一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SITE2024-116,IA2024-96(2025-03)

自律的なネットワーク内処理における ワークフローのスケジューリングに関する性能評価

新部　裕樹†　　上山　憲昭††

† 立命館大学 情報理工学研究科
〒567-8570 大阪府茨木市岩倉町 2-150
†† 立命館大学 情報理工学部
〒567-8570 大阪府茨木市岩倉町 2-150
E-mail: †gr0693pv@ed.ritsumei.ac.jp, ††kamiaki@fc.ritsumei.ac.jp

あらまし　昨今，5G の実用化などの背景から，IoT デバイスから取得されるデータを連携処理することで利用したいといった需要が高まっている．このような需要に効率的に答えるために，ネットワーク内処理という手法が検討されている．その中でも自律的なネットワーク内処理が今後求められると考える．このとき，自律分散処理における様々な制約下で，どのようにタスクをスケジュールするかという課題がある．そこで本稿では，自律的な In-Network Computing における制約条件を整理し，それらを満たす最もベーシックなタスクスケジューリングアルゴリズムを示す．また最先端のタスクスケジューリングアルゴリズムと比較し，その有効性について検討する．
キーワード　ネットワーク内処理, ワークフロー, タスクスケジューリング.

Performance analysis of scheduling for Workflow in autonomous in-network computing

Yuki NIIBE† and Noriaki KAMIYAMA††

† Graduate School of Information Science and Engineering, Ritsumeikan University
2-150 Iwakura-cho, Ibaraki, Osaka 567-8570, JAPAN
†† College of Information Science and Engineering, Ritsumeikan University
2-150 Iwakura-cho, Ibaraki, Osaka 567-8570, JAPAN
E-mail: †gr0693pv@ed.ritsumei.ac.jp, ††kamiaki@fc.ritsumei.ac.jp

Abstract　With the recent realization of 5G, there is a growing demand to utilize data acquired from IoT devices through linked processing. In-network computing is being considered as a method to efficiently meet this demand. Autonomous in-network computing will be in high demand in the future. The challenge is how to schedule tasks under the various constraints of autonomous distributed processing. In this paper, we summarize the constraints of autonomous in-network computing and present the most basic task scheduling algorithm that satisfies them. We also compare it with state-of-the-art task scheduling algorithms and discuss its effectiveness.
Key words　In-Network Computing, Workflow, Task Scheduling.

1. はじめに

近年普及が進んでいる 5G ネットワークには，大容量かつ高速・低遅延という特徴がある．これらの特徴を生かし，5G/Beyond5G の時代では，これまでにはないモノがインターネットに接続されるようになり，情報化されている．同時に，新たなアプリケーションやユースケースも登場している．その中でも，昨今の AI 技術の発展により，インターネット上の膨大なデータを利用し，情報の集約を行うことで新たな価値創造を行いたいといった需要が増加している．例えば，街中の IoT デバイスからセンシングされる情報を元に，人の移動や電力需要等を予測するアプリケーションが考えられる．また IoT の概念を車両に拡張した IoV では，車両と街中の IoT や情報インフラが統合されるため，車両単体ではなく街全体としての交通状況を制御するようなアプリケーションが考えられている [1].

このような情報の集約と分析といった処理の連携を受け入れる基盤として，仮想化技術とサービスファンクションチェイニング (SFC) を組み合わせることが考えられる．ここで，5Gネットワークにおいても同じような技術を用いてネットワーク機能が実現されている．ネットワーク機能が仮想的なファンクション (Virtualized Network Function: VNF) として表現され，汎用サーバの仮想化基盤上で実行されるようになった．このようにソフトウェアとしてネットワーク機能を実装する方法を，ネットワーク仮想化 (Network Functions Virtualization: NFV) という [2]．そして，これらの仮想化されたネットワーク機能を適切な順番で呼び出し，連携させる仕組みに SFC が利用されている．

