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

信学技報
TECHNICAL REPORT OF IEICE.

# 自律的 In-Network Computing における タスク粒度を考慮した重複スケジューリング

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**　ネットワークやコンピューティング技術の進歩によって，IoT デバイスが急速に普及してきている．IoT デバイスからセンシングされるデータ (IoTBD) にはよりよい社会を実現する上で価値となる情報が含まれており，これらに対して処理を連携させながら分析していくことが求められている．このような処理の連携を効率よく実現するために，クラウドだけでなくネットワーク全体で処理を提供する，In-network computing が検討されている．特に，今後も増大する IoT デバイスをサポートするためには，自律的 In-network computing によって処理の連携する手法が必要である．しかしながら，このような自律分散型の処理基盤ではタスクの重複割当が発生し，計算資源利用の効率性に問題が生じる．計算資源を有効に活用するためには，タスクを重複せずに実行することが望ましい．しかし，実際には追加で計算資源が利用できる状況下において，あえてタスクの重複を許容することによって，アプリケーションの実行性能を向上させたい場合がある．この計算資源の利用効率とアプリケーションの実行性能の間には，トレードオフの関係がある．そこで本稿では，必要最小限のタスクのみを重複実行することで遅延を削減する手法を提案する．シミュレータを用いて提案手法の評価を行い，どのような特性を持つかを明らかにする．
**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.

# Grain-Aware Task Duplication Scheduling for 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 advances in network and computing technologies, IoT devices are becoming rapidly widespread. Data sensed by IoT devices (IoTBD) contains valuable information for realizing a better society, and there is a need to analyze this data while coordinating processing. To efficiently achieve such collaborative processing, in-network computing is being considered. In particular, supporting the ever-increasing number of IoT devices requires methods for coordinating processing through autonomous in-network computing. However, in such an autonomous, distributed processing infrastructure, duplicate task assignments can occur, leading to inefficiencies in computational resource utilization. To effectively utilize computational resources, it is desirable to execute tasks without duplication. In practice, however, a trade-off exists between computational resource utilization efficiency and application execution performance. Therefore, this paper proposes a method to reduce latency by executing only the minimum necessary tasks in duplicate. We evaluate the proposed method using a simulator to clarify its characteristics.
**Key words**　In-Network Computing, Workflow, Task Scheduling.

## 1. はじめに

近年のネットワークおよびコンピューティング技術の飛躍的な進歩により，あらゆるモノやコトがインターネットを介して接続される社会基盤が形成されつつある．特に，高速大容量・低遅延・多数同時接続を特徴とする 5G/Beyond 5G の普及は，これまで以上に多様な IoT デバイスのネットワーク参加を加速させている． スマートシティ，スマートホーム，ヘルスケア，産業オートメーションといった多岐にわたる IoT アプリケーションは，その稼働に伴い膨大なデータを生成する．これらは IoT Big Data (IoTBD) と呼ばれ，その量・速度・多様性といった特性が従来のデータとは異なるものである [1]．IoTBD から実用的な価値を創出するためには，セマンティクスによる分析処理が不可欠であり，これによって物流最適化やエネルギー管理といった高度な意思決定支援が可能となる [2]． また，IoTBD の活用は，物理空間をサイバー空間上に再現するデジタルツインや Cyber Physical System の実現においても重要である [3], [4]．さらに，IoT の概念をモビリティへ拡張した IoV (Internet of Vehicles) においては，車両や都市インフラから生成されるデータをリアルタイムに解析し，個々の車両制御のみならず都市全体の交通流最適化に役立てることが求められている [5]．そのため，IoTBD のようなエッジに広がる新たなビッグデータを活用するためには，様々な処理の連携によってアプリケーションを実行する基盤が必要である．

しかしながら，エッジ領域に広がる IoTBD を活用するための計算基盤には，解決すべき課題が残されている． 従来，大規模データの処理は潤沢なリソースを持つクラウドデータセンタに集約されることが一般的であった．しかし，IoTBD の爆発的な増加とアプリケーションのリアルタイム性要求の高まりに伴い，クラウド集約型のアプローチは限界を迎えつつある．これに対し，エッジコンピューティングが提案されているが，センサや車両等のエンドデバイスは計算資源やバッテリ容量に制約があり，複雑な処理の実行は困難である [6]．また，単にエッジサーバを配置するだけでは，動的なトラヒック変動への追従や全体最適化としては不十分である．

