オートスケーリングにおけるコンテナ数変動時のシステムの振る舞いに関する一考察

水沢 直暉[†1] 岩崎 咲月[†2] 山口 実靖[†3]

†1, 3 工学院大学 工学研究科 電気・電子工学専攻 〒163-8677 東京都新宿区西新宿 1-24-2

†2 工学院大学 情報学部 情報通信工学科 〒163-8677 東京都新宿区西新宿 1-24-2

E-mail: †1 cm18039@ns.kogakuin.ac.jp, †2 j016031@ns.kogakuin.ac.jp, †3 sane@cc.kogakuin.ac.jp

あらまし 軽量な仮想化手法にコンテナ型仮想化があり，近年注目を集めている．多くの場合，分散システムの負荷は時間的に変動し，負荷に合わせてコンテナ数を調整する事が好ましい．本稿では，動的にコンテナ数を調整するオートスケーリングに着目し，コンテナ数増減時のシステムの振る舞いについて考察をする．

キーワード Kubernetes，コンテナ型仮想化，分散処理，オートスケーリング

1. はじめに

ここ数年，Docker[1]をはじめとするコンテナ型仮想化[2][3]への注目が高まってきている．ホスト OS 上でゲスト OS を動作させる VM 型仮想化[4]に対して，コンテナ型仮想化ではホスト OS 上にコンテナと呼ばれる独立した空間を作りその空間の中でソフトウェアなどを動作させる．起動する OS は一つだけであるため，一般的にコンテナ型仮想化は他の仮想化方式と比較して軽量である[5].

コンテナ型仮想化では開発環境など単一計算機内のコンテナ同士でのやり取りは容易である一方，複数計算機からなるクラスター構成では計算機間の連携が煩雑になるという問題点が挙げられる．そのため，複数の計算機で構成される大規模なシステムを構築する際には，コンテナオーケストレーションツールが広く利用されている．

コンテナオーケストレーションツールは，コンテナが稼働しているか否かの監視，負荷に応じたサーバ数の増減といったシステム管理を自動で行うツールである．これまでに Swarm[6]，fleet[7]等のツールが登場したが，2020 年現在では Kubernetes[8]が広く利用されている．

Kubernetes には，一つのコンテナイメージから複数のコンテナ（以下,「レプリカ」と呼ぶ）を作成するスケーリング機能があり，負荷に合わせてコンテナ数を調整するオートスケーリング機能も用意されている．

本論文では，頻繁にスケールイン/スケールアウトを実行し，コンテナ数が高い頻度で増減する状況に着目し，それにおける性能や増減時のシステムの振る舞いについて考察する．

本稿の構成は以下のとおりである．2 章で関連する研究を紹介する．3 章にてコンテナオートスケーリング(HPA)の基礎性能調査を行う．4 章にて考察を示し，5 章で本稿をまとめる．

2. 関連研究

2.1 Kubernetes

Kubernetes は Google の社内システム”Borg” [9]を基に開発されたオープンソースシステムであり，2014 年 5 月にバージョン 1.0 が公開された．Google Cloud Platform の GKE を皮切りに大手クラウドプロバイダがマネージドサービスを提供し始めた事で，Kubernetes がコンテナオーケストレーションツールのデファクトスタンダードとなった．

Kubernetes クラスタは，Master ノードと Worker ノードによって構成される．Master ノードは API ポイントの提供，コンテナのスケジューリング・スケーリングなどを担うのに対し，Worker ノードは稼働しているコンテナ管理を行う．

2.2 Horizontal Pod Autoscaler (HPA)

kubernetes では，ノード数を増減する Cluster Autoscaler，コンテナ数を増減する Horizontal Pod Autoscaler，コンテナごとの CPU やメモリ量を増減する Vertical Pod Autoscaler の 3 つのオートスケーリング方式が実装されている．本論文ではこのうち，Horizontal Pod Autoscaler (以下,「HPA」と呼ぶ)に着目する．

HPA は Pod(コンテナを管理するための最小単位)のレプリカ数を CPU やメモリ等の負荷の大きさに応じて自動的に増減するスケーリング方式である．必要なレプリカ数は以下に示す数式より導出される．

Replicas=ceil[currentReplicas * (currentMetricValue / desiredMetricValue)]

currentMetricValue は各 Pod から取得した 1 分間のリソース使用率の平均値，desiredMetricValue はマニフェストファイル(YAML もしくは JSON 形式で記述した

