

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

立命館大学

新部裕樹 上山憲昭

目次

- 研究背景
- 研究目的
- 先行研究
- 各条件の整理
- 提案手法
- 実験
- 評価
- まとめ

研究背景

5Gネットワーク：高速大容量・低遅延・多数接続



多くの“モノ”がネットワークに接続される
(IoTデバイスの普及, モノ・コトの情報化)



新たなアプリケーション / ユースケース
多種多様な処理を実行させることへの需要増加

- 例：スマートシティに配備されたIoTデバイス
 - IoTデバイスからセンシングされる膨大なデータを連携しながら分析 → ビッグデータの活用
 - 新たなサービス・ビジネスの創出

研究背景

- 多種多様な処理の実行
 - IoTデバイスからセンシングされるデータ
 - これらのデータは遠隔地のクラウド上で処理される

IoTデバイス：ネットワークのエッジ

データの収集



クラウド：ネットワークのコア

データの処理

- 効率性に問題
- ネットワーク
トラヒックの圧迫

➡ In-Network Computing¹の適用

ネットワークのエッジからコアに向かってデータを転送させていく中で、処理も同時に行う

研究背景

- In-Network Computing

- IoTデバイスが加速的に普及・増加する見込み
 - 2030年には2020年比で約10倍に⁵

- これまでは限られた範囲内での連携処理を想定
 - 限られた数のデバイスからデータを取得し処理する



- 広域に分布したデバイス・デバイス上のデータを用いた連携処理
 - 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処理する

➡中央ノードがタスクの実行を管理することは困難

- 中央ノードが全てのデバイスを把握し、タスクの実行場所を管理(スケジューリング)できない

研究背景

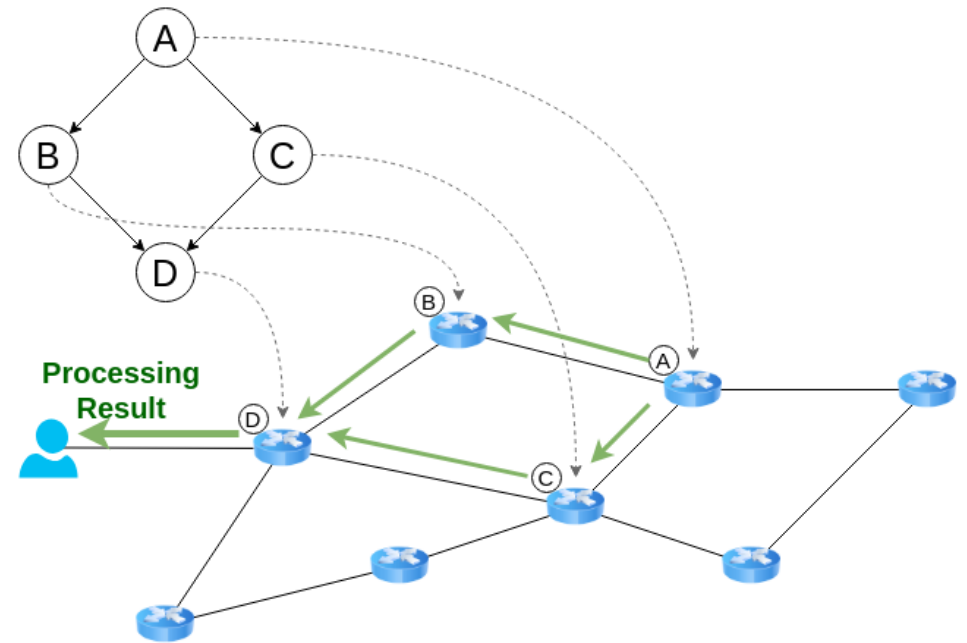
- 広域に分布した情報・計算資源を用いた処理連携を行うために
 - **自律的なIn-Network Computing**が必要
 - 多種多様な処理の実行を許容
 - 中央集権的な管理ノードが不要

目的

- 自律的なIn-Network Computing
 - 自律的であることによる制約条件を整理
 - 制約条件を満たすための方法を提案
 - 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提案・評価

In-Network Computing

- タスクスケジューリング
 - アプリケーションの依存関係が満たされるようにスケジュール
 - 各タスクの処理結果が後続タスクへ受け渡される
 - ➡先頭から後続へ順にタスクが実行され、最終的な処理結果がユーザに届く



