 2.0% 程度になると処理効率の低下が見られた．これはノードの頻繁な離脱により，所属ノード数の少ないクラスタ内でのサブタスク計算において，通信の待ち時間コストが増大し，著しい処理性能の低下が起こったものと考えられる．一方で，Transfer や新規参加時の所属先クラスタ候補の制限を行った ($H = 0.5$) 場合は，離脱頻度が大きい場合にも，比較手法に比べて計算時間が短縮されていることが分かる．

4.3　タスク数と処理時間の関係

タスク数の増加に対する処理時間の関係を図 12，図 13 に示す．縦軸はラウンドロビン手法に対する提案手法の計算時間の比を表わしている．この値が 1 より小さいほど，ラウンドロビン手法に比べて提案手法の計算時間が短いことを表している．前項に示した結果と同様に，クラスタ間で処理性能を調整することにより，処理効率の向上が可能となる．また，タスク数が増大しても処理効率の向上を確認することができた．

4.4　タスクの粒度と処理時間の関係

次に，$N = 1000, 5000$ の場合において，サブタスク数，すなわちタスクの粒度と処理時間の関係を調べた．なお今回の実験では $T = 1500$，$P = 2.0$ とした．得られた結果を図 14, 図 15 に示す．タスクの粒度を大きくすると，全参加ノードを多数のクラスタに分割することができないためクラスタサイズが大きくなり，通信コスト削減の効果が期待できない．図 14, 図 15 の結果からも，サブタスク数が多い場合に提案手法における計算時間がクラスタリングを行わない手法の値に近づいていることが分かる．また，これまでの実験結果と同様に，クラスタ間の性能調整を行うことで，タスクの粒度によらず処理効率を向上させることができている．

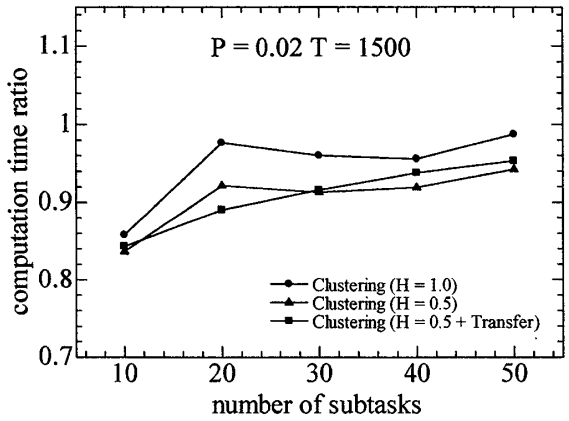

図 14　タスクの粒度に対する計算時間比 ($N = 1000$)

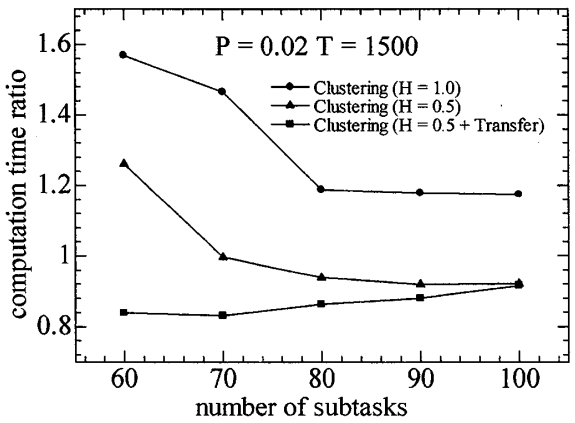

図 15　タスクの粒度に対する計算時間比 ($N = 5000$)

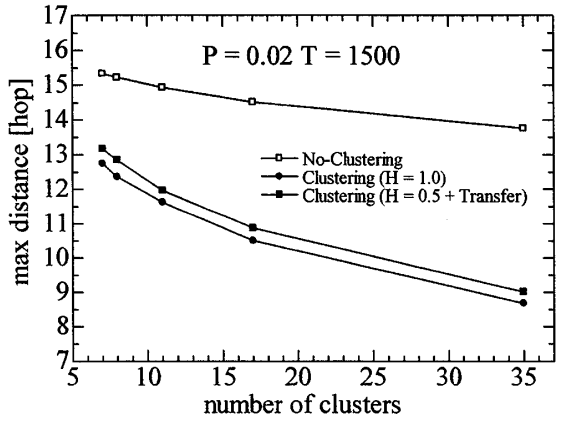

図 16　クラスタ数と最大距離の関係 (N=1000)

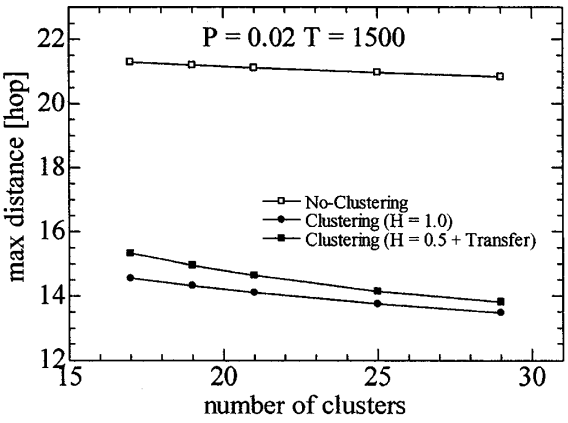

図 17　クラスタ数と最大距離の関係 (N=5000)

次に，クラスタ数に対する通信コスト（クラスタ内の他ノードとの距離の最大値）を図 16，図 17 示す．クラスタ数を多くとることで，参加ノード全体を小規模なクラスタに分割することができ，クラスタ内がより近いノードの集合体となっていることが分かる．さらに，Transfer によりクラスタ間で動的に所属ノードの移動を行う場合にも，これらクラスタリングの効果が得られていることが分かる．