これらを踏まえ，5G を始めとする仮想化技術をベースとしたネットワークには，ネットワーク機能に限らず，多種多様な処理の連携を実行できる可能性を秘めていると考えることができる．そして，このようにネットワーク内部でデータに対する処理を連携実行することを，ネットワーク内処理 (In-Network Computing) という．In-Network Computing では，これまで転送させるだけであったデータを，転送と同時に処理することができる．例えば，IoT 等のエッジデバイスから取得したデータを，クラウド上へ転送させる過程でそれらに対する分析の処理を行うことができる．これらのことから，In-Network Computing では遠隔地にあるクラウドでの処理と比較して遅延を削減できる利点がある．またクラウドが存在するデータセンタへ流入するトラヒックを削減できる利点もある．そのため，前述の連携処理に対する需要を，In-Network Computing をもって満たすことが考えられる．

また，これまでの In-Network Computing では，限られた範囲内でデータのやり取りを行い，処理の連携をすることが考えられていた．しかし，今後は IoT デバイスが更に加速的に普及増加すると考えられており，2030 年には 2020 年比で約 10 倍以上の市場規模となると予想されている [3]．そのような中で，社会に広く分布した大量の IoT デバイスからデータを取得し，それぞれのネットワーク経路上で連携して処理を実行するような状況や環境が求められると考えられる．そのため，そのような状況においては１つの中央集権的ノードが全ノードを管理しタスクの割り当てを行うことは困難である．これらのことから，各ノードが自律的にタスクの割り当てから実行までを行うような，自律分散型の In-Network Computing が必要である．

自律的な In-Network Computing を実現するためには，タスクスケジューリングの問題を考えなければならない．つまり，どのタスクをどこで実行するのか，それらを決定する方法について考える必要がある．また，これはアプリケーションの応答時間に直結する問題であるため，アプリケーションの実行を意識した割り当てを行わなければならない．

しかしながら，自律分散処理環境においてタスクをスケジューリングすることは，集中管理のクラウドや単一マシン上でのそれと異なる．それは，自律分散処理には大きな制約があるためである．まず，各ノードが知ることのできる情報に限りがある．各ノードは隣接ノードの情報のみを直接的に知ることができる．それよりも先の情報については，マルチホップでの問い合わせが必要となるためである．加えて全体の情報を知ろうとすることは，フラッディング等の手法を用いてネットワーク全体に問い合わせを行う必要がある．そのため，隣接ノード以外の情報を知るためには大きなコストを払わなければならない．よって，自律分散処理環境においては，隣接する前後のノードから得られる情報を用いて，なるべく低コストにタスクスケジューリングを実現する必要がある．

そこで本稿では，自律的な In-Network Computing を実現するために必要な要素を整理し，それらの要素を満たすための最もベーシックなタスクスケジューリングアルゴリズムを示す．そして，それらのベーシックなアルゴリズムを適用した自律的な In-Network Computing について，性能評価を行う．性能評価を行う際には，タスクスケジューリングにおける最先端のアルゴリズムと比較を行う．自律分散処理の制約下では直接適用することのできないアルゴリズムであるが，達成可能な性能の上限値として用いることで，自律分散型のタスクスケジューリングの有効性について考察する．

2. 関 連 研 究

2.1 Fog Computing

In-Network Computing と似た概念を提案するものとして，Fog Computing がある [4]．Fog Computing は，エッジデバイスとクラウドデータセンタの間に介在するネットワーク上で，コンピューティングやストレージサービス，ネットワークサービスを提供するための仮想化プラットフォームを提案している．つまり，エッジのレイヤとクラウドのレイヤの間に新たな中間層を構築するものである．クラウドよりもエッジに近い場所で処理を行うことで，低遅延を実現することや，リアルタイム性の高いアプリケーションへの対応ができる．また Fog (霧) のようにエッジとコアの間に計算リソースが分散することで，地理的に分散し広範囲のネットワークに対応することや，IoT デバイス等が大量に接続されることを許容している．また Fog Computing はクラウドと連携することも考えられており，低遅延やリアルタイム性といった利点を活かした処理を行う一方で，クラウドには大規模なデータの保存等を連携し任せることなどが考えられている．

