NFaaS: Named Function as a Service

Michał Król, Ioannis Psaras

The 4th ACM Conference on Information-Centric Networking, September 2017

rgroot 輪講

2022/05/09 kekeho

Abstract

提案内容

- これまで, Information-Centric Networking(**ICN**) コミュニティは, コンテンツ配信にフォーカスしてきた
- Edge/Fog Computing環境のサポートもしたい
- ネットワーク内での関数実行をサポートするフレームワーク**NFaaS**を提案
  - Named Data Networking (**NDN**)を拡張

Abstract

NFaaSの特徴

- ネットワーク上のどのノードからでも関数をダウンロードして実行できる
- 関数はユーザーの要求に応じてノード間を移動
- NFaaSは, ネットワーク内で関数をデプロイし, 動的にマイグレートするためのルーティング・プロトコルと, 転送戦略を含んでいる
- シミュレーションをして, delay-sensitiveな関数はedgeに, そうでないものはコア寄りに配置されることを検証した

前提知識

ICN

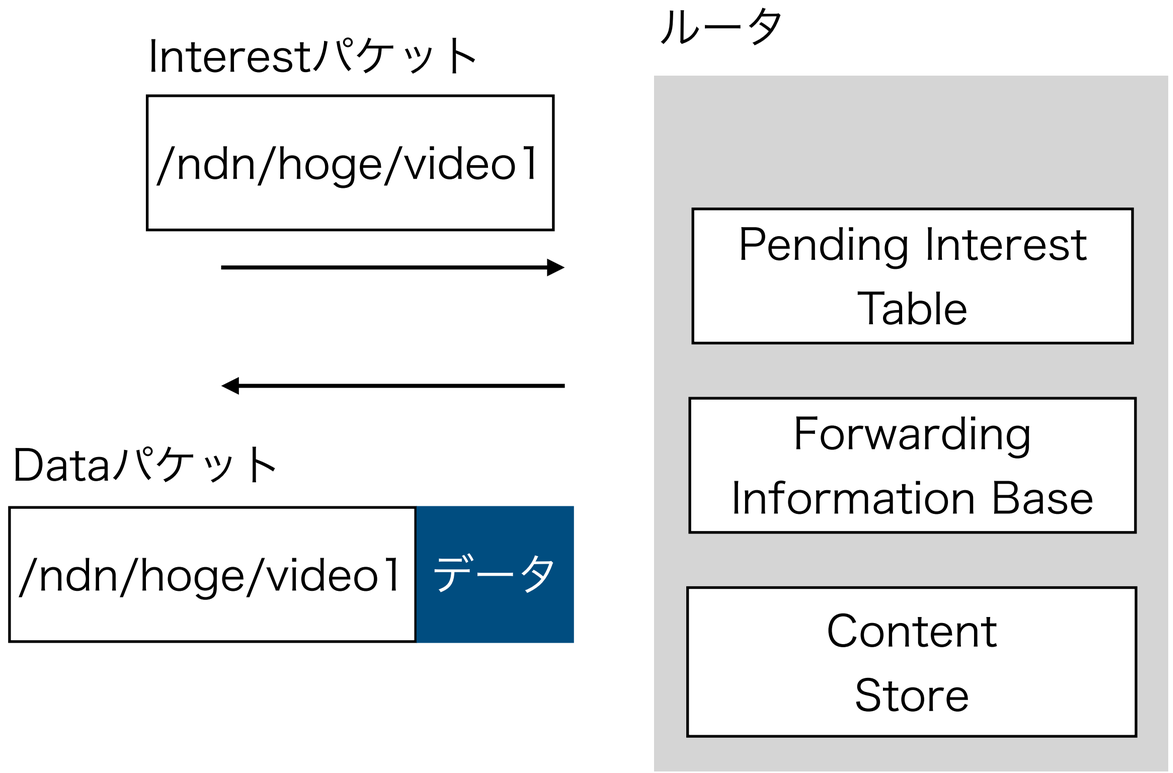
- 現在のインターネット: 通信相手のサーバーを指定して情報を取得
- ICN: 情報指向ネットワーキング
  - 欲しい情報そのものを示す識別子を利用してネットワークから情報を取ってくるパラダイム
    - 同じデータはどこから取得しても一緒
    - 考え方はIPFSと似ている