先行研究

- Fog Computing
 - エッジデバイスとクラウド間に介在するネットワーク上でコンピューティングや各種サービスを提供する方法
 - エッジとクラウドの間に新たなレイヤを構築する
 - クラウドよりもエッジに近い場所で処理を行う
 - 遅延の削減・リアルタイム性の高いアプリケーションへの対応
 - Fog(霧)のように計算リソースが地理的に分散
 - 広範囲のネットワークに対応・膨大なIoTデバイスの接続を許容

先行研究

- Fog Computing
 - Fog Computingにおけるタスクスケジューリング
 - いくつかの研究が存在
 - Fog Computingのレイヤにスケジューラを用意し, 集中的にタスクスケジューリングを行う
 - ➡ 自律的なスケジューリングは考慮されていない
 - エッジレイヤはスケジューリングの対象外
 - ネットワーク全体をスケジュール対象にできない

制約条件の整理

自律的なIn-Network Computingのために考慮しなければならない
制約条件

制約条件の整理

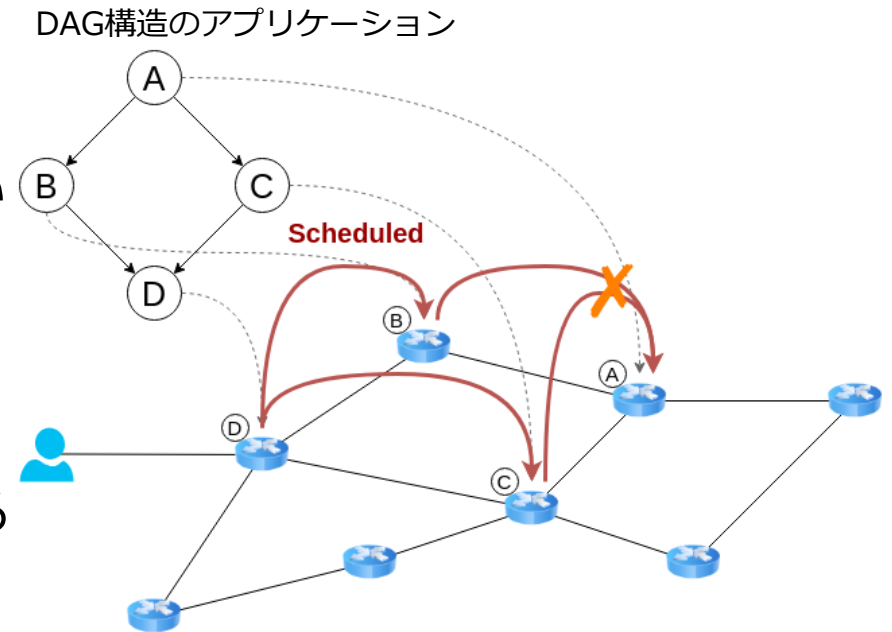
- 自律的なIn-Network Computing : 制約条件1
 - 中央集権的なスケジューラが存在しない
 - ネットワーク全体の状況を把握して管理するノードが存在しない
 - ➡各ノードが自律的にタスクをスケジュールする必要がある

制約条件の整理

- 自律的なIn-Network Computing : 制約条件2
 - 各ノードで行われた判断は必ずしも一致しない

- 自律的にタスクをスケジュールする場合、同一のタスクであってもノードによっては異なる場所へスケジュールするかもしれない

➡各タスクが依存関係を満たした状態で実行場所が必ず1つに決まることを保証できる仕組みが必要



制約条件の整理

- 自律的なIn-Network Computing : 制約条件3
 - 各ノードはネットワーク全体の情報を知り得ない
 - 直接的に接続されている隣接ノードの情報は比較的容易に取得できる
 - 一方、隣接ノードより先の情報の取得は高コスト
 - マルチホップ / ネットワーク全体へのフラッディングによる問い合わせ
 - タスクの割当先を決定する際に、全ノードの中から最適な場所を探すことができない
 - ➡隣接ノードからの情報のみを使って、なるべく全体最適な割当を行う必要性