Pod の定義ファイル)で定義したリソース要求値に対する目的のリソース値である．

通常，Pod のスケールアウトは最大 3 分に 1 回，スケールインは 5 分に 1 回行われる．各スケーリングの実行条件は以下に示す数式の通りである．

スケールアウトの条件

avg(currentMetricValue/ desiredMetricValue) > 1.1

スケールインの条件

avg(currentMetricValue/ desiredMetricValue) < 0.9

2.3 Service[10]

Service は受信した Pod 宛の通信の負荷分散を行うリソースである．通常，Pod はそれぞれ固有の IP アドレスが割り当てられる．削除された Pod は再生成される事がないため，ある時点において同時に稼働している Pod のセットと，別の時点で稼働する Pod のセットは異なる場合がある．ロードバランスのシステムを手動で構築する場合，各 Pod の IP アドレスを調べ，転送先を逐一設定する必要が生じるため，Service リソースを作成することが望ましい．

Service リソースの一種である Externalip は，特定ノードの IP アドレスで受信したトラフィックをバックエンドの各コンテナに転送する．これにより，コンテナの外部疎通性を確立している．

2.4 kube-proxy[11]

kube-proxy は Kubernetes クラスタを構成するシステムコンポーネントの一つであり，各ノード上で動作する．Service リソース宛のトラフィックを Service に紐づけられた Pod に転送する機能を有しており，2020 年現在では以下に示す 3 種類の転送方式が実装されている．

2.2.1 userspace モード

受信したトラフィックをユーザスペース上で処理をして転送する方式．転送先 Pod の選択はラウンドロビン方式で行われる．最初に選択した Pod への接続に失敗した場合，別の Pod で接続を再試行する．

2.2.2 iptables モード

受信したトラフィックをカーネルの iptables 上で処理する方式．転送先の選択はランダムに行われる．選択した Pod が応答しない場合，別 Pod への再接続をせずに接続が失敗する．

2.2.3 ipvs モード

IPVS(IP virtual server)を利用する転送方式．netlink インターフェースを呼び出して ipvs ルールを作成する．

3. HPA 基礎性能調査

本章にて，HPA の基礎調査を行う．図 1 に測定環境を，表 1，表 2 に使用した計算機の仕様を示す．Master ノードと Worker ノード 3 台からなるクラスタを構築し，Worker ノードに 80/TCP ポートをバインドするコンテナをデプロイした．各コンテナ内では HTTP サーバが起動しており，80 ポートへの要求に応答する．

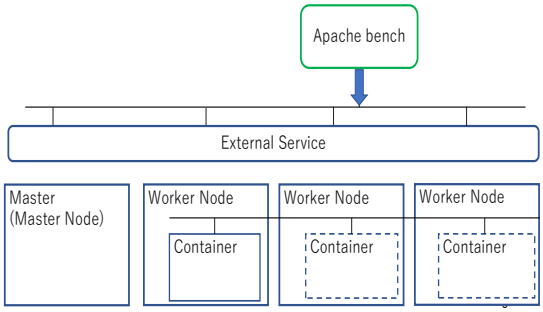

図 1 実験環境図

表 1 計算機の仕様（Master Node）

CPU	Intel Celeron CPU G1101 @ 2.27 GHz
OS	CentOS Linux release 7.6.1810 (Core)
HDD	150 GB_
MainMem	16GB
Docker ver	17.05.0-ce
Kubernetes ver	1.16.2

表 2 計算機の仕様（Worker Node）

CPU	AMD Turion(tm) II Neo N54L Dual-Core Processor
OS	CentOS Linux release 7.6.1810 (Core)
HDD	500 GB_
MainMem	4GB
Docker ver	17.05.0-ce
Kubernetes ver	1.16.2

3.1 負荷固定時の性能

本節にて，kube-proxy の転送方式が iptables モードの場合と，userspace モードの場合の性能を比較する．コンテナ数は 3 とし，クラスタ外の計算機からリクエスト数が 500,000 で並列実行数が 100 の設定で Apache bench (ab)により負荷をかけた．ab は各転送方式で 10 回ずつ実行し，処理時間の平均を算出した．

転送方式ごとの処理時間の平均を図 2 に示す．グラフより，userspace モードでの処理時間は iptables モードでの処理時間の約 2.7 倍となり，処理性能は iptables

モードの方が高いことがわかる.