Fog Computing におけるタスクスケジューリングについて，いくつかの手法が提案されている．しかし，それらは Fog Computing の階層にタスクスケジューラを用意し，そのスケジューラが集中的にスケジューリングを行う方法をとっている [5]．そのため，このような場合においてはこれまでのクラウドや分散システムで用いられてきたスケジューリングアルゴリズムを直接適用可能である．しかしスケジューラが管理できるノード数には現実的に限界がある．またエッジレイヤにあるデバイスについては管理対象外なため，それらからデータを集約するなどのタスクに関しては，Fog Computing 以外の方法で実現する必要がある．

2.2 ICN-SFC

情報指向ネットワーク (Information Centric Networking: ICN) を用いた SFC が，In-Network Computing を実現する方法として提案されている．

ICN は従来の IP アドレスベースでホストを指定する通信方法とは異なり，コンテンツ名ベースで要求するコンテンツそのものを指定して通信を行う手法である．ICN を SFC に組み合わせることで，タスクの処理結果を要求する際に，タスクがどの場所に存在するか (IP アドレス) を知る必要がなくなる．そのため，純粋にどのタスクをどこで実行すべきかという問題に集中することができる．また，ICN の持つネットワーク内キャッシュによって，処理結果の効率的な再利用が可能である．

まず，単一タスク構造のアプリケーションを想定した手法として，NFN (Named Function Networking) が提案されている [6]．処理要求を受け取った NFN ルータが，そのタスクの実行場所となり，実行するタスクと入力データを他のルータに対して要求する．そのため，常にに要求を受け取った NFN ルータが実行を担当するため，タスクスケジューリングのような問題は想定されていない．

次に，チェイン構造のアプリケーションを想定した手法として，ICN-FC が提案されている [7]．チェイン構造のアプリケーションではタスクの依存関係が鎖状につながっており，タスクの実行順序が 1 直線に決定している．そのため，この順序を満たすように，タスクを順番に実行し，その実行結果を後続のタスクの入力データとする必要がある．ICN-FC では，各タスクを保持するノードがネットワーク上に存在し，それらが常にそのタスクの実行を担当する．そのため，タスクを新たにどこかへ割り当てるといった，タスクスケジューリングのような問題は想定されていない．その代わりに，同一のタスクを複数のノードが保持していた場合に，どのノードを選択するかといった，経路選択問題がある．

提案されている 2 つの手法は，どちらも実行場所が固定あるいは決定的であり，どのタスクをどこで実行すべきかといった観点からアプリケーションの実行を最適化することができない．そのため，一般的なタスクスケジューリングを ICN-SFC に持ち込んだ処理モデルを，ここでは考える．想定するアプリケーション構造は，一般的なタスクスケジューリング問題で考えらているような，有向非巡回グラフ (Directed Acyclic Graph: DAG) 構造とする．アプリケーション構造は一般化すると DAG として表現できる．また近年アプリケーション構造が複雑化していることを踏まえ，最も汎用的に処理連携が可能となるために，DAG を想定する．そして，各タスクの実行場所は固定せず，どのノードでも実行できる処理モデルとする．そしてタスクは，一般的なコンテナタスクスケジューリング問題で考えられているように，どこかのノードが保持しており，実行場所として決定したノードはそこからタスクをダウンロードしてくるものとする．また DAG の依存関係を満たすために，各タスクは先行するタスクから処理結果を受け取る必要がある．こういった要素を踏まえ，タスクの実行時間やダウンロード時間，後続タスクへの処理結果の受け渡し時間等を全て考慮したう上で最適となる実行ノードを選び，スケジュールする必要がある．