これらの課題を解決する新たな手法として，In-network computing (INC) が注目されている．INC は，ネットワークデバイスに計算能力を持たせ，データ転送経路上で処理を行う手法である [7]．これにより，不要なデータ転送を抑制し，エッジコンピューティングよりも高いスループットと，クラウドよりも低い遅延を同時に実現することが可能となる． しかし，既存の INC 研究の多くは，限定された範囲のデバイスを中央集権的に管理することを前提としている．2030 年には IoT デバイス数が現在の 10 倍以上に達すると予測される中で [8]，広範囲に分散する膨大なデバイスと計算リソースを中央集権的に制御することは，スケーラビリティの観点から現実的ではない．したがって，ネットワーク上のノードが自律的に連携し，アプリケーションを実行する，自律的な In-Network Computing の実現が求められる．

本稿では，この自律的な分散処理基盤を実現するアプローチとして，情報指向ネットワーク (ICN: Information-Centric Networking) と Service Function Chaining (SFC) を統合した手法である，ICN-SFC に着目する [9], [10]．従来の IP ベースの SFC では，計算リソースの場所と識別子が密結合しており，SDN コントローラ等による集中的な管理が必要であった．これに対し ICN-SFC では，ユーザは目的のアプリケーションを名前で指定するだけでよく，ネットワークが自律的に最適な機能やコンテンツを発見・ルーティング・実行することが可能である．また，ネットワーク状態やリソース負荷に応じた動的な制御も統合的に実現できる． 以上のことから，IoTBD においてスケーラブルかつ低遅延な INC を実現するためには，ICN-SFC が構造的に適しており有用である．

昨今の IoT アプリケーションの高度化・複雑化 [11] を踏まえると，INC の実現には，単純な直列状のサービスチェインだけでなく，依存関係が分岐・合流する有向非巡回グラフ (DAG) 構造への対応が不可欠である [10]．しかし，自律的なノード連携によって DAG アプリケーションを実行する場合，タスクの重複実行という課題が生じる．ICN-SFC では通常，最終タスクから開始タスクへ向けて逆順に要求を送信し，実行ノードを確定させた後に順方向へ処理を進める．この際，DAG 構造においては一つのタスクが複数の後続タスクから要求される状況が発生する．自律分散環境では大域的な調整や同期が困難であるため，各ノードが自身の経路表 (FIB) に従い異なる経路で要求を転送した結果，同一タスクが複数のノードで重複してスケジューリングされる可能性がある．

図 1 に，これにより引き起こされるタスクの重複実行の様子を示す．この無制限な重複実行は，計算資源の浪費に繋がる．私達はこれまで，計算資源の有効活用の観点から，このタスクの重複を完全に排除するスケジューリング手法を検討してきた [12]．しかし，重複の排除はリソース節約に寄与する一方で，必ずしもアプリケーションの実行時間短縮にはつながらない．計算資源に余力がある環境下では，ある程度意図的に重複実行を許容することで，通信遅延や処理待ち時間の影響を隠蔽し，実行性能を向上させられる可能性がある．すなわち，計算資源の消費量とアプリケーションの実行性能の間にはトレードオフが存在する．

そこで本稿では，DAG 構造と各タスクの粒度に着目し，アプリケーション全体の実行時間短縮を目的として，投機的にタスクを重複実行する手法を提案する．提案手法は，無制限に重複を許す場合よりも計算資源消費を抑制しつつ，重複を完全に排除する場合よりも高い実行性能を実現することを目指す．これにより，資源効率と実行速度のトレードオフにおいてバランスの取れたタスクスケジューリングを実現する．また，計算機シミュレーションによる評価を通じ，計算資源消費と実行性能の関係性における本手法の特性と有効性を明らかにする．

