ヘテロジニアス負荷分散クラスタにおける負荷分散手法の提案

萩原景太[1]　菅谷みどり[1]

概要：近年，IoT と AI サービスの実現のためにクラウド基盤上でのアクセラレータの活用が見込まれる．一方，データセンタでは，ユーザの要求の変化への対応，リソースの有効活用と応答時間短縮のため異種ノードが追加され，ヘテロジニアスなクラスタが生じることが考えられる．ヘテロジニアスなクラスタではノードの性能が不均一であり，均一なノードを前提とした負荷分散では過負荷/過小負荷なノードが生じることが課題となる．本研究ではヘテロジニアス負荷分散クラスタにおいてパフォーマンスを向上することを目的として重み付き負荷分散を提案する．

キーワード：ヘテロジニアス負荷分散クラスタ，負荷分散クラスタ，負荷分散

1. はじめに

近年，人工知能などの高性能な計算処理と，IoT デバイスなどが連携したサービスなどにより社会課題が解決されることが期待されている[1]．本目的を持つサービスにおいては，大量の IoT デバイスから収集されたデータをスケーラビリティのあるクラウド上の負荷分散クラスタで効率的に処理することが求められる． 近年では Linux Foundation[2]のプロジェクトの一つである Cloud Native Computing Foundation(CNCF)[3]によりコンテナテクノロジーの Containerd[4]，コンテナオーケストレータの Kubernetes[5]といったオープンソースのフレームワークにおいても，負荷分散クラスタが用いられている．このように，負荷分散クラスタは今後さらに普及が期待される．

また，個々の計算機ノードについては，近年 GPU，FPGA などのアクセラレータによる高速化，低消費電力化が行われている[6,7]．そのため，クラウド上でサービスを運用する際も，こうしたアクセラレータを効率よく活用することが求められる．実際のクラウドコンピューティングではユーザの要求の変化への対応，リソースの有効活用と応答時間短縮のため，ハードウェア構成の異なる異種ノードが次々に追加される．このことから，ヘテロジニアスなクラスタ環境においても過負荷/過小負荷なノードが生じないよう，性能に合わせた活用が課題となる[8]．ノード間の負荷を均等にするための技術の一つとして負荷分散が挙げられる[8]．しかし，現在の負荷分散クラスタで使用されている負荷分散はノードの性能が均一であることを前提とする．そのため，現状の負荷分散クラスタは，ヘテロジニアスな環境では過負荷/過小負荷のノードを生じる恐れがある．

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

そこで本研究ではヘテロジニアス負荷分散クラスタでのパフォーマンス向上を目的とし，重み付きの負荷分散手法を提案する．クラスタ内のノードの異種性を考慮した負荷分散により過負荷/過小負荷を回避し，低コストで高いスループットとパフォーマンスの達成を目的とする．本論文の構成は以下の通りである．まず，2 節で技術課題の詳細について述べる．3 節で提案，4 節で今後の課題を述べる．

2. 課題

2.1 均一なクラスタを前提とした負荷分散

現在の負荷分散クラスタで普及している負荷分散は Round Robin やハッシュによる均一な負荷分散である．Kubernetes は iptables や IPVS(IP Virtual Server)をベースに負荷分散を行うが重み付きの負荷分散はサポートされていない．IPVS 単体では Weighted Round Robin，Weighted Least Connection 等の重み付きの負荷分散が実装されているが，複数台のロードバランサが稼働する際は一貫性が問題となる．この問題を回避したロードバランサとして Google の Maglev[9]が挙げられる．Maglev は一貫性のあるハッシュと，Connection Tracking Table によってロードバランサが複数台稼働していても一貫性のある負荷分散が可能となっている．しかし，Maglev は不均一なノードを仮定した負荷分散については十分対応していない．

2.2 重みの設定方法

IPVS では重み付きの負荷分散アルゴリズムが実装されているが，これらのアルゴリズムでの重みは一般的に手動で設定される．大量のサービスをホストする場合はこれらの重みをサービス，ノードの種類ごとに設定しなければならず，現実的ではない．また，重みの値の決定では CPU の周波数等の指標で見積もった性能を元に設定する手法が提案されている[10]が，同種の計算資源でしか比較ができない．