前提知識

NDN

- NDN: Named Data Networking
  - 米国のICNプロジェクト. リファレンス実装などを出している.
- Interestパケットでリクエストする
- FIB: 転送先Faceを管理する. 経路制御表.
- PIT: Interestを受信したFaceを記録するテーブル
- CS: コンテンツをキャッシュしておく領域

(Faceはコンテンツ送受信を行う論理的なインターフェイス)



1. Introduction

背景と目的

- Edge Computingしたい
  - ARやIoTの普及で, 低遅延, Edgeで処理するニーズが高まっている
- Edgeノードへのアプリケーションの動的配置など, 多くの課題がある
- ICNの原理を用いれば, 上記の課題を解決できそう
- あらゆるサービスが, ネットワーク上のあらゆるノードで動作し, ユーザーのーに基づいて最適なノードに移動し, 需要に応じてインスタンスを複製できるようにしたい

1. Introduction

先行研究

- Named Function Networking [1]
  - λ計算をICNでやる
  - λ関数だけでは表現が難しい

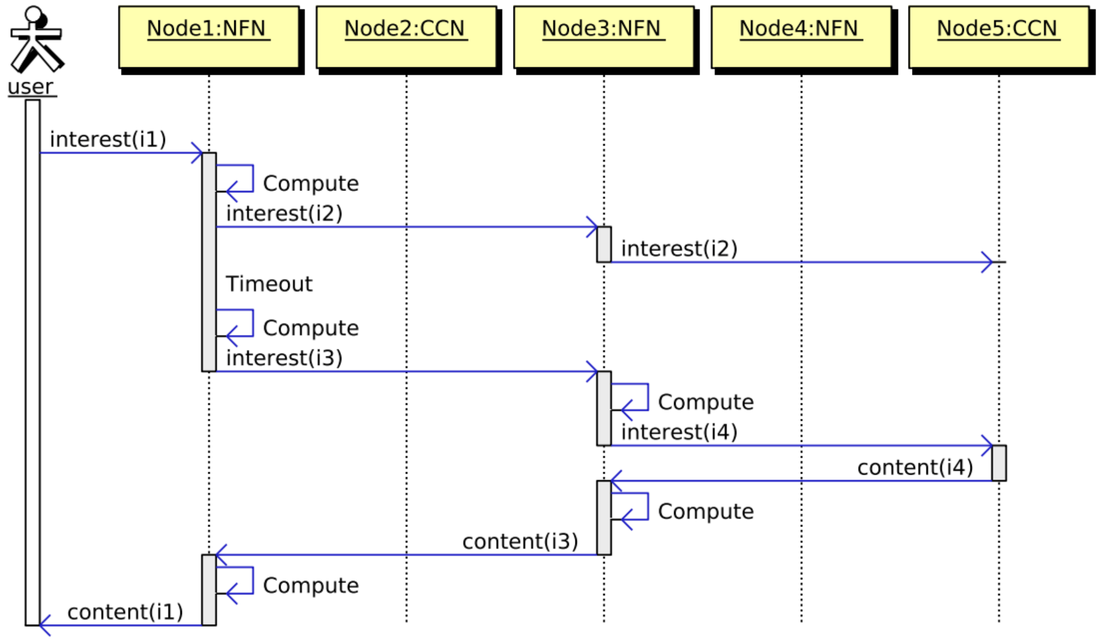


Figure 5: Experiment 2 – computation push (the network works around a CCN-only node)

i_1`{/bin/scala/wrdcnt (/node5/doc)}`
i_2`{/bin/scala/wrdcnt (/node5/doc)}`
i_3`{(λx.(x /node5/doc)) (/bin/scala/wrdcnt)}`
i_4`{/node5/doc}`

[1]より引用

1. Introduction

Unikernel

- 本研究では, **Unikernel** [2]を使う
  - 単一のアプリのみをベアメタルで動かす軽量イメージ
  - アプリが使う最低限必要なものしかリンクしない
  - 汎用OSのレイヤを省けるので軽量