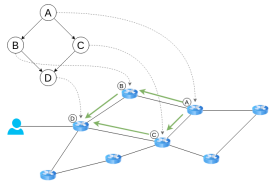

図 1　In-Network Computing の動作例

本稿では，ここで想定した DAG 構造のアプリケーションを想定した ICN-SFC を，In-Network Computing の実現方法の 1 つの例として採用する．そして，本稿の評価実験の際には，この採用した処理モデルを用いて実験を行う．

3. In-Network Computing の動作

DAG 構造のアプリケーションをスケジュールし，実行する場合の In-Netowork Computing における動作を説明する．図 1 に，In-Network Computing の動作例の図を示す．ここでは例としてタスク A, B, C, D からなるアプリケーションが与えられている．このアプリケーションは DAG 構造で示せる依存関係を持っている．そのため，この依存関係が満たされるようにタスクをスケジュールする必要がある．加えて，各タスクはそれぞれ自身の処理結果を後続タスクに渡せるよう，受け渡し先がどこであるか知っている必要がある．このために，ノードごとに経路表を作成し，処理結果が必ず後続タスクに流れていくようにしなければならない．なんらかの指標に従ってこのアプリケーション内のタスク全てを割り当てた後に，先頭のタスクから実行が始まる．ここで，図中では省略されているが，各ノードは割り当てられたタスクを自身に配置するために，そのタスクを配布しているノードからダウンロードしてくる必要がある．そして，各タスクは依存関係として持っている全ての先行タスクの処理結果を持ってして処理が行われる．そのため，先行タスクの処理結果が到着するまでは，実行せずに待機することとなる．ここでは，タスク A の処理結果が後続タスクであるタスク B, タスク C に受け渡され，それぞれが処理を行う．最後に，タスク B とタスク C の実行結果がタスク D に受け渡される．ここでタスク B とタスク C の両方の処理結果が到着してから，タスク D は処理が開始される．このように，先頭タスクから後続タスクに向かって，タスクが順に実行されることで，In-Network Computing においてアプリケーションが実行される．

4. 自律的な In-Network Computing を実現するための制約条件

自律的な In-Network Computing を実現するためには，いくつかの制約条件がある．そのため，それらの制約条件を満たす形で，タスクスケジューリングを行わなければならない．そこで，まずそれら制約条件について，整理を行う．

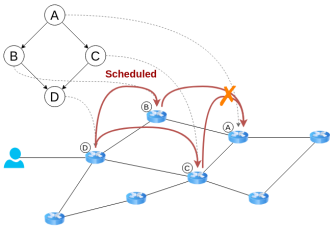

図 2　制約条件 2 によって実現できない挙動の例

1 つ目の制約条件は，スケジューラの役割を行う中央集権的なノードが存在しないことである．従来の In-Network Computing においては，各ノードの状況を把握しているスケジューラがおり，それが集中的に全てのタスクの実行場所を決定していた．自律的な In-Network Computing においてはそれができないため，各ノードが自律的にタスクをスケジュールしなければならない．そのため，DAG アプリケーションにおいては，後続タスクが割り当てられたノードが，直接の先行タスクを他のノードにスケジュールする必要がある．後続タスクが先行タスクをスケジュールする形を繰り返し行うことで，中央集権ノードの存在しない自律分散処理においても，DAG アプリケーションのタスクスケジューリングが可能である．