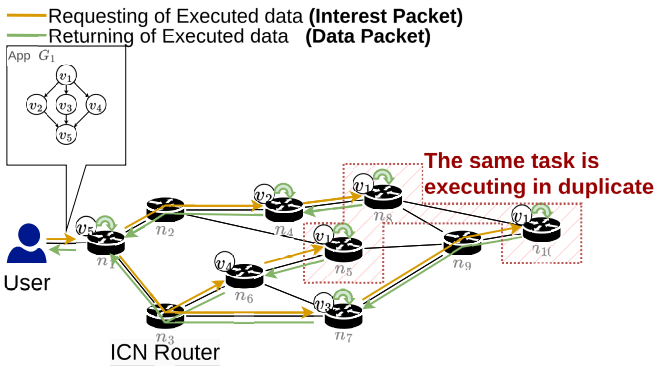


図 1　自律的な In-network computing でタスクが重複する様子

## 2. 関連研究

### 2.1 ICN-SFC

ICN では IP アドレスといったホストを指定した通信を行うのではなく，取得したいコンテンツの名称を指定して通信を行う．そのため，コンテンツを要求する際にコンテンツを保持するホストのアドレスを取得・解決する必要がない．このパラダイムは，ICN-SFC におけるタスクの連携においても適用できる．つまり，タスクを実行するノードのアドレスを指定するのではなく，タスクそのものを指定することでタスクの実行要求をする．

ICN-SFC の中でも DAG 構造のアプリケーションを想定したものとして，AutoICN が提案されている [10]．AutoICN の実行フローは，単純な Longest match によるパケット転送ではなく，自律的なタスクスケジューリングとして機能する．各ノードは Interest を受け取ると，DAG 全体の実行完了時間を最小化するための判断を行う．これは，タスクを自身で実行するべきか，あるいは他の適切なノードへ転送するかの決定である．具体的には，タスクの実行完了予測時間やデータ転送時間に基づいて *blevel* (bottom level) と呼ばれる指標を計算し，ネットワークの動的な状況に応じた最適なタスク割り当てを行う．これは，通常の SFC が計算リソースへのルーティングに主眼を置いているのに対し，AutoICN はタスク間の依存関係と計算資源の配分を考慮した「スケジューリング」を ICN の転送プレーン上で自律的に行う点に特徴がある．つまり，AutoICN でも他の SFC 同様に，既に配置済みのタスクへのルーティングを行うが，その途中で実行場所として実行完了時刻短縮に最適な計算資源が見つかった場合，そこに新たにタスクを割り当て，実行場所として決定する．AutoICN では，実行時間短縮に貢献できるノード全てが実行場所になり得る．

AutoICN は単一の Interest フローに対しては効率的な割り当てを行うことができるが，DAG 構造を持つアプリケーションにおいては，タスクの重複割り当てという重大な課題が存在する．DAG 構造のアプリケーションでは，先行タスクの処理結果が，分岐した複数の後続タスクによって必要とされるケースが発生する．これを，Fork 構造と言う．タスク割り当てが進行し，後続タスク群がそれぞれノード A とノード B という異なるノードに分散して配置された場合を考える．このとき，各後続タスクは，自身の実行に必要な共通の先行タスクに対して，個別に実行要求を送信することになる．AutoICN のような自律分散環境では，各ノードは自身のローカルな FIB とネットワーク状況に基づいて独立してルーティングを行う．そのため，ノード A から送信された先行タスクへの要求と，ノード B から送信された要求が，それぞれ異なる経路を通り，最終的に別々の計算ノードに到達してしまう可能性がある．このように，同一タスクに対する実行要求であるにもかかわらず，要求元の違いやネットワーク上の経路選択の結果として，別々のノードで処理が開始されてしまう現象が発生する．我々はこの問題を，DAG の分岐したフロー (Fork) が正しく合流 (Join) できず，処理が拡散してしまう様子になぞらえて，Fork Not-Join (FNJ) 問題と呼ぶ．FNJ 問題が発生すると，本来一度だけ実行されればよいはずのタスクが，ネットワーク内の複数箇所で重複して実行されることになる．これにより，計算資源の浪費によるネットワークの圧迫や効率の低下といった問題が生じる．

私達のこれまでの取り組みでは，タスクを重複実行がない形でスケジューリングするため，DAG をトポロジカルソートによって一直線に並び替えた上で必ず 1 つずつ実行要求していくことで，FNJ 問題を解決した [12]．トポロジカルソートの例を図 2 に示す．しかしながら，AutoICN のような自律的な環境において，1 回のルーティングで良好な実行性能を得られるノードへたどり着ける可能性は高くない．加えて，DAG を一直線に並び替え，一筆書きのようにタスクをスケジュールしていく都合上，処理結果の返送経路長が長くなってしまう．このような理由から，従来手法の無制限な重複を許容する手法と比較すると，大幅に実行性能が劣化してしまった．