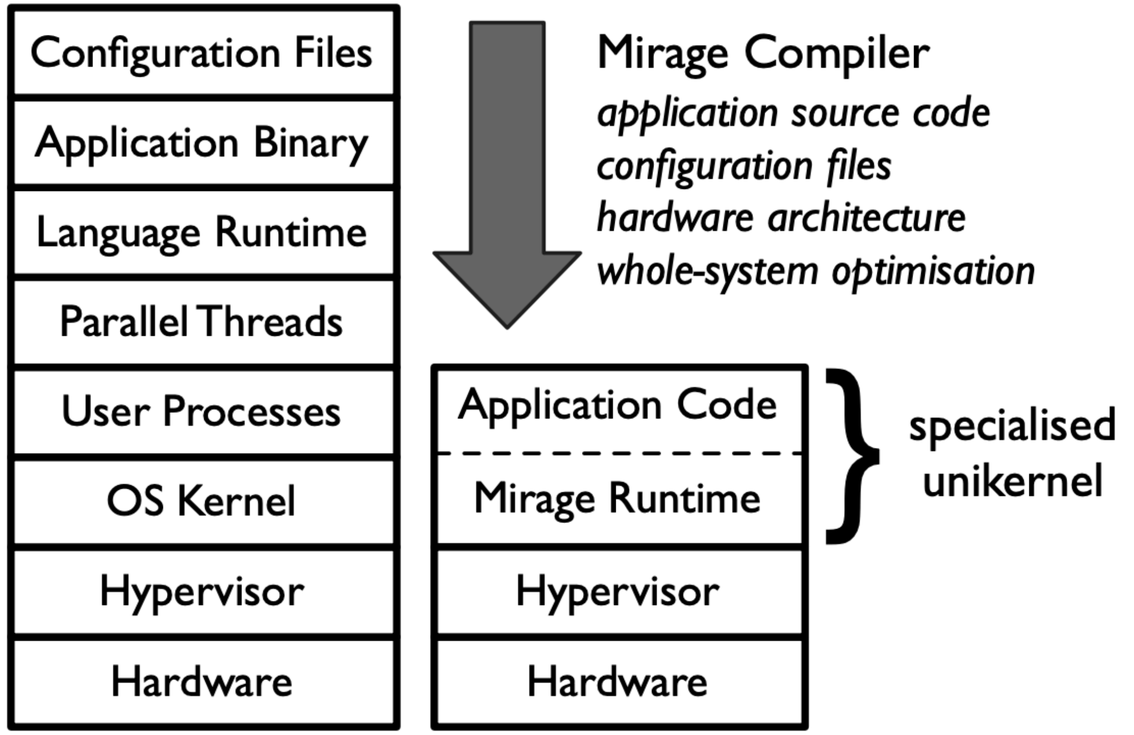


Figure 1: Contrasting software layers in existing VM appliances *vs.* unikernel's standalone kernel compilation approach.

[2]より引用

1. Introduction

NFaaSの紹介

- Interestパケットを介して名前付き関数の実行を要求
  - 関数のinput(引数?)もInterestパケットに含める
- ノードはContents Storeの他に, Unikernelをキャッシュするストレージ(**Kernel Storage**)と, それらを実行する計算キャパシティを持っている
- Delay-sensitiveな関数と, Bandwidth-hungryな関数の二種類が存在
  - 前者は低遅延を要求しているので, なるべくEdge寄りで動かしたい
- NDNスタックの拡張としてNFaaSを実装

2. System Design

概観

- 人気になるほどユーザーの近くに関数を持ってくる
- 計算キャパシティを超えたら, Interestを周りのノードに転送する

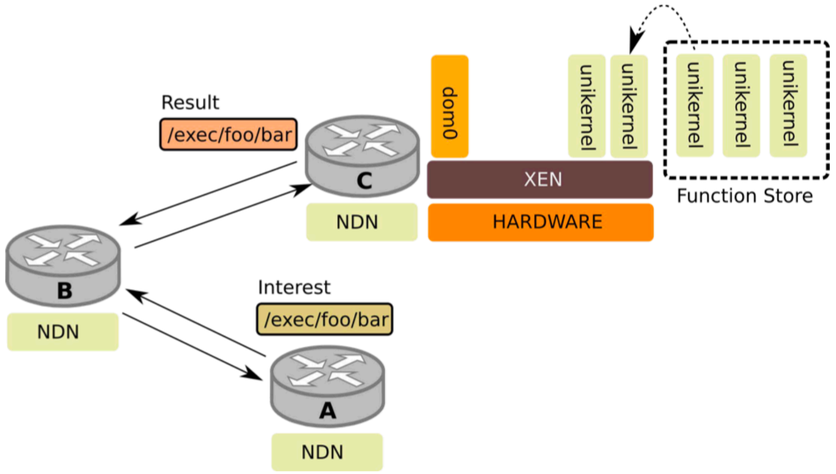
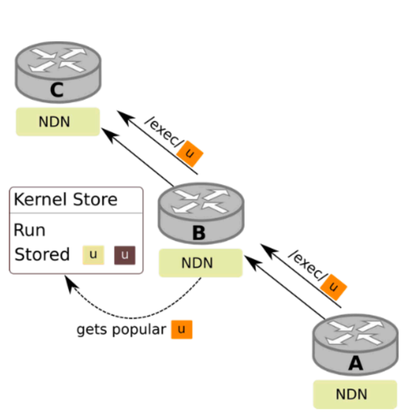
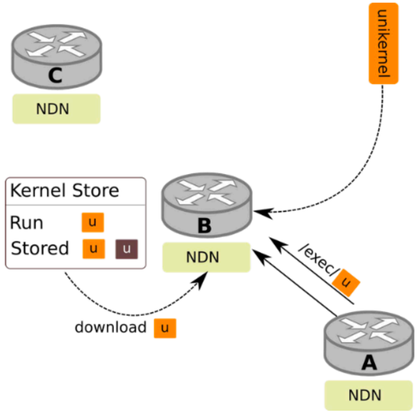