5　クラスタリングに伴うメッセージオーバヘッド

動的クラスタ再構成法によって生じるノード間のメッセージオーバヘッドについて考察する．提案手法では，VC に参加しているノードを複数のクラスタに分割する．さらに，これらのクラスタはクラスタを構成するノードの自律分散的な動作によって随時その構成が更新される．クラスタ再構成の際に，ノード間においてメッセージ交換によるオーバヘッドが生じる．本節では，クラスタの規模の増大に伴ってメッセージ交換数が増大する Join，Transfer の各アルゴリズムにおけるオーバヘッドを，上田ら [11] の P2P 方式に基づくクラスタリング手法と比較する．

5.1　ノードの新規参加に伴うオーバヘッド

提案手法では，ノードが新規に VC に参加する際，所属先クラスタの候補に対して近接度計算を行う．いま，各クラスタは所属ノード数がほぼ一定数（$S \times R$）であるとし，また所属先クラスタ候補の制限を行わない場合を考える．新規参加ノードは，D 個のクラスタに対して，最も距離の近い最下位層のリーダと最も距離の遠い最下位層のリーダを Join アルゴリズムによりそれぞれ探索する．このため，提案手法では，ノードが新規に参加する際のメッセージオーバヘッドは，c を定数として $O(cDk \log_k SR)$ となる．

これに対して，上田らの手法では，ノードはクラスタの最上位層から各階層のサブクラスタにおいて，リーダとの距離だけでなく，サブクラスタ内の全てのノードとの距離測定を行い，ノードとサブクラスタ間の距離を測定する．これは，P2P 手法でクラスタリングを行う場合，本研究で用いた階層型クラスタ [6] とは異なり，各階層のノードに対して下層のノード集合の位置を代表させることが難しいためである．P2P 方式によりノードのクラスタリングを行い，提案手法と同様の新規参加方法を実行した場合，そのメッセージ交換に要するオーバヘッドは $O(cDk^2 \log_k SR)$ となる．

5.2 クラスタ間でのノードの移動に伴うオーバヘッド

次に，提案手法におけるクラスタ間のノードの移動に伴うメッセージオーバヘッドについて検討する．提案した Transfer アルゴリズムでは，下記の 2 つの操作においてメッセージ交換が多く発生する．

1. 移動対象となるコミュニティの各リーダが移動先のサブクラスタを探索する
2. 1 で探索した移動先サブクラスタを，コミュニティの他のノードへ通知する

1 については，各リーダが移動先のクラスタに対してそれぞれ Join を実行し，また 2 については，通知の際に最下位層までの各階層において再帰的にサブクラスタのメンバに対して通知を行う必要がある．このため，提案手法では，クラスタ間で所属ノードの移動を行う際のオーバヘッドは $O(k^2 \log_k SR)$ となる．

一方，上田らの手法で構築したクラスタに対して Transfer を実行することを想定した場合，前項で述べたように，Join において $O(k^2 \log_k SR)$ のメッセージ交換が必要となるため，オーバヘッドは $O(k^3 \log_k SR)$ となる．

6 まとめと今後の課題

本研究では，ノード間通信が発生するような大規模並列計算を VC で実行させることを目的に，既存の並列計算モデルを拡張した．また，VC 環境において提供された計算機資源を管理するための構造として，階層型クラスタ管理構造を構築し，これを動的かつ自律分散的に再構成する手法を提案した．シミュレーションによる性能評価から，ノードの離脱頻度が大きく，局地的な処理性能の低下が起こるような場合でも，クラスタ間で処理性能の調整を行うことで，最大で約 16%の処理効率の向上が得られることを確認した．

今後の課題としては，ノードごとの性能の変動を考慮した場合の動的クラスタ再構成法の改良が挙げられる．また，計算時間を最小にするための冗長数の最適値の導出も挙げられる．

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助成 (061102002) によって行われた．関係各位に感謝いたします．

参考文献

[1] SETI@home, http://setiathome.ssl.berkeley.edu/.

[2] Folding@home, http://folding.stanford.edu/.

[3] GIMPS http://www.mersenne.org/.

[4] distributed.net http://www.distributed.net/.

[5] Einstein@Home http://einstein.phys.uwm.edu/.

[6] S. Banerjee, S. Parthasarathy, and B. Bhattacharjee. "A protocol for scalable application layer multicast", Technical Report CS-TR-4278, Department of Computer Science, University of Maryland College Park, USA, 2001.

[7] S. Banerjee, C. Kommareddy, and B. Bhattacharjee. "Scalable peer finding on the internet", In proceedings of GLOBECOM'02, vol. 3, pp. 2205–2209, 2002.

[8] Agrawal, A. and Casanova, H. "Clustering Hosts in P2P and Global Computing Platforms", Proc. Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems, Tokyo, pp. 367-373, 2003.

[9] Krishnamurthy, B. and Wang, J. "On Network-Aware Clustering of Web Clients", Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, New York, pp. 97-110, ACM Press, 2000.

[10] Bestavros, A. and Mohrotra, S. "DNS-based Internet Client Clustering and Characterization", Proc. 4th IEEE Workshop on Workload Characterization (WWC'01), Austin, pp. 159-168, 2001.

[11] 上田達也, 安部広多, 石橋勇人, 松浦敏雄, "P2P 手法によるインターネットノードの階層的クラスタリング", 情報処理学会論文誌, Vol. 47, No.4, pp. 1063–1076, 2006.

[12] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. "How to model an internetwork", In proceedings of IEEE Infocom, vol. 2, pp. 594–602, March 1996.