7H-03

負荷履歴を用いたKubernetesにおけるアダプティブスケジューリング

蕪木　貴央†　　藤田　悟‡
法政大学大学院情報科学研究科†　　法政大学情報科学部‡

1. はじめに

近年，アプリケーションの仮想化技術のひとつとして，コンテナ技術が注目を集めている．コンテナとは，ハードウェアを仮想化する従来の仮想マシン(VM)とは違い，OSの仮想化を行う事でアプリケーションを分離する仕組みであり，VMと比べて軽量，かつ，起動が高速である事が特徴である．

コンテナをクラスタ上で運用するためには，コンテナオーケストレータと呼ばれるプログラムを用いる．このオーケストレータとしてはKubernetes, Docker Swarm, Apache Mesos等が開発されているが，本研究はKubernetesを対象とした．

Kubernetesのスケジューラは，CPU，メモリ，ストレージの状態をスケジュール時に確認し，それぞれのリソース使用量が少ない（または多い）ノードに優先的にスケジュールする，後述のWorst-Fitあるいは Best-Fitに相当する処理を行う．加えて，現実の運用では，必ずしもビン，即ちクラスタのノードの削減が最適とは限らず，Quality of Service (QoS)が優先される事も多い．

そこで本研究では，Kubernetesのスケジューラを拡張し、ネットワークの実行履歴を用いてコンテナ間の通信量を考慮するスケジューリング方式を提案し，その効果を実験で評価する．

2. 関連研究

仮想マシンやコンテナをクラスタに効率良くスケジュールする基本方式として，ビンパッキング問題が知られている．ビンパッキング問題は，大きさの異なるアイテムをビンと呼ばれる容器に詰め込み，そのビンの数を最小にすることを問うNP困難問題である．Kubernetesで用いられるWorst-FitやBest-Fitは，Ullmanによって初めて証明された[1].

ビンパッキングに限らず，VMやクラウドコンピューティング環境で用いるアルゴリズムとして，エネルギー効率やQoSなどの異なる観点から様々な手法が提案されている [2]．ネットワークに焦点を当て，木構造を用いてビンパッキングを改良した研究としては[3]がある．

Kubernetesの最小管理単位は ポッドと呼ばれ，単一あるいは複数のコンテナで構成される．Kubernetesのスケジューラの研究として，このポッドをグループ化してスケジュールする手法 [4]や Quota と呼ばれる制約システムを改良する手法[5]が提案されてきた．

3. 負荷履歴を用いたスケジューリング

3.1. 基本設計

本研究では，Kubernetesによる高可用なクラスタリソースの管理機能に注目し，分散環境で重要となる通信量の最適化が可能なスケジューリング方式を提案する．既に述べたように，Kubernetes の最小管理単位はポッドである．Kubernetes上のアプリケーションは，複数のポッドで構成され，ポッド間には様々な通信が発生する．各ポッドは，高可用性のために複数のノード(クラスタを構成するVMまたは物理マシン)上に分散配置されて実行されることになるが，ポッド間の依存性の違いから，相互の通信が多いポッドが異なるノードに配置されると，期待性能を達成できなくなる．ポッド間の通信量については，実行環境依存であり，事前に想定することが難しい．そこで，本研究では，サービスの運用開始前に，仮運用期間を設定し，ポッド間の通信量を負荷履歴として蓄積し，本運用の際に，この負荷履歴を利用してポッドの最適配置を行うスケジューラを研究開発した．

3.2. 通信の測定と保存

通信量履歴の取得にあたっては，Kubernetesのネットワークを定義する Container Network Interfaceの監視を行うプログラムを自作した．すなわち，各ポッドの持つプライベートアドレスを割り出し，そのアドレスの通信量を監視することで，ポッド間の通信量を把握した．また，取得し

Adaptive Scheduling Based on Historical Resource Usage in Kubernetes
† Takao Kaburaki, Graduate School of Computer and Information Sciences, Hosei University
‡ Satoru Fujita, Faculty of Computer and Information Sciences, Hosei University