3.2 コンテナ数ごとの性能

本節にて，コンテナ数 1～6 及び，最小コンテナ数が 1 で最大コンテナ数が 3 の HPA 使用時においての性能を比較する．kube-proxy の転送方式は iptables モードとした．リクエスト数が 500,000 で並列実行数が 100 の設定で ab により負荷をかけた．ab は 10 回実行し，コンテナ数ごとの処理時間の平均を算出した．

転送方式ごとの処理時間の平均を図 3 に示す．グラフより，コンテナ数=3 の時に最も処理時間が短くなったことがわかる.これは，Worker ノードが 3 台でかつ，3 台すべてにコンテナが一つずつデプロイされた事で負荷が均等に分散されたためと考えられる．一方，コンテナ数が 4，5 の時は処理時間が短くならず，コンテナ数=6 の時は 10 回 ab を実行したうちのいずれもエラー(Connection reset by peer)による異常終了をしたため，処理時間を算出する事が出来なかった．これは，1 つのノードにコンテナが複数個デプロイされ，コンテナごとに使えるリソースが減少した事でスループットが低下したと考えられる.

HPA 使用時はコンテナ数=4，5 の時と比較して性能が低い様に見えるが，これは測定開始時のコンテナ数が 1 だったことで測定開始直後の処理時間がコンテナ数=1 の時と同等になったためである.図 4 に示した ab 実行回数ごとの処理時間のグラフより，ab の 2 回目以降はコンテナ数が増加し処理時間がコンテナ数=3 と同等になったことがわかる.

3.3 負荷変動時の性能

本節にて，iptables モードの場合と userspace モードの場合において負荷が変動する状況における振る舞いを調査する．関連研究の 2.2 節より，通常の HPA の設定ではスケールアウトは最大 3 分に 1 回，スケールインは 5 分に 1 回行われる．しかし本実験では，スケールアウト及びスケールインの間隔を 10 秒とし，頻繁にコンテナ数が増減する状況でのシステムの動作を観測した.

コンテナの最小個数は 1，最大個数は 3 とし，desiredMetricValue= 0.25 とした．コンテナのスケールアウト条件は currentMetricValue≧0.275，スケールイン条件は currentMetricValue≦0.225 とした．また，リクエスト数が 500,000 で並列実行数が 100 の設定と，リクエスト数 100,000 で並列実行数 1 の設定で交互に各 10 回(合計 20 回)ab を実行した.

転送方式ごとのコンテナ数の推移を図 5，図 6 のグラフに示す．グラフの縦軸はコンテナのレプリカ数，横軸は測定開始からの経過時間[s]である．また，緑色の縦線は直前で実行していたベンチが正常に終了した時刻，赤色の縦線はエラーによってベンチが途中で終

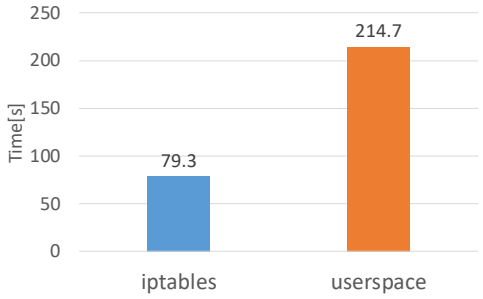

図 2 転送モードごとの性能

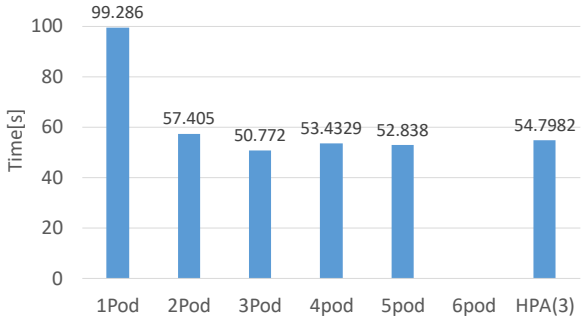

図 3 コンテナ数ごとの性能

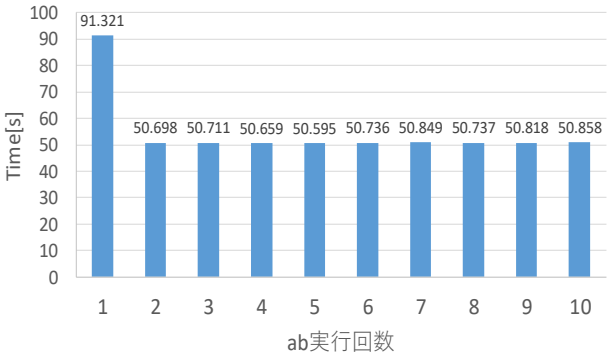

図 4 HPA(コンテナ数 1～3)性能

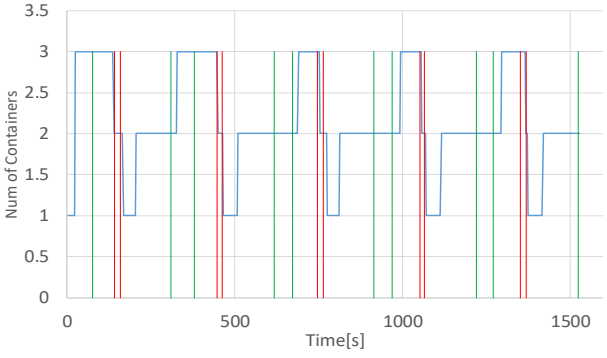

図 5 HPA 使用時のコンテナ数の推移(iptables)