これらのことを言い換えると，重複実行によってタスクを複数回要求することで良好な実行性能が得られる可能性を高め，実行時間を短縮することが考えられる．加えて，重複要求時には一筆書きのようにタスクをスケジュールしていくフローから離脱して新たな要求を始める形になるため，より短い返送経路長でそのタスクから先を実行できる．このことから，タスクの重複実行によって様々な場所へ要求することは，実行性能の向上を期待して投機的にタスクを重複すると捉えると，良い面もある．

無秩序なタスクの重複実行では，アプリケーション全体の実行時間短縮に寄与しないタスクまでもが重複実行され，計算資源が無駄に消費されてしまう．そのため，重複実行によって実行時間を短縮する場合には，アプリケーション全体の実行時間短縮に影響度の高い，ボトルネックとなっているタスクに限って重複するのがよい．そこで本稿では，重複対象のタスクと重複実行回数の両方を効果のある範囲に限定して重複スケジュールすることで，実行性能の向上を狙いつつ，消費する計算資源の量を抑える手法を提案する．

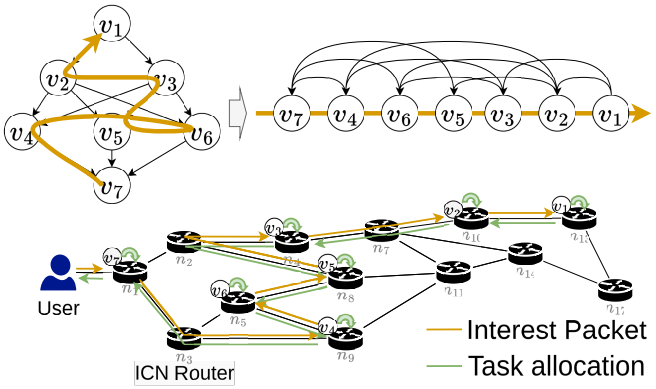


図 2　トポロジカルソートによる重複排除の例

## 3. 提案手法

タスクを実行時間の短縮を期待して投機的に実行することは有効であるが，無制限な重複はネットワークを圧迫し，不必要な計算資源消費から効率を低下させる．そこで本節では，ICN-SFC において重複対象のタスクと最大重複回数を限定してタスクを重複することで，消費する計算量を抑えつつアプリケーション全体の実行時間を短縮させる方法を提案する．具体的には，DAG の静的解析に基づく重複のための「予算 (Budget)」の決定と，経路上の動的統計情報に基づくタスクの「粒度 (Grain)」を用いて，適応的な重複実行の判断を目指すものである．本手法は，2 段階の制御を行う．Phase1 が DAG の

静的解析と予算配分，Phase2 が動的情報での重み付けと重複判断である．つまり，Phase1 が静的な情報を扱い，Phase2 が動的な情報を扱う．次節からは，それぞれのフェーズに分かれて提案手法の詳細を説明する．

### 3.1 Phase1: 静的解析と予算配分

Phase1 では DAG を静的に解析し，全体の実行時間短縮のために重複対象とすべきタスクと，その重複数を決定する．まず，アプリケーションの DAG $G = (V, E)$ に対して静的解析を行う．全体の処理時間を最小化するため，重複の対象は DAG のクリティカルパス $CP \subseteq V$ 上のタスクに限定する．クリティカルパスはアプリケーション全体の実行時間を決定するため，クリティカルパス上のタスクの実行時間が短縮されると，全体の実行時間も短縮される可能性が高い．そのため，ここではクリティカルパス上のタスクをボトルネックタスクと見なし，重複対象とする．

次に，無制限な重複を防ぐための，アプリケーション全体での最大の重複実行回数を決定する．不必要な量の重複を避けつつ並列性を最大化することで重複実行の効果を得るため，Brent の定理に基づく理想的な並列度を総予算 $B_{total}$ の基準とする．アプリケーション内の総計算量 $W$ とクリティカルパス上の総計算量 $L$ を用いて，以下のように定義する。

$$B_{total} = \frac{W}{L}$$

Brent の定理は，Work stealing という手法においてタスクスケジューリングの分野に適用されている [13]．Work stealing は，暇なプロセッサが忙しいプロセッサからタスクを盗み (重複し) 実行するというものである．ここで，アプリケーションの並列度が理想的な並列度 $W/L$ に近い場合，タスクを盗む動作がランダムに行われたとしても，最適に近いスケジュールが得られるとしている．ここで，本提案手法で考えている投機的なタスクの重複は，Work stealing を push 型で逆に行っているものと言える．そのため，本手法においても最大の重複数を理想的な並列度 $W/L$ に限定することで，重複の効果を得つつ必要以上に計算資源を消費することを抑えられると考える．