たデータは，時系列データベースである Prometheus に保存した．

3.3.スケジューラ

スケジューラは，Prometheus に保存されている履歴の 30 分平均値を利用して，ポッドのスケジュールを行う．スケジュール対象のポッドが追加された時，スケジューラはそのポッドの履歴を参照して 0 から 100 の整数でスコアリングを行い，通信量が多いポッドが配置されているノードに優先的にポッドが配置されるようにする．

4. 実験

4.1. 実験環境

実験は，Kubernetes クラスタに，先述のネットワーク監視ツールを追加した環境で行う．Kubernetes クラスタは，6 台の Raspberry Pi 4B (4GB)によって高可用構成で作成する．抽出したデータの収集，保存，処理を行うデータベースには,時系列データベースの 1 つである Prometheus を用いる．Prometheus 実行用に，更に 1 台の Raspberry Pi 4B を利用する．

4.2. 実験

実験には，ネットワークのスループットを測定するツールである iperf を用いる．1 つの iperf クライアントは 1 つの iperf サーバに接続し，通信を行う事で，スループットを測定する．本研究では，iperf サーバ 1 つを含むポッドを 10 個，それらのいずれかのサーバと通信する iperf クライアントを 1 つ以上含むポッドを 10 個用いて，スループットを測定する．この時ポッドに含まれる iperf クライアントは，実験 A で 1 つ，実験 B で 3 つである(図 1).

実験では，最初に 45 分間ポッドを動作させ，その後クライアントのポッドを一旦削除し，リスケジュールを行う．リスケジュール前後においてそれぞれ 45 分間稼働させた時点で，各ポッドが実行されているノードと，全体のスループットの 30 分平均を確認し, 本手法のスケジューラによる変化を調べる．

4.3. 実験結果

各クライアントは，それぞれ最も通信量が多いポッドと同一のノードに配置されることが確認できた．その結果，スループットの 30 分平均のグラフ(図 2)より，実験 A，B 共に，リスケジューリング後に全体のスループットが向上したことが確認できた．

5. おわりに

本研究によって，ネットワークの状況を考慮することで，QoS が改善される可能性がある事が分かった．一方で，CPU やメモリ等，その他のリソースについても履歴を用いて，最適なリソースを選択しながらスケジュールを行うといったことが考えられるため，今後の課題としたい．

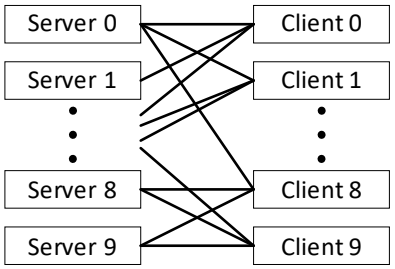

図 1. 実験 B の構成

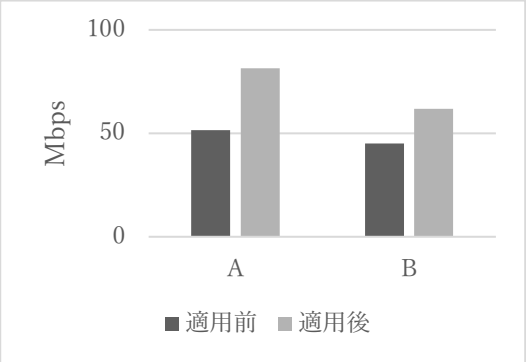

図 2. 実験 A, B それぞれのリスケジューリング適用前後のスループット

参考文献

[1] J.D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University, Princeton, NJ, 1971.

[2] Usmani, Z. and Singh, S., 2016. A Survey of Virtual Machine Placement Techniques in a Cloud Data Center. Procedia Computer Science, 78, pp.491-498.

[3] Dong, J., Wang, H. and Cheng, S., 2015. Energy-performance tradeoffs in IaaS cloud with virtual machine scheduling. China Communications, 12(2), pp.155-166.

[4] GitHub. 2021. Kubernetes-Sigs/Scheduler-Plugins. [online] Available at: <https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/kep/2-lightweight-coscheduling> [Accessed 6 January 2021].

[5] GitHub. 2021. *Kubernetes-Sigs/Scheduler-Plugins*. [online] Available at: <https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/kep/61-Trimaran-real-load-aware-scheduling> [Accessed 6 January 2021].