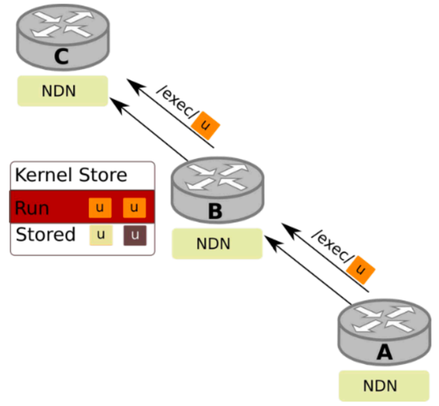
Figure 2: System architecture.



(a) A kernel becomes popular.



(b) Kenel Store requests to download the kernel.



(c) When overloaded, Node B forwards a part of requests towards the cloud.

Figure 3: NFaaSHigh-Level Overview

2. System Design

Naming Moving Functions

- /exec接頭辞を導入
  - deley-sensitiveな関数: /exec/delay
  - bandwidth-hungryな関数: /exec/bandwidth
- 接頭辞のあとに関数名
- 関数名のあとに関数へのInputを添える
- 付加情報
  - タスクのデッドライン
  - 並列実行を防ぐためのDiscoverフィールド(?)

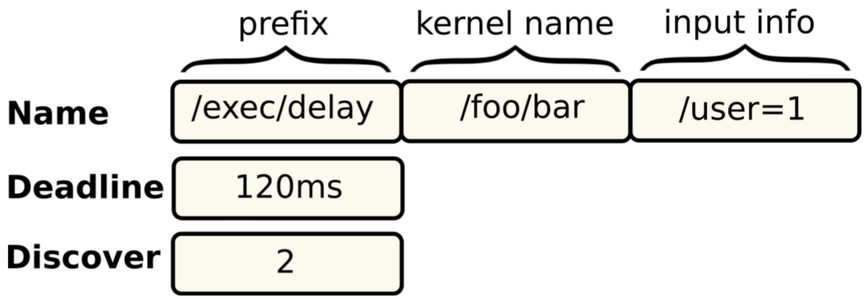


Figure 4: Interest packet structure

2. System Design

Storing and Executing Moving Functions

- **Measurement Table**: どのUnikernelをキャッシュし, 実行するか決めるための, 過去の統計情報
- 保存する情報
  - Deadline: タスクのデッドライン
  - Function Popularity: 直近のi Interestのうち, リクエストされた割合
  - Average Hop Count: リクエストしてきたクライアントの平均ホップ数
  - Preferred Face: Interestを転送する際の宛先Faceのリスト
  - Average Service Delay: 転送したInterestの結果が帰ってくるまでの時間?