最後に，決定された重複予算 $B_{total}$ を各重複対象タスクに割り振る．ここで予算の割り振りを効果的に行うため，各重複対象タスク $v_x \in CP$ の静的粒度 $g_{static}(v_x)$ を算出する．粒度算出のための各タスクの関係性を，図 3 に示す．粒度は計算コスト $w(v_x)$ と通信データ量 $d(v_x)$ の比率であり，タスク $v_x$ の Fork set $F_x = v_1, v_2, ..., v_m$ の粒度 $g(J_x)$ と Joint set $J_x = v_1, v_2, ..., v_m$ の粒度 $g(F_x)$ を用いて，以下の式で算出される [14].

$$g(J_x) = \frac{\min\limits_{k=1:m} D_p(v_x, \alpha_{END})}{\max\limits_{k=1:m} D_c(d_{k,x}, L_{END})}$$

$$g(F_x) = \frac{\min\limits_{k=1:m} D_p(v_x, \alpha_{END})}{\max\limits_{k=1:m} D_c(d_{x,k}, L_{END})}$$

$$g_{static}(v_x) = \min g(F_x), g(J_x)$$

ここで， $D_p(v_x, \alpha_{END})$ はタスク $v_x$ の処理能力 $\alpha_{END}$ での処理時間，$D_c(d_{x,k}, L_{END})$ は依存関係 $d_{x,k}$ の帯域幅 $L_{END}$ での通信時間である．このフェーズでは静的な解析を行っているため，処理能力や帯域幅といった指標の動的な値は入手できない．そのため，この静的解析を行う，要求送信ノードの値を代表値としてそれぞれ $\alpha_{END}$ と $L_{END}$ に代入し，算出を行う．$g_{static}(v_x)$ が大きいタスクは、計算に対して通信コストが相対的に小さいため，重複によるネットワーク負荷のペナルティが低い．つまり，重複によって遠隔のノードにスケジュールした場合にも，通信ペナルティの影響を受けずに早く処理結果を返送できる可能性がある．よって，$g_{static}(v_x)$ が小さいタスク $v_x$ は，重複による実行時間短縮の成功可能性が高いと言える．したがって，本手法では $CP$ 上のタスクを $g_{static}(v)$ の降順にソートし，上位のタスクに対して優先的に Budget を配分する．分配方法は，予算 $B_{total}$ を線形に分割する形とする．以上を持って，Phase1 での DAG の静的解析を終了し，実行要求の転送を開始する．

### 3.2 Phase2: 動的情報での重み付けと重複判断

Phase2 では，実行要求を受け取った各ノードが，Phase1 での静的解析の結果割り振られた予算を用いて，動的情報を加味した重複判断を行う．初めに，現在要求が回ってきたタスク，即ち割り当てようとしているタスクが，クリティカルパス上に存在し，重複対象となっているかを判断する．もし重複対象でない場合には，重複割当を行わず，これまでの手法と同様に重複を排除するような一筆書きの割当を続ける．もし重複対象である場合には，動的な情報を用いた重複判断に移る．

ノードは要求パケットを受信した際、ヘッダに記録された経路統計情報を用いて、動的粒度 $g_{dynamic}(v_x)$ を算出する．

$$g(J_x) = \frac{\min\limits_{k=1:m} D_p(v_x, \alpha_{avg})}{\max\limits_{k=1:m} D_c(d_{k,x}, L_{avg})}$$

$$g(F_x) = \frac{\min\limits_{k=1:m} D_p(v_x, \alpha_{avg})}{\max\limits_{k=1:m} D_c(d_{x,k}, L_{avg})}$$

$$g_{dynamic}(v_x) = \min g(F_x), g(J_x)$$

ここで $g_{dynamic}(v_x)$ を算出するための平均処理能力 $\alpha_{avg}$ と平均帯域幅 $L_{avg}$ は，経路統計情報から以下のように算出できる．

