B-013

ヘテロジニアスハードウェア環境向け Kubernetes クラスタによる動的な負荷分散手法の検討

Consideration of Dynamic Load Balancing Methodology in Heterogeneous Hardware Environments with Kubernetes Clusters

萩原 景太† 菅谷 みどり†
Keita Hagiwara Midori Sugaya

1. 背景

近年，コンテナテクノロジーの台頭により，コンテナ技術をベースとしたクラスタ管理システムである Kubernetes [1]が注目を集めている．高信頼でスケーラブルな特性から，Google や Amazon のクラウドサービス等で利用されている[2,3]．また，Kubernetes は特定の機械学習を行うための最適な環境を提供も行っており [4]，そうしたサービスへの需要は拡大しつつある．こうした機械学習においては，GPU をはじめとした特定のアクセラレータを用いることでより高速な計算手段を提供することが期待されている．例えば MLaaS 向け GPU クラスタ[5]や，再構成可能な FPGA が CNN 等で有効活用されている例が報告されている[6].

このように異なるアクセラレータを搭載した計算機を Kubernetes で利用しようとした場合，これらのアクセラレータの能力に応じた負荷分散の手法が必要である．しかし，現在，異種 Node 間の差異に応じた負荷分散手法は十分提案されていない.

Node 間の差異を考慮する手段として，Node Selector[7]や Node Affinity[7]を使った，Pod（コンテナにカプセル化されたアプリケーション）のスケジューリングポリシーの設定が挙げられる．Node Selector は条件に合致する Node に Pod をスケジュールする機能である． Node Affinity は必須条件と推奨条件を設定し，必須条件を満たしながら，可能な限り推奨条件を満たす Node に Pod をスケジュールする機能である．これらを使用することで，特定のハードウェアを持つ Node に Pod をスケジュールすることができる．しかし，Node Selector では異種 Node を追加する場合には，規模のスケーラビリティを活かすことが難しくなる．Node Affinity では選択される Node が均一であるとは限らないため，計算力に差が生じることが考えられる．以上の点から Pod のスケジューリングによる解決は十分ではなく，負荷分散による解決が必要と言える.

大手のクラウドプロバイダに目を向けると，Google のクラウドでは Maglev[8]と呼ばれる負荷分散アルゴリズムが使われている．Maglev は Source IP/Port，Destination IP/Port，IP Protocol number の 5-tuple のハッシュを用いて転送先のサーバを決定する．この負荷分散手法によってパケットの均等な分散を高速に実現するが，Node の差異，負荷状況が反映されないという課題がある.

2. 目的/提案

本研究は，異種環境での機械学習アプリケーションをはじめとするアクセラレータの使用を要するアプリケーションの平均応答時間短縮を目的とする．目的の実現のために，Kubernetes において Node の差異による計算力の差と負荷状況に応じた負荷分散手法を提案する．本稿では提案する負荷分散手法のモデル及び設計と実装について述べる．また，先行研究である Maglev[8]をはじめとするハッシュ計算がベースとなる負荷分散手法シミュレーションによる比較を行い，提案手法の有効性について述べる.

2.1 提案する負荷分散手法

提案アルゴリズムは Pod が n 個稼動し，Pod_iの計算力がx_iである場合，Pod_iへルーティングされる確率p_iは(1)式により決定される.

$$p_i := \frac{x_i}{\sum_{k=1}^{n} x_k} \qquad (1)$$

x_iは Metrics Data をパラメータとする予測モデルによって決定される．x_iのパラメータとなる Metrics Data は Metrics Server[9]で収集した値である.

2.2 設計と実装

本研究で提案する負荷分散手法では各 Node と Pod の負荷状況に応じた計算力を見積もるための予測モデル（2.1 節）を用いる.

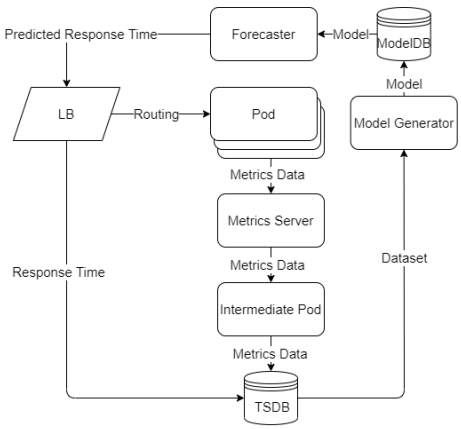

図 1 システムの設計

図 1 は，予測モデルを生成し，負荷分散に反映するシステムの設計を表したものである．Load Balancer（LB）は TSDB（Time Series Database）に実際の応答時間を Push する．Pod や Node の Metrics Data は Metrics Server[9]で収集し，Intermediate Pod を介して TSDB に Push する．Model Generator は一定の周期で TSDB に Push した Metrics Data と実際の応答時間を使って予測モデルを作成し，ModelDB に保存する．Forecaster は一定周期で予測モデルを使って Pod 毎の予測応答時間を LB に送信する．LB は予測応答時間に基づき，各 Pod へのルーティング確率を更新する.