Table 1: Measurement Table entry

Function Name	/delay/foo/bar/			
Deadline	120ms			
Popularity	2/10	7/10	4/10	3/10
Hop Count	2.43			
Faces	netdev1		netdev2	
Delay	94ms		86ms	

2. System Design

Resolving Moving Functions

- Measurement Tableに基づいて, 観測されたすべてのUnikernelについて**Unikernel Score**を計算する

新しいリクエストほど重み付け

$$unikernel\ score = \sum_{i=0}^{m} \frac{p_i}{n} \cdot (m - i) + (R - h_i) \cdot t_m$$

人気度

delay-sensitiveな関数の場合, ホップ数を考える

- Unikernel scoreが大きい順にカーネルをストアしようとする

  - m: ノードは, n個のパケットのうち直近mエポックの記録を残す
  - p_i : Function Popularity. 直近n回のリクエストで, このUnikernelに対してこのノードを通過したInterestの数
  - h_i : Average Hop Count
  - t_m $(0 \text{ or } 1)$: delay-sensitive(1)かbandwidth-hungry(0)か
  - R : ノードの半径(ホップ数)を示すシステムパラメータ. Rの内側にdelay-sensitiveな関数を配置したい. (**Scoped Flooding**)

2. System Design

Resolving Moving Functions

- InterestのルーティングはNDNのルーティング・プロトコルに沿っている

2. System Design

Forwarding Strategies

- ノードがローカルでUnikernelを実行できないとき, 転送戦略に沿ってInterestパケットが転送される
- Deley-sensitive/Bandwidth-hungryそれぞれの転送戦略がある

2. System Design

Forwarding Strategie for Delay-sensitive

- sendDiscoveryPackets:
  近隣のどこに対応するUnikernelが存在するか, デッドラインは満たせそうか探すためにdiscovery interest packetを送信する
  - Discovery interestを受信したノードは, 関数は実行せず, ダミーdata packetで応答する
  - この結果はMeasurement Tableに保持される
- これによりScoped Floodingを実現

```
Data: Interest packet
Result: Output face
outFace = null;
while face = prefferedFaces.hasNext() do
    if !face.isOverloaded() then
        score = loadBalancer.calculateScore(face) if
         score > maxScore then
            maxScore = score;
            outFace = face;
        end
    end
end
if outFace == null then
    sendDiscoveryPackets(interest);
    outFace = FIB.getCloudFace();
end
return outFace
```

Algorithm 1: Delay constrained forwarding strategy

2. System Design

Forwarding Strategy for Bandwidth-hungry

- 要求ノードの近くにいる必要はないので, delay-sensitiveのときみたいにdiscovery mechanism は必要ない

```
Data: Interest packet
Result: Output face
outFace = null;
while face = prefferedFaces.hasNext() do
    if !face.isOverloaded() then
        score = loadBalancer.calculateScore(face) if
         score > maxScore then
            maxScore = score;
            outFace = face;
        end
    end
end
if outFace == null then
    FIB.getPrefixFace();
end
if outFace == null then
    FIB.getCloudFace();
end
return outFace;
```

Algorithm 2: Bandwidth hungry forwarding strategy

2. Security Considerations

Forwarding Strategy for Bandwidth-hungry

- 現時点ではあんまり考えてない
- 今後の予定:
  - それぞれのUnikernelはPublisherによって署名されるべき
    - ノードはTrusted Imageのみ実行すべき
  - DoS攻撃に対処し, 実行に対して課金するためにも, Interestパケットにも署名されるべき
  - 入力パラメータと結果データは非公開であるべきなので, いい感じに鍵配布しないとな
    ← 今後の研究課題です

3. Evaluatior

In Small Topology

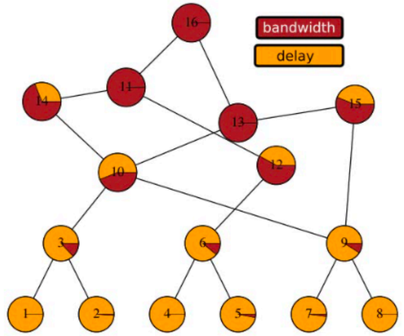


Figure 6: "Delay sensitive" and "bandwidth hungry" functions placement.

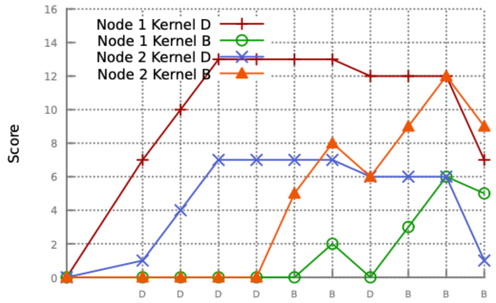


Figure 7: Score function evolution.