$$L_{avg} = \frac{\sum_{i=0}^{|P(p_i)|} L_{i,i+1}}{|P(p_i)| - 1}$$

$$\alpha_{avg} = \frac{\sum_{i=0}^{|P(p_i)|} \alpha_i}{|P(p_i)|}$$

ここで，$P(p_i)$ は要求パケット $p_i$ が通ってきた経路 $P(p_i) = (n_0 = A(v_{END}), n_1, n_2, ..., n_n)$ を表しており，これまで経由してきた経路上の統計情報に対して平均値を算出する．各経由情報は Interest パケットヘッダ内の Application Parameters フィールドに格納されり．この挙動により，In-band network telemetry に似た挙動と情報を ICN-SFC でも実現できる．静的解析では初期値として静的解析を実行したノードの値を代表値として，通信環境を仮定していたが，実際のネットワークでは混雑状況により有効帯域が変化する．本手法では，この $g_{dynamic}(v_x)$ を用いてタスクの優先順位をローカルで再評価する．そして，各重複対象タスクに対する予算の分配状況を，再評価によって入れ替える．

具体的には，通信コストが想定以上に増大している場合，そのタスクへの重複予算の割り当てを取り消し，下位のタスクあるいは次点の候補と予算を入れ替えることになる．これにより，ネットワークの環境に適応した重複判断と重複制御を実現できる．最後に，経路統計情報に基づいた予算の再評価・再分配を受けて，現在割り当てようとしているタスクに重複

のための予算が残っているかを判断する．ここで予算が残っている場合には，このタスクを 1 回だけ重複し，そのタスクを基準とした実行パス上にあるタスクの予算を 1 減らす．以上が，動的情報を加味した重複判断である．

図 4 に，本提案手法を用いて重複制御を行った例を示す．要求前の静的解析と，転送中の動的情報を加味した重複判断により，実行時間短縮に対する必要性の高いタスクのみが重複スケジュールされる．

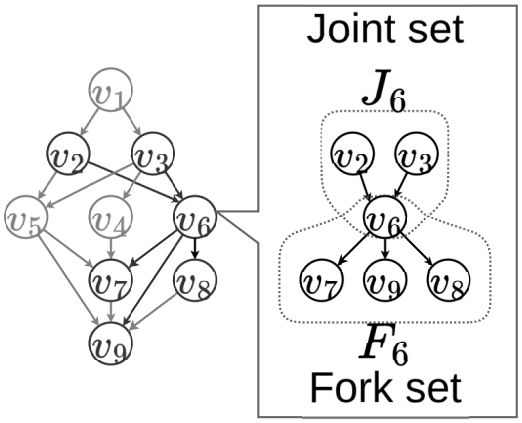


図 3　Grain 算出のための Joint/Fork set の例

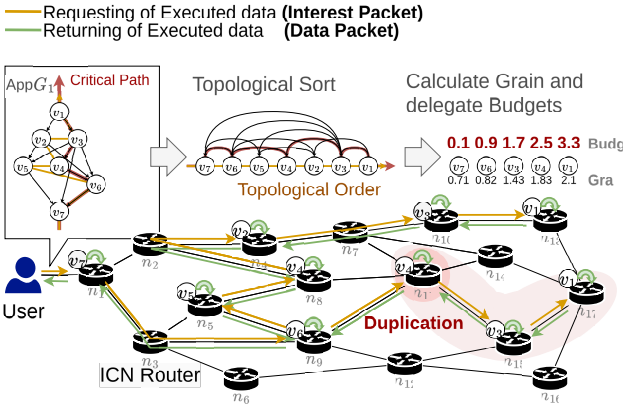


図 4　DAG 構造と粒度に基づいて重複を行う例

## 4. 性能評価

### 4.1 実験環境

提案手法の性能と特性を明らかにするため，計算機シミュレーションを用いて評価を行う．ここでは，従来手法である AutoICN [10] の環境に基づいたワークフローシミュレータである，icn-sfcsim [15] を用いた．表 1 に，評価環境のパラメタを示す．ICN Node がアプリケーションに対する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ノードでもある．ICN Router はその間に介在するネットワークノードであり，INC における処理を担うノードである．

本シミュレーションでは，ランダムに生成した DAG をアプリケーションとして用いる．投入するアプリケーションのパラメタを表 2 に示す．各アプリケーション内のタスク数を 20 個とし，各タスクが持つ後続タスク数を最大 5 個で設定している．ランダム DAG による実験においては，より多くのアプリケーションを想定し，それらに対する汎用的な性能を評価するためにアプリケーションの CCR (Communication to Computation Ratio) を変化させることが考えられる．CCR はアプリケーション全体の通信と処理の比率であり，アプリケーションを特徴づける指標の 1 つである．そのため，本シミュレーションにおいても CCR を 0.1 から 10.0 まで変化させた計 11 パターンを評価している．

