エッジクラスタを用いたレイテンシに敏感な Function-as-a-Service における過負荷時の応答時間について公平なリソース割当

池田 匠[1]　山内 拓人[1]　鈴木 絵理[1]　鄭 顕志[1,2]

概要：
FaaS には通信遅延の問題があるため，リアルタイム性が重要なシステムでの採用が難しい．Wang らの提案した LaSS はエッジクラスタで処理を行い，事前にコンテナを作成することで待ち時間を削減する手法であるが，過負荷時における CPU リソース割当においては応答時間について公平でない課題があった．本論文では，要求 CPU リソースを指標として用いた応答時間について公平なリソース割当手法を提案する．LaSS との比較実験を行い，応答時間の分散が 79%減少したことから，有効であることが示された．

Fair Resource Allocation for Response Time under Overload in Latency Sensitive Function-as-a-Service with Edge Clusters

TAKUMI IKEDA[1]　TAKUTO YAMAUCHI[1]　ERI SUZUKI[1]　KENJI TEI[1,2]

1. はじめに

FaaS (Function-as-a-Service) [1] には，エッジデバイスとクラウドサーバー間での通信遅延が生じやすい特徴があることから，リアルタイム性の求められるシステムにおいて採用が難しいという課題がある [2]．FaaS とは，開発者がサーバーを管理することなく，アプリケーションの構築及び実行が可能なサーバーレスコンピューティングの一つであり，関数を記述したコードをデプロイするだけで実行が可能である．この通信遅延の課題を解消するために，Wang ら [3] は，エッジデバイスとの地理的距離が近いエッジサーバーで可能な限り処理を行い，クラウドサーバーの負荷を軽減するエッジコンピューティングの手法 LaSS を提案している．しかし，LaSS では過負荷時における各コンテナへの割当リソースについては応答時間の公平さを考慮していないという課題がある．そこで，本論文では，応答時間削減のため，要求 CPU リソースを考慮した各コンテナへの CPU リソース割当手法を提案する．

本論文の構成は以下の通りである．2 章で LaSS の技術と課題について示し，3 章にて LaSS の課題を解決するための手法を提案する．4 章では LaSS と比較した評価実験の概要及びその結果に対する定量的，定性的評価を示す．5 章で本論文に関連する研究を紹介し，6 章にて本論文のまとめと将来研究について述べる．

2. LaSS: Latency Sensitive Serverless computations

本論文において扱う FaaS システムのアーキテクチャを図 1 に示す．1 台の Controller ノードと複数台の Worker ノードによって構成されたエッジクラスタを用いる．関数実行リクエストが Controller へ送信され，ロードバランサによって Worker ノード内のコンテナへ振り分けられる．一方で，エポックと呼ばれる一定間隔 (e.g., 10sec 〜 1min) でコンテナ制御処理を行う LaSS モジュールが Controller 内に設置されている．

LaSS の目的はサーバーレス関数のリクエスト到着から実行開始までの待ち時間を一定の制限時間内 (e.g., 100ms)

[1] 早稲田大学
Waseda University
[2] 国立情報学研究所 (NII)
National Institute of Informatics

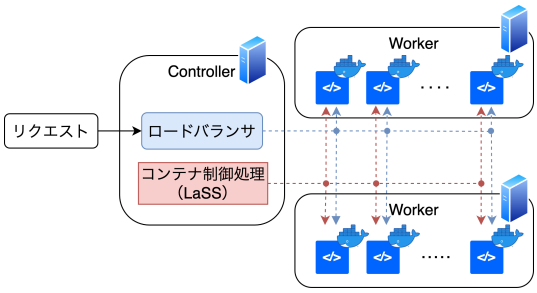

図 1　前提とする FaaS システム

に抑えることである．この待ち時間を抑えるためには，ボトルネックであるコンテナの初期化時間を短くする必要がある．そのため，LaSS では各エポックでの各関数の到着率を記録し，指数加重移動平均 (EWMA) によって次回の到着率を予測することで，事前にコンテナを作成している．このとき，コンテナに割り当てる CPU リソースは，通常，関数登録時に開発者が設定した要求 CPU リソースを用いる．しかし，リクエストの総要求 CPU リソースがエッジクラスタの総 CPU リソースを超える過負荷状態となった場合，関数登録時に開発者が設定する重みに比例した最小限の CPU リソースを保証し，要求 CPU リソースが割り当てられない場合，これを割り当てる．

LaSS では過負荷時において，割当リソース量での公平性を考慮しているが，応答時間の公平性は考慮していない．そのため，過負荷時において，重みが同じ関数は要求リソース量によらず同量のリソースが割り当てられる可能性があるという課題がある．