提案手法

制約条件を満たす最もベーシックなアルゴリズム

提案手法

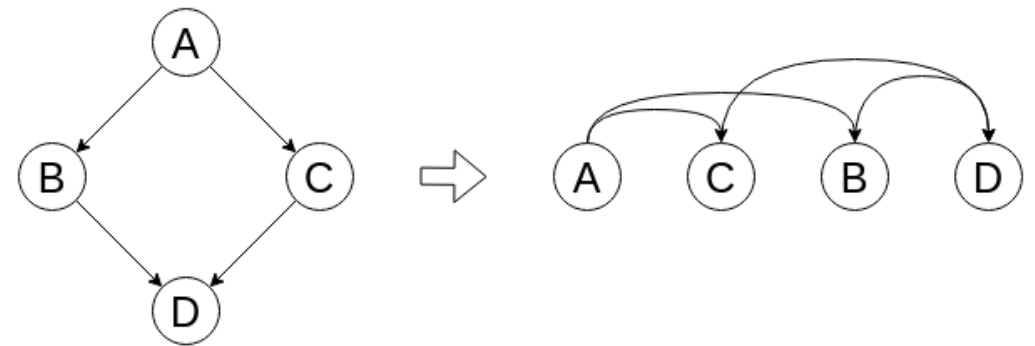
- 制約条件1：中央集権的なスケジューラが存在しない
 - 後続タスクが先行タスクをスケジュールする
 - 最後尾から先頭に向かって繰り返す
 - ➡アプリケーション内の全てのタスクをスケジュールできる

提案手法

- 制約条件2：各ノードで行われた判断は必ずしも一致しない

- DAGをトポロジカルソートする

- 依存関係を満たした状態でDAGを一直線に並び替える



- タスク割り当ての操作が並列して行われなくなる
- スケジュールするタスクとスケジュールされるタスクが1対1で対応

➡各タスクが依存関係を満たし、必ず1つの実行場所に決まることを保証できる

提案手法

- 制約条件3：各ノードはネットワーク全体の情報を知り得ない
 - 隣接ノードの情報のみを使って実行場所を決定する
 - タスクをスケジュールする際に、各ノードは自身と隣接ノードのどちらで実行すべきか判断する
 - 自身の場合は自ノードにスケジュール
 - 隣接ノードの場合はタスクのスケジュール要求を送信
 - タスクがスケジュールされるまでスケジュール要求の送信を繰り返す
 - ➡各ノードは限定的な情報だけを使って実行場所としてよい場所を発見できる

提案手法

- まとめ

1. 後続タスクが先行タスクをスケジュールする
2. DAGをトポロジカルソートする (一直線にする)
3. よい実行場所が見つかるまでスケジュール要求の転送を繰り返す