2 つ目の制約条件は，互いのノードは自律的に動作しているため，それらの間で同期を取ることが困難であるということである．図 2 にこの制約条件によってできない動作を示す．タスク B とタスク C は共通の先行タスクであるタスク A を持つ．そのため，タスク A を 1 つのノードで実行し，その処理結果をそれぞれで共通利用したい．しかし，タスク B とタスク C はそれぞれ別のノードに割り当てられている．ここで，互いのノードは自律的に動作しているため，それぞれのノードが先行タスク A をどこに割り当てようとしているか，知り得ない．よって，タスク B とタスク C が同期を持って先行タスクを同じ場所に割り当てることはできない．そのため，DAG 構造のような依存関係が分岐する場合に置いては，先行タスクが必ず 1 つの場所に割り当てできるような，他の仕組みを考える必要がある．図 3 に，この DAG 構造をトポロジカルソートした例を示す．トポロジカルソートとは，DAG の依存関係を満たした形で，各タスクを一直線にソートする方法である．一直線上に並んでいるが，DAG の各タスクの親子関係は崩れることなく，維持されている．またこの関係性を最低限守っていればトポロジカルソートできているため，必ずしもこの順序でなくてもよい．今回の場合は A, C, B, D の順に並んだが，A, B, C, D の順でも問題ない．そして図 4 に，このトポロジカルソートを用いて自律的にタスクの割り当てを行う様子を示す．これは，先程のトポロジカルソートの結果を用いて，1 つ目の制約条件に従って後続タスクから先行タスクに向かってタスクの割り当てを行ったものである．このように割り当てることで，先行タスクが必ず 1 つの場所に割り当てられることを保証できる．加えて，このときに経路表を適切に設定するととで，これまでと同じように各タスクは後続タスクに処理結果を渡すことができる．

図 3　トポロジカルソートの例

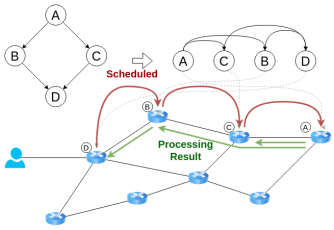

図 4　トポロジカルソートを用いた自律的なタスクの割り当ての例

3 つ目の制約条件は，タスクの割当先を決定する際に，各ノードが参照できる情報は隣接ノードの情報のみということである．各ノードは直接的に繋がっている隣接ノードの情報については，比較的容易に取得することができる．しかし，それより先のノードの情報を取得するためには，マルチホップで問い合わせをする必要があるため，大きなコストがかかる．加えて，ネットワーク全体の状況を知るためには，問い合わせのパケットをフラッディングによってネットワーク全体へ行き渡らせる必要があるため，更に高コストである．通常のタスクスケジューリングにおいてタスクの割当先を決定する際には，全ノードの中から最適な場所を探すことになる．このような操作は，自律分散処理環境においてはできない，あるいはできたとしても非常にコストがかかってしまう．そのため，なるべく全体最適となるような割り当てを，隣接する前後のノードから得られる情報のみで行う必要がある．逆に言えば，隣接ノードから得られる情報のみを使うことで，自律分散処理環境においては非常に低コストにタスクの割り当てを行うことができる．

5. 制約条件を満たしたタスクスケジューリングアルゴリズム

制約条件が 3 つある中で，なるべく全体最適となるような，自律的なタスクスケジューリングを行う方法を考えなければならない．ここで，タスクスケジューリングには大きく分けて 2 つのフェーズがある．1 つ目が，DAG 構造内のタスクを優先順序付ける Task Ordering である．2 目が，優先順序付けられたタスクを適切なノードに割り当てる Task Assignment である．この 2 つのフェーズについて，制約条件 1 の後続タスクから先行タスクに向かって割り当てを行うという前提のもと，アルゴリズムを考える．

5.1 Task Ordering

Task Ordering のフェーズについては，制約条件 2 の，DAG の依存関係を満たした状態で順序付けるというものが当てはまる．つまり，トポロジカルソートを用いて，タスクに順序を付けることを考える必要がある．トポロジカルソートを行う方法として，ReadyList を用いた順序付けアルゴリズムを提案する．ReadyList は，全ての後続タスクが順序決定済となっ

たタスクが入るリストである．図 5 に，ReadyList を用いてトポロジカルソートした場合の様子を示す．ここで，制約条件 1 の後続タスクから先行タスクへスケジュールするという前提条件と合わせるため，一番後続の END タスクからトポロジカルソートを開始し，一番先頭の START タスクに到達するまで行っている．この操作によって，最低限満たさなければならない条件を満たした形で Task Ordering が完了した．