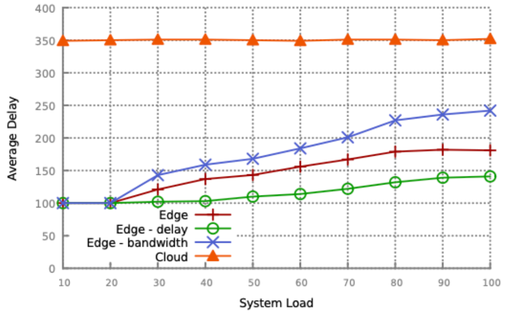


Figure 8: Average delay for different system load values.

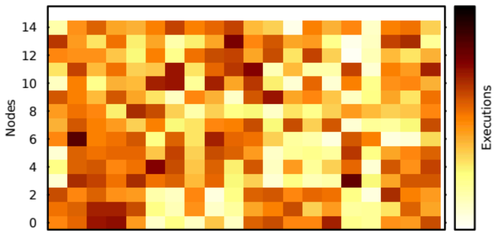


Figure 9: Function executions on different nodes with 100% KS storage size.

Figure 10: Function executions on different nodes with 25% KS storage size.

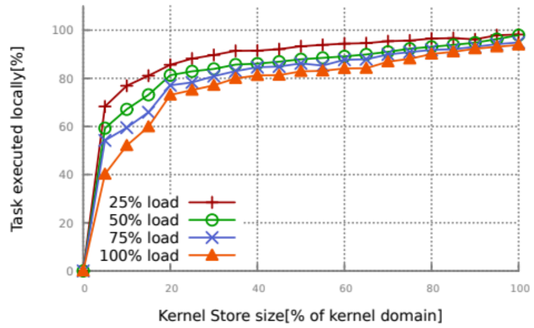


Figure 11: Locally executed tasks for different KS storage size values.

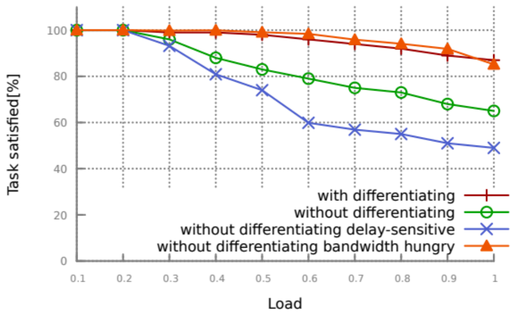


Figure 12: Task satisfaction rate for different system load values.

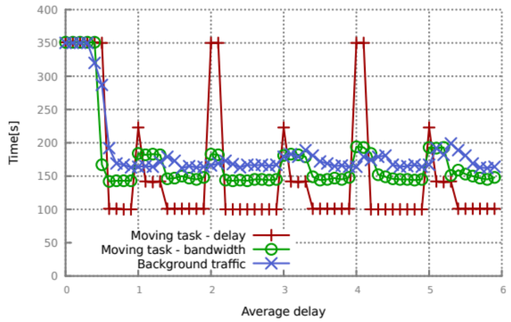


Figure 13: Delay evolution for moving tasks.

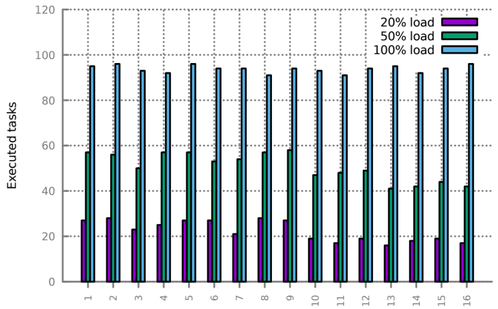


Figure 14: Tasks executed on different nodes.

3. Evaluation

In Sprintlink Topology

- RocketFuel Sprintlink Topology [3]
  - 実際のインターネットに近いトポロジ? でもやってみた

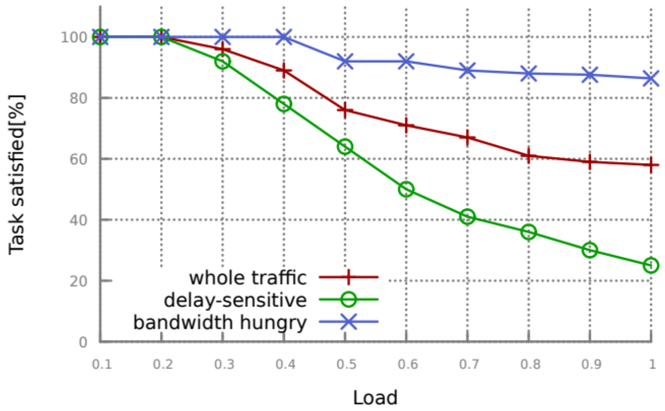


Figure 15: Task satisfaction rate for different system load values.