- 自律的なIn-Network Computingが実現できる

ベーシックなアルゴリズム

各制約条件を満たした, ベーシックなアルゴリズムを提案

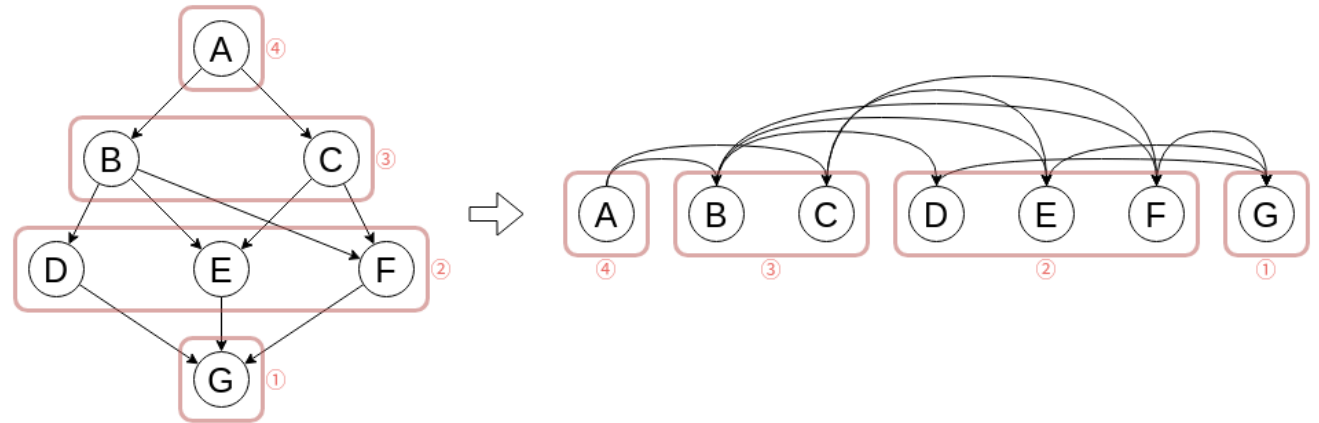
提案手法：ベーシックなアルゴリズム

- タスクの並び替えの方法

- トポロジカルソート

- ReadyListを用いて実現
 - 後続タスクがすべてソート済のタスク集合

- おおまかな順序が決定

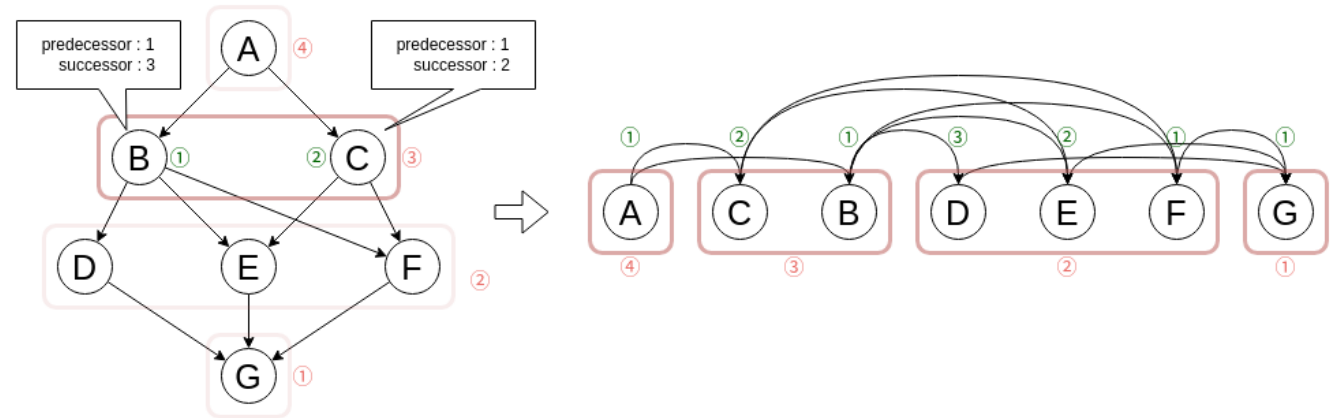


提案手法：ベーシックなアルゴリズム

- タスクの並び替えの方法

- 細かな優先順位付け

- 同時にReadyListに入る
タスクがある



- 後続タスク数と先行タスク数を用いて優先順位付け

- 先行タスク数が多い：優先度 低
 - 後続タスク数が多い：優先度 高

$$priority(v_i) = \frac{num(suc(v_i))}{num(pred(v_i))}$$

提案手法：ベーシックなアルゴリズム

- タスク割り当ての方法
 - よい実行場所が見つかるまでスケジュール要求の転送を繰り返す
 - 実行場所としてよい場所かどうか, どのように判断する?
 - Blevelを用いる
 - そのタスクからENDタスクまでの残り時間 (処理 + 通信にかかる時間)
 - Blevelが小さくなる場所をよい実行場所として決定する
 - 自ノードと隣接ノードのどちらがBlevelが小さくなるか?