3. 過負荷時における応答時間について公平な CPU リソース割当

本論文では，エッジクラスタを用いたレイテンシに敏感な FaaS における過負荷時の応答時間について公平なリソース割当手法を提案する．LaSS では，過負荷時の CPU リソース割当に重みを指標として用いていたが，本手法では要求 CPU リソースを指標として用いる．これにより，応答時間について公平なリソース割当が可能となる．

本手法を Algorithm 1, 2 に示す．Algorithm 1 が全体のアルゴリズムを示し，Algorithm 2 はその中で呼び出す処理（再帰的なリソース割当）を記述している．本手法はエッジクラスタの総 CPU リソース C_E，次のエポックで到着すると推測される関数の配列 $\boldsymbol{f}$，各関数の要求 CPU リソースの連想配列 $\boldsymbol{C_f}$ を入力とし，各関数の割当リソースの連想配列 $\boldsymbol{C'_f}$ を出力とする．まず，$\boldsymbol{f}$ を要求リソースの降順にソートした後，要求リソースの大きいものから順に割当リソースを決定していく．ソートした配列 $\boldsymbol{f'}$ の先頭 f を取り出し，$\boldsymbol{C_f}$ から f の要求リソース C_f を取り出す．C_f の $\boldsymbol{C_f}$ 全体に対する割合 r_f を算出し，C_f または $C_E \times r_f$ の小さい方を割り当てる．割当リソースが決定したらエッジクラスタの総 CPU リソースを再計算し，まだ割り当てられていない関数を割り当てていく．

Algorithm 1 応答時間について公平な CPU リソース割当

Input: $C_E,\ \boldsymbol{f} = [f_1, \cdots, f_n],\ \boldsymbol{C_f} = \mathbf{HashMap}\langle f, C_f\rangle$
Output: $\boldsymbol{C'_f} = \mathbf{HashMap}\langle f, C'_f\rangle$
1: $\boldsymbol{f'} \leftarrow \mathbf{sortByRequiredCapacityDesc}(\boldsymbol{f}, \boldsymbol{C_f})$
2: **return** $\mathbf{ARR}(C_E,\ \boldsymbol{f'},\ \boldsymbol{C_f},\ \mathbf{HashMap}.empty)$

Algorithm 2 Allocate Resources Recursively

$\mathbf{ARR}(C_E,\ \boldsymbol{f'},\ \boldsymbol{C_f},\ \boldsymbol{C'_f})$

Input: $C_E,\ \boldsymbol{f'},\ \boldsymbol{C_f},\ \boldsymbol{C'_f} = \mathbf{HashMap}.empty$
Output: $\boldsymbol{C'_f} = \mathbf{HashMap}\langle f, C'_f\rangle$
1: **if** $\mathbf{isEmpty}(\boldsymbol{f'})$ **then**
2:　**return** $\boldsymbol{C'_f}$
3: **else**
4:　$f \leftarrow \boldsymbol{f'}.head$
5:　$C_f \leftarrow \boldsymbol{C_f}[f]$
6:　$r_f \leftarrow C_f / \sum \boldsymbol{C_f}.values$
7:　$C'_f \leftarrow \mathbf{min}(C_f,\ C_E \times r_f)$
8:　**remove** $f, \langle f, C_f\rangle$ **from** $\boldsymbol{f'}, \boldsymbol{C_f}$
9:　**add** $\langle f, C'_f\rangle$ **to** $\boldsymbol{C'_f}$
10:　**return** $\mathbf{ARR}(C_E - C'_f,\ \boldsymbol{f'},\ \boldsymbol{C_f},\ \boldsymbol{C'_f})$
11: **end if**

4. 評価および議論

本論文における Research Question を以下に示す．RQ の評価のため，LaSS と本提案手法との比較実験を行った．

RQ　LaSS と本提案手法では，応答時間はどう変わるか．

4.1 実験概要

まず，実験環境について説明する．controller ノード 1 つと worker ノード 2 つから成るエッジクラスタを AWS 上に構築した．この 3 つのノードのサブネットは別々に構成されており，マシンスペックは共通である．マシンスペックをまとめた表を表 1 に示す．また，実行するサーバーレス関数は「2 から n までの素数を求めるプログラム (n を引数とし，以下 prime 関数と呼ぶ．)」を用いた．

表 1　マシンスペック

項目	スペック
CPU	4 core 第 2 世代カスタム Intel Xeon スケーラブルプロセッサ
メモリ	8 GiB
ネットワーク	10Gbps Ethernet
OS	Ubuntu 20.04LTS Server