一方で，同一タイミングで ReadyList に入ったタスク間では，順序が決定していない．Task Ordering では完全な順序を決定する必要があるため，これについても何らかの指標を持って順序付けなければならない．ここで，最もベーシックな方法として，各タスクの先行タスク数，後続タスク数を見て順序付ける方法を提案する．先行タスクが多いタスクは，中々必要な処理結果が集まらず，実行が遅れる可能性が高い．そのため，優先度を下げるという考え方ができる．一方で，後続タスクが多いタスクは，そのタスクの実行が遅れてしまうと，その遅れの影響が及ぼす範囲が大きい．そのため，優先度を高くするという考え方ができる．以上 2 つの考えを元に，ここでは各タスクの後続タスク数を先行タスク数で割ったものを優先度の指標とし，同時に ReadyList に入ったタスク間で比較することとする．図 6 にこの指標を元に優先順序を決定した様子を示す．図では 3 つ目の ReadyList について優先順序の計算を行っているが，実際には全ての ReadyList に対して操作を行う．

以上の ReadyList を用いた最低限のトポロジカルソートと，先行/後続タスク数を用いた指標の 2 つを用いることで，DAG を完全な順序付けすることができる．

5.2 Task Assignment

Task Assignment のフェーズについては，制約条件 3 の，隣接する前後のノードの情報から割り当ての判断を行うというものが当てはまる．ここでは最もベーシックな判断方法として，タスクスケジューリングの中で最先端アルゴリズムのうちの 1 つである，HEFT (Heterogeneous Earliest Finish Time) でも用いられていた，Blevel という指標を利用することとする．

Blevel は，そのタスクから END タスクまでの最悪の残り時間である．つまり，そのタスクから END タスクまでの複数ある経路 (各後続タスクを経由したパス) のうち，もっとも経路長の長いものを Blevel として用いる．このとき，制約条件 1 によって，END タスクから START タスクに向かってタスクの割り当てを行っているため，Task Ordering の順序に従って Task Assignment をする場合，そのタスクから END タスクまでの全てのタスクは既に割り当て済である．そして割り当て済みのタスクに関しては，DAG ファイルに詳細を記録して渡していくことにより，先行タスクは後続タスクの割り当て結果について知ることができる．よって，そのタスクは Blevel を計算することができる．Blevel については各ノード間の帯域幅とタスクのデータ量の重み，各ノードの処理能力とタスクの処理の重みを用いることで，予想される残り時間が算出される．その中で最悪のものが Blevel となる．

このとき，中央集権的な分散処理では全てのノードに対して Blevel を計算し，Blevel が最小にできる場所を割当先として

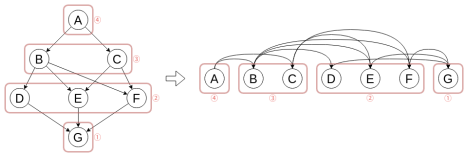

図 5　ReadyList を用いたトポロジカルソートの例

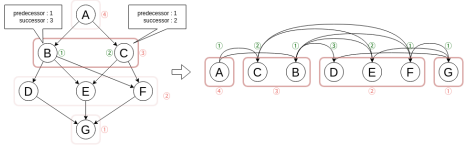

図 6　先行/後続タスク数を用いた指標による優先順序付け

選ぶことになると思われる．しかしながら，自律分散処理においては制約条件 3 の隣接ノードの情報しか判断材料がない．そこで，各ノードは，その次に割り当て先を探すタスクについて，自身のノードの Blevel と，経路表から選択された隣接ノードの Blevel を計算する．そして，自身の Blevel の方が小さかった場合には自身にそのタスクを割り当て，隣接ノードの Blevel の方が小さかった場合には隣接ノードに割り当てを委任する．ここで隣接ノードに割り当てが委任された場合，その隣接ノードは再度同じように自身の Blevel と，その先の隣接ノードの Blevel について計算を行い，割り当てを委任すべきか，自身に割り当てるべきかの判断を行う．このように，先行タスクの割り当て要求が回ってきたノードについては，自身で実行したほうがよいのか，はたまた経路表から選択される隣接ノードに割り当てを委任したほうがよいのか，この点にのみ判断を行う．これが繰り返されることで，ある程度 Blevel が最小化されたノードに Task Assignment をすることができる．