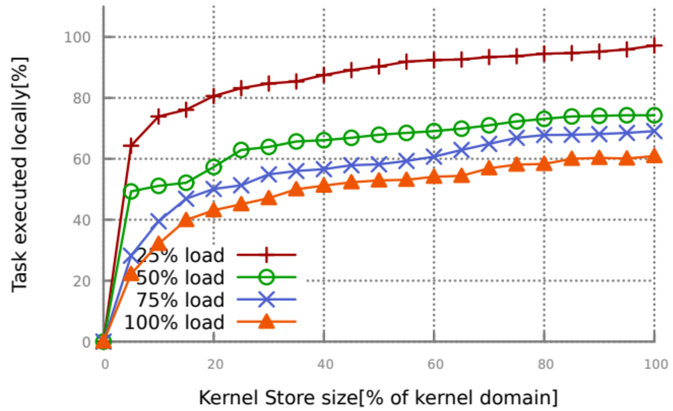


Figure 16: Locally executed tasks for different KS storage size values.

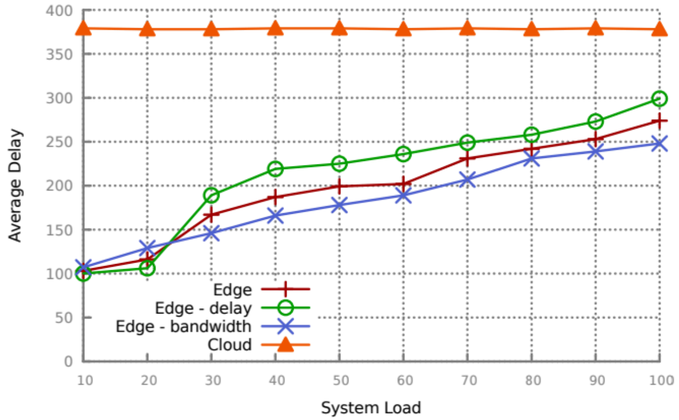


Figure 17: Average delay for different system load values.

3. Evaluation

要約

- Scoped floodingは小規模・規則的なトポロジだと低遅延・良好な満足度を達成するが, 大規模で不規則的なトポロジだとなかなか苦戦する
- NFaaSはすべてのシナリオで, 要求されたタスクの大部分をエッジで実行し, 遅延を大幅に削減できている

4. 感想

Forwarding Strategy for Bandwidth-hungry

- ノードの計算結果が正しいかどうか, 担保できていないのでは?
- 各ノードをすべて同じ会社が運営しているという状況は考えにくそう
  - どのノードが関数を実行したか(=ユーザに対して請求権を持つか), どう合意するの?
    - †Blockchain†でいい感じにできそう
      ex: CoopEdge [4]

参考文献

- [1]: Manolis Sifalakis, Basil Kohler, Christopher Scherb, and Christian Tschudin. 2014. An information centric network for computing the distribution of computations. In Proceedings of the 1st ACM Conference on Information-Centric Networking (ACM-ICN '14). Association for Computing Machinery, New York, NY, USA, 137–146. https://doi.org/10.1145/2660129.2660150

- [2]: Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. 2013. Unikernels: library operating systems for the cloud. SIGARCH Comput. Archit. News 41, 1 (March 2013), 461–472. https://doi.org/10.1145/2490301.2451167

参考文献

- [3]: Neil Spring, Ratul Mahajan, and David Wetherall. 2002. Measuring ISP topologies with rocketfuel. SIGCOMM Comput. Commun. Rev. 32, 4 (October 2002), 133–145. https://doi.org/10.1145/964725.633039

- [4]: Liang Yuan, Qiang He, Siyu Tan, Bo Li, Jiangshan Yu, Feifei Chen, Hai Jin, and Yun Yang. 2021. CoopEdge: A Decentralized Blockchain-based Platform for Cooperative Edge Computing. In Proceedings of the Web Conference 2021 (WWW '21). Association for Computing Machinery, New York, NY, USA, 2245–2257. https://doi.org/10.1145/3442381.3449994