次に，実験内容について説明する．表 2 に示す 3 つの異なるパラメータを持つ prime 関数を並列に実行させ，過負荷状態となるようにした．各関数の到着率は 30[req/s] で固定とし，1 分間繰り返し実行した．そして，毎回の応答

時間を計測し，各関数ごとの平均を算出した．また，表 2 中の CPU リソースと n の組における実行時間は事前に計測しており，各パラメータ間の実行時間が近くなるよう要求 CPU リソースを設定している．

表 2　3 つの prime 関数のパラメータと実行時間

関数名	メモリ [MB]	要求 CPU リソース [vCPU]	n	実行時間 [ms]
func1	512	210	3000	107.85
func2	512	300	4000	135.55
func3	512	400	5000	130.92

4.2　実験結果

実験結果を図 2 に示す．この図において，X 軸は手法，Y 軸は各手法における各関数の平均応答時間を示している．また，応答時間についての公平性を確かめるため，各手法ごとの分散を算出した．それを表 3 に示す．

この表から，本提案手法の方が LaSS と比較して各関数間の応答時間の分散が 79%減少した事がわかる．

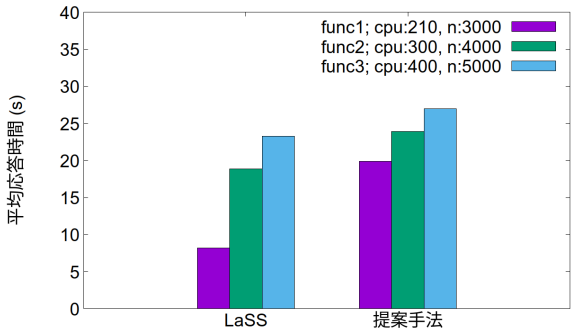

図 2　LaSS と本提案手法の平均応答時間

表 3　LaSS と本提案手法における平均応答時間の分散

手法	分散 [s^2]
LaSS	40.13
提案手法	8.44

4.3　議論

本実験において，本提案手法と LaSS を比較して各関数間の応答時間の分散が 79%減少したことから，要求 CPU リソースを指標とした本提案手法は応答時間について公平なリソース割当に有効であると言える．しかし，本提案手法においても関数間の分散が生じている．これは表 2 における関数間の実行時間の分散が影響していると考えられる．要求 CPU リソースと n の組は関数の実行時間に直接影響するため，応答時間も同じく影響を受ける．よって本提案手法は，開発者の設定する要求 CPU リソースに大きく左右され，関数間における実行時間の分散が小さくなるよう設定されていないと応答時間の公平性を保てないことが推測される．これは，外的妥当性への脅威である．内的妥当性について，本提案手法では同様の環境・条件で実験を行った場合，同等の結果が得られると推測されるため，再現性は担保されていると言える．

5.　関連研究

本研究と同様に，サーバーレスコンピューティングにおける CPU リソース割当の適正化の研究が行われている．Zhao ら [4] は，サーバーレスコンピューティング向けの軽量なオートスケーラーについて設計と実現可能性を議論している．オートスケーラーに用いるアルゴリズムとして，機械学習ベースのアルゴリズムには大量のトレーニングデータが必要であることから，サーバーレスプラットフォームにおいては有用でない可能性があり，単純移動平均 (SMA) や指数移動平均 (EMA) などの軽量なアルゴリズムがサーバーレス関数により適していると結論づけた．

6.　おわりに

本論文では，エッジクラスタを用いたレイテンシに敏感な FaaS における過負荷時の応答時間について公平なリソース割当手法を提案した．本提案手法は要求 CPU リソースを指標として用いるものであり，LaSS との比較実験により，応答時間について公平なリソース割当に対して有効であることが示された．しかし，本提案手法は要求 CPU リソースに大きく左右されるものであり，関数間における実行時間の分散が小さくなるよう設定されていないと応答時間の公平性を保てないことが推測される．本研究における理想は応答時間の分散が 0 になることであるので，また本提案手法では不十分である．この課題の解決につながる将来研究として，本提案手法における要求 CPU リソースについて，ランタイムにおける動的な適正化が挙げられる．

参考文献

[1] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Serverless programming (function as a service). In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2658–2659, 2017.

[2] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[3] Bin Wang, Ahmed Ali-Eldin, and Prashant Shenoy. Lass: Running latency sensitive serverless computations at the edge. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '21, page 239–251, New York, NY, USA, 2021. Association for Computing Machinery.

[4] Yuxuan Zhao and Alexandru Uta. Tiny autoscalers for tiny workloads: Dynamic cpu allocation for serverless functions. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 170–179, 2022.