6. 性能評価

6.1 実験環境

前章で示した最もベーシックなタスクスケジューリングアルゴリズムについて，計算機シミュレーションを用いて評価を行う．ここでは，自律的な In-Network Computing の 1 実現例として，DAG 構造のアプリケーションを想定した ICN-SFC にて実験を行う．実験には icn-sfcsim というシミュレータ [8] を用いた．表 1 に，評価環境のパラメータを示す．ICN Node がアプリケーションの実行要求を行うユーザであり，また処理対象のデータを持つエッジデバイスでもある．ICN Router がその間に介在するネットワークのノードである．そのため，タスクはこの ICN Router に対してスケジュールされる．表 2 に投入するアプリケーションのパラメータを示す．各アプリケーションごとのタスク数を 25 とし，それぞれのタスクが持つ後続タスク数を 1〜5 の間としている．そしてランダムに DAG を生成し，ネットワークに投入している．そして，これらのランダム DAG による実験においては，アプリケーションの CCR (Communication to Computation Ratio) を 0.1 から 10.0 まで変化させならが実験する．これは，CCR がアプリケーションの持つ特徴を表すパラメータであるため，変化させることにより

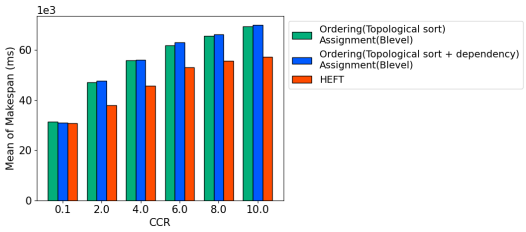

図 7　各アルゴリズムにおける平均 MakeSpan の変化

アルゴリズムの汎用性を評価するためである．

評価対象のアルゴリズムは 3 つである．まず，Task Ordering のトポロジカルソートとタスクの依存関係を用いた優先順序付けを行った場合である．次に，Task Ordering のトポロジカルソートのみを行なった場合である．どちらも Task Assignment は Blevel によって行う．最後に，達成可能な性能の上限として，集中管理で HEFT アルゴリズムを実行した場合である．これらを比較し，自律的な In-Network Computing におけるタスクスケジューリングがどの程度有効か，評価を行う．

表 1　実験環境のパラメータ

Parameter	Value
ICN Node #	100
ICN Router #	500
CPU MIPS	2000 - 4000
Bandwidth (MBps)	100 - 1000

表 2　アプリケーションのパラメータ

Parameter	Value
# of tasks for each Apl.	25
Max # of successors for each task	5
CCR for each Apl.	(0.1, 2.0, 4.0, 6.0, 8.0, 10.0)

6.2 実験結果

図 7 に，実験結果のグラフを示す．このグラフでは，各アルゴリズムにおける平均の MakeSpan を比較している．今回示した自律分散処理におけるベーシックなタスクスケジューリングアルゴリズムとして，トポロジカルソートのみで後はランダムな状態で Task Ordering を行なったものと，トポロジカルソートに加えて，タスクの先行/後続タスク数を用いて Task Ordering を行なったものがある．どちらも Task Assignment に関しては Blevel を用いて行なっている．比較の結果，この 2 手法については大きな差はみられないことがわかった．ここから，今回の実験ではベーシックなアルゴリズムとして各タスクの先行/後続タスク数を用いて Task Ordering を示したものの，その効果は限定的であったと考えられる．これは，実際のアプリケーション実行時には Task Ordering の順序通りにタスクが実行されるわけではなく，条件を満たしたタスクから順に並列に実行が行われるためであると考えられる．つまり，自律的な In-Network Computing における制約条件がある中では，Task Ordering によって詳細に順序付けるよりも，Task Assignment の方法により，タスクがどこに割り当てられるかについての方が，アプリケーションの実行時間に大きな影響があることがわかった．