提案手法：ベーシックなアルゴリズム

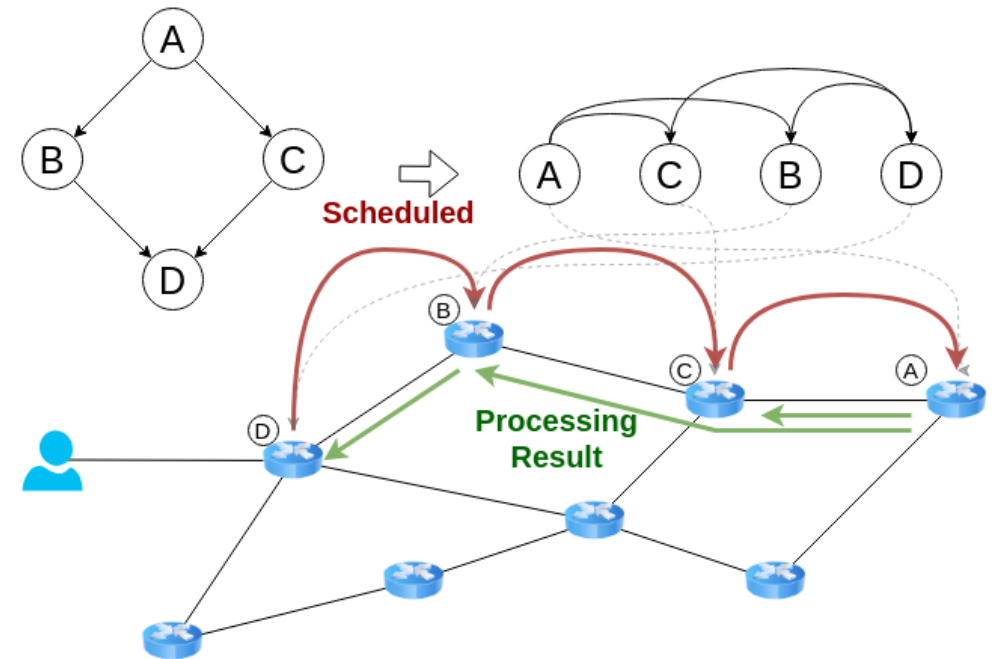
- まとめ

- タスクの並び替え方法

- ReadyListでのトポロジカルソート
+ 先行・後続タスク数での順序付け

- タスクの割り当て方法

- Blevelを使って、よい実行場所が見つかるまで転送を繰り返し、割り当て



評価実験

- 自律的なIn-Network Computingの可能性を調べる
 - ベーシックなアルゴリズムがどの程度の性能になるのか評価

タスクの並び替えについて性能への影響度を評価

- **条件1 :**

- ベーシックなアルゴリズム
 - 並び替え (トポロジカルソート + 先行・後続タスク数)
 - 割り当て (Blevel)

- **条件2 :**

- ベーシックなアルゴリズムから詳細な並び替えを削除 (おおまかな並び替えのみ)
 - 並び替え (トポロジカルソートのみ)
 - 割り当て (Blevel)

評価実験

- 自律的なIn-Network Computingの可能性を調べる
 - ベーシックなアルゴリズムがどの程度の性能になるのか評価

達成可能な性能上限値と比較しどの程度性能劣化するか？

- **条件3**：集中管理でのスケジューリングにおける最先端のアルゴリズム：**HEFT**
 - 自律的なIn-Network Computingでは実現できない方法
 - 達成可能な性能の上限値として、比較対象とする
- タスクの割り当て・並び替えの影響度を明らかにしたうえで、最先端のアルゴリズムと比較し自律的なIn-Network Computingにどの程度可能性があるか探る

評価実験

- 実験環境

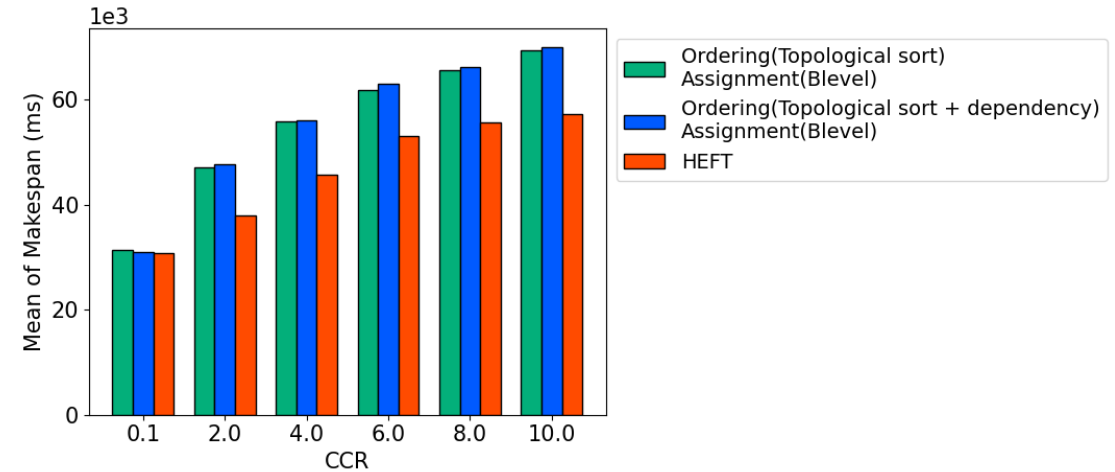
- icn-sfcsim シミュレータで評価実験
 - ICN (情報指向ネットワーク) を用いたIn-Network Computingのシミュレータ
- ネットワークトポロジ
 - エッジデバイス : 100台
 - ルータ : 500台
 - ランダムネットワーク
- 投入アプリケーション
 - タスク数 : 25
 - CCR (通信のウェイトと処理のウェイト) : 0.1 ~ 10.0まで変化させる
 - ランダムアプリケーション