比較対象としては，NFJ 問題に対する制御を全く行わない手法 (従来手法，Basic Method)，トポロジカルソートによって重複のないように割り当てる手法 (以前の取り組み，Advanced method1)，そして必要なタスクのみ重複させる手法 (提案手法，Advanced method2) の 3 つである．これらの手法について，アプリケーションの実行時間，タスクの総割り当て数を比較する．

表 1　実験環境のパラメタ

| Parameter | Value |
|---|---|
| ICN Node # | 100 |
| ICN Router # | 5000 |
| Proc. Rate (MIPS) | 2000 - 4000 |
| Link Bandwidth (Mbps) | 100 - 1000 |

表 2　アプリケーションのパラメタ

| Parameter | Value |
|---|---|
| # of tasks for each Apl. | 20 |
| # of Apl. | 1 |
| # of successors for each task | 0 - 5 |
| CCR of each Apl | (0.1, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0) |

### 4.2 実験結果

図 5 に，各手法における平均の割り当て回数についてのグラフを示す．割当回数が実質的に計算資源の消費量と言える．従来手法は FNJ 問題に対して一切制御を行わないため，DAG の持つ Fork-Join 構造に基づいてタスクの重複割当が大量に生じている．以前の手法である重複排除の制御手法については，常にアプリケーション内に含まれているタスク数だけ割り当てるため，割当回数が最も小さい．最後に，提案手法については，従来手法と以前の手法の中間となるような割当回数を記録している．提案手法の全体の最大重複数を制限するアプローチにより，すべてのタスクを無差別に重複させるのではなく，必要なタスクのみを重複させていることがわかる．

図 6 に，アプリケーションの実行時間を示す．従来手法は FNJ 問題の制御を行わず大量の重複を発生させることで，くまなく周辺ノードへ要求を行い，結果としてその中からよい実行性能を得られるノードで実行することに成功したものが現れる．これにより，他の手法と比較してかなり短い実行時間を得ることができる．以前の手法は，最小の計算資源消費量でアプリケーションを実行させるため，従来手法と対象的に実行時間は大きくなっている．これらに対し，提案手法は従来手法と以前の手法の中間となるような実行性能となった．これらのことから，提案手法は重複に対して制御を行わない場合と，重複のない割り当てを行う場合の中間のような性能を実現していることがわかった．つまり，本手法は無制限な重複を許容する従来手法よりも計算資源消費を抑制しつつ，重複を完全に排除する以前の手法よりも高い実行性能を実現することができており，計算資源消費量と実行性能のトレードオフの