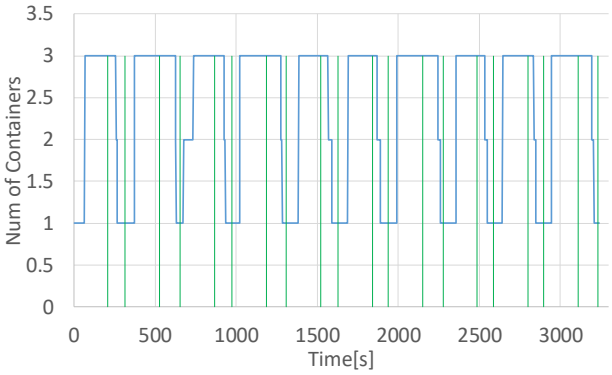

図 6 HPA 使用時のコンテナ数の推移(userspace)

了した時刻を示している．グラフより，iptables モードではコンテナ数の減少時に ab が異常終了したことがわかる．一方，図 6 の userspace モードを用いた実験ではコンテナ数の変動による ab の異常終了は見られなかった．

4. 考察

kube-proxy の userspace モードと iptables モードでは，iptables モードの方が性能に優れていた．userspace モードでは受信したパケットがカーネルスペースとユーザスペース間を行き来するため，処理時間が長くなるためである．

一方で，iptables モードではコンテナ数減少時にエラーが発生したのに対し，userspace モードではエラーが生じなかった．userspace モードではコンテナへの接続を拒否された際に自動的に別コンテナへの接続を再試行するが，iptables モードではこの機能を有していない事から，削除されたコンテナへのアクセス要求を拒否されたことで接続が失敗したと考えられる．

5. おわりに

本稿では kube-proxy の転送方式ごとの性能評価及び，コンテナ数が高頻度で増減する状況の Kubernetes システムの性能と振る舞いについて調査を行った．結果，カーネル空間で動作する iptables モードが性能に優れるが，同点転送方式はコンテナ数減少時にエラーが生じることを示した．

今後は iptables モードのシステムを改変し，スケーリング時にエラーが生じない手法を考察する予定である．

参 考 文 献

[1] Dirk Merkel, "Docker: lightweight Linux containers for consistent development and deployment", Linux J. 2014, pp. 239, 2014.

[2] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, Larry Peterson, "Container-based operating system virtualization: a scalable high-performance alternative to hypervisors", Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (EuroSys '07), pp. 275-287, 2007.

[3] A.B.S., H.M.J., J.P. Martin, S. Cherian, Y. Sastri, "System Performance Evaluation of Para Virtualization Container Virtualization and Full Virtualization Using Xen OpenVZ and XenServer", 2014 Fourth International Conference on Advances in Computing and Communications, pp. 247-250, 2014.

[4] Zhang B. et al. (2010)" Evaluating and Optimizing I/O Virtualization in Kernel-based Virtual Machine (KVM)", In: Ding C., Shao Z., Zheng R. (eds) Network and Parallel Computing. NPC 2010. Lecture Notes in Computer Science, vol 6289. Springer, Berlin, Heidelberg

[5] Miguel G. Xavier, Marcelo V. Neves, Fabio D. Rossi, Tiago C. Ferreto, Timoteo Lange, Cesar A. F. De Rose, "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments", 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2013,

[6] Docker.inc, "Swarm mode overview", https://docs.docker.com/engine/swarm/,July 2019.

[7] Core OS," fleet 1.0.0 Documentation", https://coreos.com/fleet/docs/latest/, Dec 2019.

[8] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," in IEEE Cloud Computing, vol. 1, no. 3, pp. 81-84, Sept. 2014. doi: 10.1109/MCC.2014.51

[9] Abhishek Verma, Luis Pedrosa, Madhuker Korypolu, "Large-scale cluster management at Google with Borg", EuroSys '15: Proceedings of the Tenth European Conference on Computer Systems, no.18, Pages 1–17, Bordeaux, France, April 2015.

[10] CLOUD NATIVE COMPUTING FOUNDATION,"Service", https://kubernetes.io/docs/concepts/services-networking/service/, Dec 2019.

[11] CLOUD NATIVE COMPUTING FOUNDATION,"kube-proxy",https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/, Dec 2019.