実験結果

- 条件3(最先端のアルゴリズム)とは大きな差

- 最先端のアルゴリズムで集中的にスケジュールした方が常に性能が良い

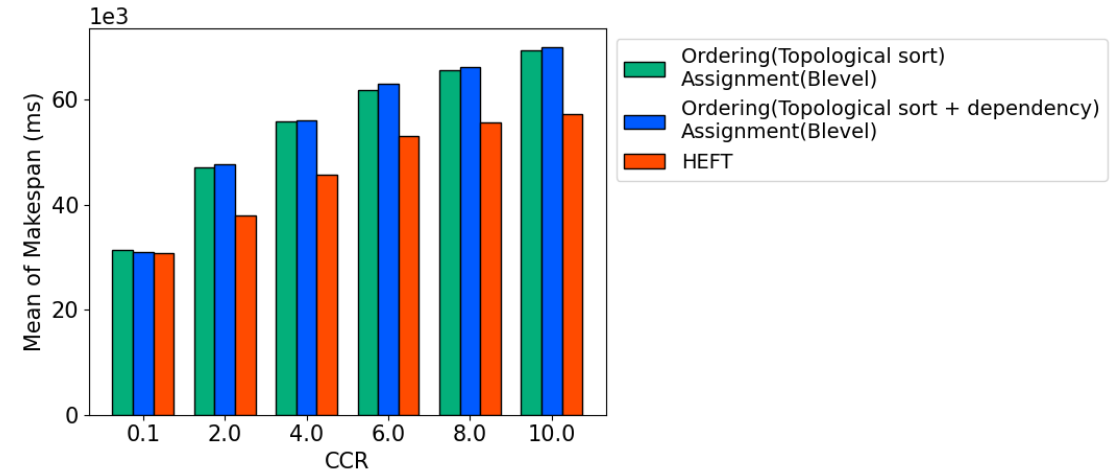
- しかし, 提案手法(自律的)が分散処理として意味がない程性能劣化しているわけではない



- 自律的なIn-Network Computingにおける制約条件がある中では, 十分に有効な範囲に性能劣化が留められていると考える

実験結果

- 条件1と条件2に差がない
 - 詳細な並び替えを行うかどうかは性能に影響しない
 - トポロジカルソートによっておおまかな順序のみ守られていけばよい



- 順序を詳細に決定したところで、必ずその順序で実行されるわけではないため
- 性能向上のために、今後は割り当て先の決定方法を検討する必要がある

まとめ

- 自律的なIn-Network Computingを実現する方法を示した
- 最もベーシックなアルゴリズムに対する性能評価を実施
 - 集中管理の最先端アルゴリズムからの性能劣化は有効な範囲に留まる
 - タスクを詳細に並び替えることはあまり性能へ寄与しない
 - 今後はタスクの割り当て先を決定する手法について検討
- ベーシックなアルゴリズムを元に改良することで、性能向上が期待できる

参考文献

1. 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), New York, NY, USA, 2012, pp. 13 - 16
2. Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. "Networking named content." In Proceedings of the 5th international conference on Emerging networking experiments and technologies (CoNEXT '09). Association for Computing Machinery, New York, NY, USA, 2009, pp. 1 - 12.
3. Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Be- ichuan Zhang. 2014. Named data networking.
4. 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
5. S. Al-Sarawi, M. Anbar, R. Abdullah and A. B. Al Hawari, "Internet of Things Market Analysis Forecasts, 2020 - 2030," 2020 FourthWorld, Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), London, UK, 2020, pp. 449-453.