また自律的な In-Network Computing におけるタスクスケジューリングの結果と，集中管理における最先端のアルゴリズムである HEFT とを比較すると，大きな差があるものの，分散処理として成り立たないほどの性能劣化があるわけではないとわかった．そのため，自律的な In-Network Computing で様々な制約条件があることを考えると，HEFT には劣るものの，十分に有効といえる範囲に性能が収まっていると考えられた．

7. まとめと今後の展望

本稿では，自律的な In-Network Computing を実現するための制約条件を整理した上で，最もベーシックなアルゴリズムを示した．そして，それらのベーシックなアルゴリズムを用いた場合に，自律的な In-Network Computing におけるタスクスケジューリングがどの程度有向なものであるか，考察を行った．この結果から，最先端のアルゴリズムの 1 つである HEFT による割り当て方には劣るものの，性能劣化の割合としては許容できる範囲に留まっており，また最もベーシックなアルゴリズムであることから考えると，自律的なタスクスケジューリングとして十分な有効性があると考えられた．また，自律分散処理におけるタスクスケジューリングでは，詳細に Task Ordering するよりも，各タスクをどこに割り当てるかについての方が重要度が高いとわかった．

今後はこのベーシックなアルゴリズムを元に，HEFT の性能に近づくことを目標に，ヒューリスティックなアルゴリズムを構築したい．また今回はネットワーク環境を比較的大きなものとして実験を行なったため，仮に規模の小さいネットワークに In-Network Computing を適用する場合，自律的であることのメリットを含め，性能等の有効性を検討していきたい．

文　献

[1] B. Ji et al., ”Survey on the Internet of Vehicles: Network Architectures and Applications,” in IEEE Communications Standards Magazine, vol. 4, no. 1, pp. 34-41, March 2020.

[2] R. Mijumbi, J. Serrat, J. -L. Gorricho, N. Bouten, F. De Turck and R. Boutaba, ”Network Function Virtualization: State-of-the-Art and Research Challenges,” in IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 236-262, Firstquarter 2016.

[3] S. Al-Sarawi, M. Anbar, R. Abdullah and A. B. Al Hawari, ”Internet of Things Market Analysis Forecasts, 2020 – 2030,” 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), London, UK, 2020, pp. 449-453.

[4] F. Bonomi, R. Milito, J. Zhu と S. Addepalli, ”Fog computing and its role in the internet of things”, Proceedings of the first edition of the MCC workshop on Mobile cloud computing, MCC ’ 12. New York, NY, USA: Association for Computing Machinery, Auguest 2012, pp. 13 – 16.

[5] M. R. Alizadeh, V. Khajehvand, A. M. Rahmani と E. Akbari, ”Task scheduling approaches in fog computing: A systematic review”, International Journal of Communication Systems, vol. 33, no. 16, p. e4583, November 2020.

[6] C. Tschudin and M. Sifalakis, ”Named functions and cached computations,” 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 2014, pp. 851-857.

[7] L. Liu et al., ”ICN-FC: An Information-Centric Networking based framework for efficient functional chaining,” 2017 IEEE International Conference on Communications (ICC), Paris, France, 2017, pp. 1-7.

[8] 金光 永煥，花田 真樹，金井 謙治，中里 秀則：ワークフローにおける ICN を用いたファンクションチェイニング，信学技報，vol. 119, no. 461, IN2019-120, pp. 249-254, 2020 年 3 月．