加えて様々なアクセラレータの組み合わせでの性能，アプリケーションとの相性も評価できないという課題がある．

3. 提案

3.1 概要

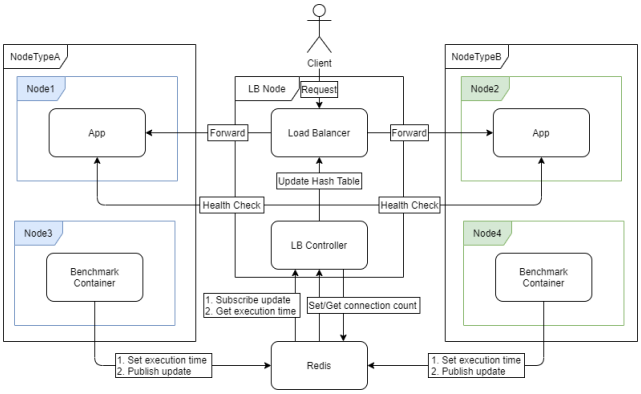

図 1 提案システムの模式図

$$p_i = \frac{\sum t_k}{t_i} \dots (1)$$

$$w_{ij} = \frac{p_i}{c_{ij} + 1} \dots (2)$$

先に述べた課題を解決するため，アプリケーションを模したコンテナでのベンチマークをベースとした重み付きの Maglev ハッシュによる負荷分散を提案する．図 1 は提案システムの模式図である．Node1 と Node2 には負荷分散対象のアプリケーションが稼働する．Node3，Node4 はそれぞれ Node1，Node2 と同種のノードであり，負荷分散対象のアプリケーションに対するノードの性能を計測する．ベンチマークコンテナのイメージは負荷分散対象のアプリケーションの計算を模したプロセスをエントリーポイントとする．そのため，プロセスの終了によりコンテナはライフサイクルを終える．この挙動を踏まえノードの性能評価の指標はベンチマークコンテナの生存時間とした．

3.2 重みの算出方法

アプリケーションが稼働するノードの種類が $type_1, type_2, \dots, type_n$ で構成され，各種ノードでのベンチマークコンテナの生存時間が $t_1, t_2, \dots, t_n$ であるとき，$type_i$ のノードの性能 p_i を式(1)で定義する．性能の高いノードの過負荷を防ぐため，$type_i$ のノード $_j$ のコネクション数を c_{ij} とした時，重み w_{ij} は式(2)で定義する．実装上は，重み w_{ij} はハッシュテーブルに占める $type_i$ のノード $_j$ の割合で表す．

3.3 ロードバランサへの重みの反映

ベンチマークの結果は Redis[11]データベースに書き込まれ，Pub/Sub 通信で LB Controller に通知される．LB Controller はデータベースにあるベンチマークの結果を使い式(1)から p_i を算出し，Redis データベースに書き出したのちに Pub/Sub 通信で各ロードバランサに通知する．各ロードバランサは Redis データベースにあるノードのタイプごとの重みを元に式(2)を使って w_{ij} を算出する．

3.4 ハッシュテーブル更新

ハッシュテーブル更新は既存 Maglev と比較して次の 2 つの条件下で挙動が異なるものとした．1 つ目は未知の種類のノードの追加である．サービスのバックエンドに未知の種類のノードが追加された際に，追加されたノードと同種のノード上でベンチマークコンテナが稼働し，コンテナの生存時間を計測する．コンテナの生存時間は Redis データベースに書き込まれ，Pub/Sub 通信により各ロードバランサへ重みの更新を通知する．ロードバランサはこの Pub/Sub 通信をトリガーとしてハッシュテーブルを更新する．2 つ目はコネクション数の反映である．LB Controller は定期的に各ノードへのコネクション数をカウントし，Redis データベースに書き出す．また，定期的に他の LB Controller が Redis データベースに書き出したコネクション数を読み出し，自身のコネクション数に加算した重みでハッシュテーブルを更新する．

4. 今後の課題

本稿ではヘテロジニアス負荷分散クラスタでの負荷分散手法を提案した．実測に基づく評価が無いため，今後は実装に基づく評価をして有効性を確認する必要がある．

参考文献

[1] Society 5.0 - 科学技術政策 -, 内閣府, https://www8.cao.go.jp/cstp/society5_0/
[2] Linux Foundation - Decentralized innovation, built with trust, The Linux Foundation, https://www.linuxfoundation.org/
[3] Cloud Native Computing Foundation, The Linux Foundation, https://www.cncf.io/
[4] containerd - An industry-standard container runtime with an emphasis on simplicity, robustness and portability, The Linux Foundation, https://containerd.io/
[5] Kubernetes, The Linux Foundation, https://kubernetes.io/
[6] Qizhen Weng and Wencong Xiao and Yinghao Yu and Wei Wang and Cheng Wang and Jian He and Yong Li and Liping Zhang and Wei Lin and Yu Ding, MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters, 19th USENIX Symposium on Networked Systems Design and Implementation., https://www.usenix.org/conference/nsdi22/presentation/weng
[7] 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.
[8] Pawan Kumar and Rakesh Kumar. 2019. Issues and Challenges of Load Balancing Techniques in Cloud Computing: A Survey. ACM Comput. Surv. 51, 6, Article 120 (November 2019), 35 pages. https://doi.org/10.1145/3281010
[9] Daniel E. Eisenbud and Cheng Yi and Carlo Contavalli and Cody Smith and Roman Kononov and Eric Mann-Hielscher and Ardas Cilingiroglu and Bin Cheyney and Wentao Shang and Jinnah Dylan Hosein, Maglev: A Fast and Reliable Software Network Load Balancer, 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16)., https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eisenbud
[10] A. Gupta, O. Sarood, L. V. Kale and D. Milojicic, "Improving HPC Application Performance in Cloud through Dynamic Load Balancing," 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2013, pp. 402-409, doi: 10.1109/CCGrid.2013.65.
[11] Redis, Redis Ltd., https://redis.io/