†芝浦工業大学
Shibaura Institute of Technology, 3-7-5 Toyosu, Koto-ku, Tokyo 135-8548, Japan

3. 既存手法との比較

ハッシュ計算をベースとする既存手法よりも，提案手法の応答時間が改善されることを確認するため，表 1 の条件でシミュレーションを行った．

表 1：シミュレーション条件

項目	条件
Pod_iの計算力c_i	$X \sim N(100, s^2)$
Pod の計算力の標準偏差s	$0 \leq s \leq 20$
クラスタ全体の計算力	$\sum c_i = 100{,}000$
タスク単体を完遂するために必要となる計算量w	$w \geq 50$
単位時間あたりに追加されるタスク数n	$w \cdot n \leq \sum c_i$

Pod_iは割り当てられたタスクを単位時間当たりc_iだけ処理することが可能である．c_iは平均 100，標準偏差sの正規分布に従う．タスクが複数割り当てられた際には，ラウンドロビン方式で処理を進める．既存手法の実装にあたり 5-tuple ハッシュ計算に使うデータが存在しない．そのため，既存手法の負荷分散のルーティング先の決定は，Java の Math クラスの random メソッドで代用した．提案手法のルーティング確率の更新は単位時間ごとに行った．メトリクスに応じたPod_iがk個のタスクを実行している時の計算力x_iは(2)式で定義される．

$$x_i := \frac{c_i}{k} \qquad (2)$$

結果を図 2~4 に示す．

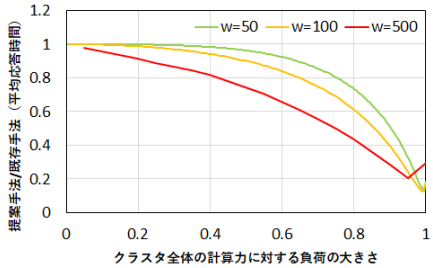

図 2 タスク単体の重さwに対する応答時間の比較（$s = 0$）

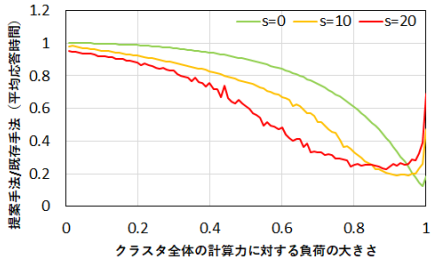

図 3 計算力の標準偏差による応答時間の比較（$w = 100$）

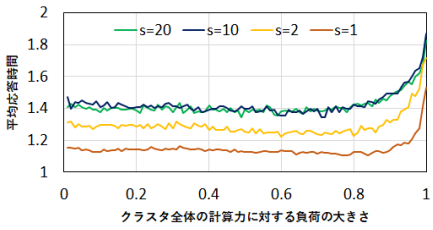

図 4　s=20 に対する平均応答時間の比率（$w = 100$）

図 2~4 において横軸はクラスタ全体の計算力に対する負荷とし，$\frac{w \cdot n}{\sum c_i}$の式により算出した．クラスタ全体の計算力は 100,000 としたことから，$w = 50$のタスクが単位時間に 2,000 個追加される時は横軸の値は 1 となる．

図 2 はタスク単体の重さの影響調査を目的とした．Pod の計算力の標準偏差$s = 0$の時，タスク単体の完遂に必要となる計算量wを変えた時の，既存手法に対する提案手法の平均応答時間の比率の評価を示す．結果からwの値と，単位時間あたりの負荷の総量nが増えるにつれ提案手法による負荷分散の効果が大きくなった．

図 3 は Pod の計算力の標準偏差の影響を調査することを目的として$s = 0,10,20$の時の既存手法に対する提案手法の平均応答時間の比率の評価を行ったものである．横軸の値が 1 付近の場合を除き，Pod の計算力の標準偏差sが大きくなる程，提案手法により改善される度合いが増大している．

図 4 は，標準偏差の大きさと提案手法による実際の平均応答時間を評価した．標準偏差$s = 0$との比較のため，$s = 0$との比率を示した．結果から，提案手法の実際の平均応答時間は標準偏差sが小さいと，平均応答時間が短くなると分かる．しかし，標準偏差sが大きくなるに従い，平均応答時間の増大が収束する傾向が見られる．

4. 結論と今後の課題

本稿ではクラスタを構成する Node の差異，負荷状況を考慮した負荷分散手法の提案と，シミュレーションによる既存手法との比較を行った．結果から，提案手法は平均応答時間を改善することが分かった．また，Pod の計算力の標準偏差が小さくなるほど，平均応答時間が短くなることが示唆された．しかし，標準偏差がある程度大きくなると平均応答時間の増大が収束する傾向がある．しかし，規模のスケーラビリティの観点から計算力の標準偏差はある程度許容すべきであると考える．シミュレーションには予測モデルの生成時間が考慮されていないため，今後は予測モデルの生成を交えた有効性の検証を行う．

謝辞

本研究は，JST，CREST，JPMJCR19K1 の支援により実現しました．

参考文献

[1] Z. He, "Novel Container Cloud Elastic Scaling Strategy based on Kubernetes," 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC), 2020, pp. 1400-1404, doi: 10.1109/ITOEC49072.2020.9141552.
[2] “Kubernetes - Google Kubernetes Engine (GKE) | Google Cloud”, https://cloud.google.com/kubernetes-engine, 2022 年 6 月 15 日参照.
[3] “Amazon EKS（AWS でマネージド Kubernetes を実行）| AWS“, https://aws.amazon.com/jp/eks/, 2022 年 6 月 15 日参照.
[4] “Babylon Case Study | Kubernetes”,https://kubernetes.io/case-studies/babylon/, 2022 年 6 月 15 日参照.
[5] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding, “MLaaS in the Wild: タスク Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters”, 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), (2022).
[6] J. Fowers et al., "A Configurable Cloud-Scale DNN Processor for Real-Time AI," 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), 2018, pp. 1-14, doi: 10.1109/ISCA.2018.00012.
[7] “Assigning Pods to Nodes | Kubernetes”, https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/, 2022 年 6 月 19 日参照.
[8] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, Jinnah Dylan Hosein,” Maglev: A Fast and Reliable Software Network Load Balancer”, 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’16), (2016).
[9] “Resource metrics pipeline | Kubernetes”, https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-metrics-pipeline/, 2022 年 6 月 15 日参照.