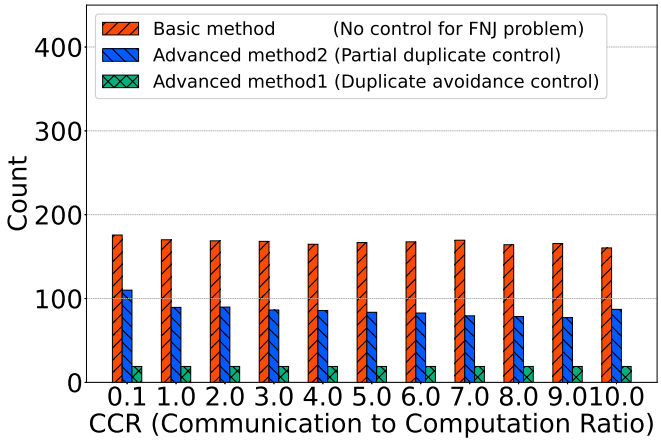


図 5　各手法におけるタスクの総割当回数

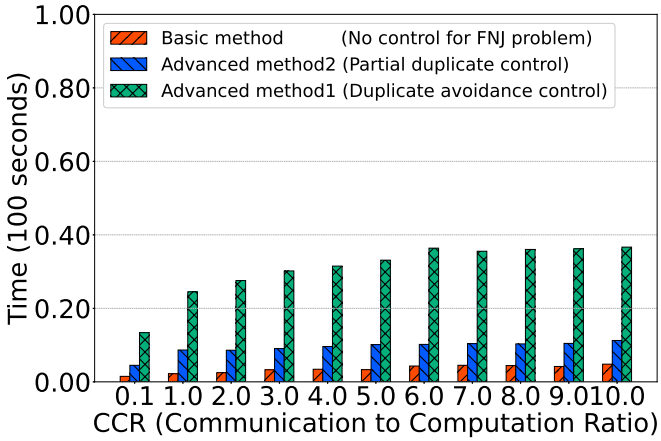


図 6　各手法におけるアプリケーション実行時間

関係においてバランスのよい割当を実現できていると言える．

## 5. まとめと今後の展望

本稿では，自律的な In-network computing における計算資源消費量と実行時間のトレードオフの関係の中で，DAG の構造とタスクの粒度に基づき必要なタスクのみを重複させることで，バランスのよい割当を実現するタスクスケジューリング法を提案した．

今後はより大規模・より動的なネットワークへの対応として，動的指標での重み付けをより高度化していくことが考えられる．現在はシンプルな統計値をそのまま利用しているが，統計値を元に周囲のネットワーク状況を推測するようなファンクションを各ノードが持つことにより，より精度の高いタスクの順序付け・重複判断ができるようになると考えられる．このファンクションには，機械学習や LLM といった AI 的手法を使うことが挙げられる．そのため，今後の展望としては，動的指標の活用方法を高度化させていくことにより，幅広いシチュエーションに適用できるよう手法を拡張していきたい．

**謝辞**　本稿は JSPS 科研費（25K03113, 23K28078）の助成を受けたものである．ここに記して謝意を表す．

## 文　献

[1] Mohsen Marjani, Fariza Nasaruddin, Abdullah Gani, Ahmad Karim, Ibrahim Abaker Targio Hashem, Aisha Siddiqa, and Ibrar Yaqoob. Big iot data analytics: Architecture, opportunities, and open research challenges. *IEEE Access*, Vol. 5, pp. 5247–5261, 2017.

[2] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P. Sheth. Machine learning for internet of things data analysis: a survey. *Digital Communications and Networks*, Vol. 4, No. 3, pp. 161–175, 2018.

[3] Michael Grieves and John Vickers. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. In Franz-Josef Kahlen, Shannon Flumerfelt, and Anabela Alves, editors, *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*, pp. 85–113. Springer International Publishing.

[4] Fei Tao, Qinglin Qi, Lihui Wang, and A.Y.C. Nee. Digital twins and cyber − physical systems toward smart manufacturing and industry 4.0: Correlation and comparison. *Engineering*, Vol. 5, No. 4, pp. 653–661, 2019.

[5] Mario Gerla, Eun-Kyu Lee, Giovanni Pau, and Uichin Lee. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pp. 241–246.

[6] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B. Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys Tutorials*, Vol. 19, No. 4, pp. 2322–2358, 2017.

[7] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pp. 13–16. Association for Computing Machinery.

[8] Shadi Al-Sarawi, Mohammed Anbar, Rosni Abdullah, and Ahmad B. Al Hawari. Internet of Things Market Analysis Forecasts, 2020 − 2030. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pp. 449–453.

[9] Lei Liu, Yang Peng, Mehdi Bahrami, Liguang Xie, Akira Ito, Sevak Mnatsakanyan, Gang Qu, Zilong Ye, and Huiping Guo. ICN-FC: An Information-Centric Networking based framework for efficient functional chaining. In *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7.

[10] 金光永煥, 花田真樹. Icn ワークフローにおける自律的なタスク割当てアルゴリズムの性能解析. 電子情報通信学会技術研究報告, Vol. 125, No. 13, pp. CS2025–3, 13–18, 2025 年 4 月.

[11] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, Vol. 29, No. 7, pp. 1645–1660, 2013. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond.

[12] Yuki Niibe and Noriaki Kamiyama. Task scheduling with duplication avoidance for icn-based autonomous in-network computing. In *2025 IEEE 31st International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pp. 1–2, 2025.

[13] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, Vol. 46, No. 5, p. 720 − 748, September 1999.

[14] A. Gerasoulis and T. Yang. On the granularity and clustering of directed acyclic task graphs. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 6, pp. 686–701, 1993.

[15] Kanemitsu Lab. ncl-teu/ncl_icn-sfcsim. https://github.com/ncl-teu/ncl_icn-sfcsim, 2025. Accessed: 2026-01-17.