Vol.2019-OS-147 No.12
2019/7/25

Elastic Scheduling に向けたマイクロサービス性能モデルの検討

千葉 立寛[1,a)]　中澤 里奈[1]　堀井 洋[1]

概要：近年，クラウド上に構築されるアプリケーションは，コンテナおよびコンテナマネージメントシステムの登場により，マイクロサービス型アーキテクチャで構成されることが一般的となってきている．マイクロサービスで構成されたアプリケーションは，個々のコンテナを容易にデプロイ，アップデート，スケールさせることが可能となった一方で，マイクロサービス個々の性能を正確に把握してアプリケーション全体として最適化することは，よりいっそう複雑さを増している．レプリカ数やリソースの割り当て，適切なタイニングでのスケーリングなどを適応的に判断して柔軟なスケジューリングを行うためには，マイクロサービスの性能モデルの構築が必要である．本稿では，アプリケーションおよびマイクロサービス全体の実行ログやトレースを元に性能モデルを生成する手法を検討し，Kubernetes 上に実装したプロトタイプを紹介する．また，生成したモデルを用い，マイクロサービスに対するリソース割り当ての最適化を行う．

1. はじめに

近年，Docker に代表されるコンテナ型仮想化技術の発展と，コンテナを管理するためのコンテナオーケストレーションシステムの登場により，クラウド上で動作するアプリケーションの多くが，マイクロサービス型アーキテクチャで構築されることが一般的となってきている．個々のサービスや機能ごとに分割されたコンテナ群が，それぞれで公開する API(REST や gRPC) を通じて疎結合的に構築されたマイクロサービスにおいては，アプリケーションが提供するサービスを停止することなく，必要に応じてコンテナをアップデートしたり，コンテナを増やしてスケーリングさせたりするような Elasticity を導入することが容易なため，モノリシックに構築されたアプリケーションと比べて柔軟性や自律性を高めることが可能となっている [1]．

デプロイされたマイクロサービス型アプリケーションの性能を最適化するためには，コンテナ個々において最適なリソース配分やコンテナレプリカ数，ワークロードの特性に応じた適切なタイミングでのスケーリング・デスケーリングなどを適応的に判断することが求められるが，それらを実現するためには，個々のサービスやコンテナの性能を正確に判断・把握していくことが必要不可欠である．

このようなアプリケーションの振る舞いを正確に判断するためのアプローチとして，様々なメトリクスをもとにした性能モデルの構築が挙げられる．精度の良い性能モデルを用いることで，モデル化された結果に基づいたキャパシティプランニングやロードバランシング，性能ボトルネックの解析，さらには動的な性能予測によるスケーリングの実現等が可能となる．性能モデルの構築という観点で見た場合，マイクロサービス型アプリケーションにおける性能モデルの構築は，いくつかの点で従来のモノリシック型アプリケーションと比べて非常に難しくなってきている．1つ目は，コンテナ化による高いポータビリティ性によって，実行されるクラウド環境が多岐にわたる点である．同一のクラウド環境であっても同じマシン，ひいては同じアーキテクチャ (x86_64, ppc64le, aarch64, etc.) で実行されるとは限らず，固定されたシステムを前提にすることが出来ないことが多い．2つ目は，コンテナそのものが頻繁にアップデートされる点である．機能の追加・削除やコンポーネント間の構造化データの変更などが，頻繁に起こりうるマイクロサービスにおいては，固定されたモデルがそのまま使えるとは限らない．3つ目は，それら各コンテナ間が複雑に連携して一つのアプリケーションを構築している点である．多階層で構築されるマイクロサービスでは，複数の独立したコンテナ・コンポーネントを加味して全体として最適化する必要がある．

Kubernetes に代表されるようなコンテナを管理するオーケストレーションシステムにおいては，CPU やメモリ，ネットワーク利用率，リクエストの到着レートやアプリケーション固有のメトリクスなど，数多くのメトリクスが

[1] 日本アイ・ビー・エム（株）　東京基礎研究所
IBM Research
[a)] chiba@jp.ibm.com

定期的かつ自動的に収集され，時系列データの形式で保存されるテレメトリーサービスが利用可能になっていることが多い．例えば，コンテナのレプリカを増減させるスケーリングを最適化するとした場合，これらのデータを用いて，アドホックに定義されたルール・ヒューリスティクス (CPU 利用率のしきい値など) に基づくコンテナスケーリングが一般的に取られるアプローチの一つである．ヒューリスティクスベースの手法もある程度のレベルまでは十分に機能すると考えられるが，上述のような複雑さを増したマイクロサービスにおいて，個々に適したヒューリスティクスを整備しながらより柔軟なスケーリングを実現することは困難を極める．利用可能なメトリクスデータをもとに個々のサービスに対して適応した性能モデルを継続的かつ自動的に構築するためには，機械学習的なアプローチを用いることが性能最適化においても重要である．

本稿では，Kubernetes 上で収集可能な様々なメトリクスをもとに，マイクロサービスの性能モデルを作成するためのアプローチとして，時系列モデリングを行っていく手法を検討していく．Kubernetes 上で動作して継続的にモデルを構築するためのプロトタイプを実装し，統計的および機械学習的アプローチを用いて作成したモデルの比較・評価を行う．さらに，それらのモデルを用いた応用例として，Kuberenetes クラスタ全体にデプロイされたコンテナのキャパシティプランニングを行い，結果として最大で 45%程度のリソース使用率が改善することが確認された．

2. 背景

2.1 Kubernetes

Kubernetes[*1] は，近年最も使われているコンテナアプリケーション管理のためのオーケストレーションシステムである．Google で使われていた Borg [2] をベースとし，クラスタ上でのコンテナのスケジューリング，負荷に応じたロードバランスやコンテナのオートスケーリング，停止したコンテナのリスタートなど，コンテナシステム全体を管理するための仕組みを有している [3]．コンテナ環境をホスティングする事実上のデファクトとなりつつあり，クラウドネイティブという言葉に代表されるように，Kubernetes を中心としたエコシステムが形成されており，メトリクスを保存するための時系列データベースである Prometheus や，サービスメッシュを柔軟に構築する Istio など，多くのサービスやコンポーネントが Kubernetes と連携・協調して動作している．

Kubernetes に Horizontal Pod Autoscaler(HPA)[4] と呼ばれる機能がある．HPA では，ユーザが指定したレプリカ数に収まる範囲で，特定のメトリクスを定期的に計測して，事前に設定した基準値を上回る/下回る場合に Pod を自動的に増減させるための機能である．例えば，過去 5 分間の平均 CPU 利用率が 80%を超えた場合，新たな Pod を追加してスケールさせることが出来る．事前にアプリケーションの振る舞いやリソースの利用率など多くの知識がある場合，単純なルールで Elasticity を実現できる一方で，これらを個々のサービスやコンテナに適切にセットするのは非常に難しい課題であり，いくつかの研究がなされている [5]．

[*1] https://kubernetes.io/

2.2 Machine Learning for Systems

近年の機械学習やディープラーニング技術の発展により，様々な分野での応用が始まっており，システム最適化に対して ML/DL を適用する動きが広まっている [6]．例えば，Google 内のデータセンターのエネルギー効率をニューラルネットワークを用いて学習し，冷却コストを大幅に改善する最適化を行えることを示している [7]．VM のアロケーションやジョブスケジューリングなど，今後も多くのエリアで適用が進み，線形性・非線形性を考慮して解きたい問題に適切なモデリング手法を選んでいく必要がある．

2.3 時系列モデリング手法

時系列データが与えられたときに，それを用いてモデリングする手法は大きく分けて，統計的モデリングと機械学習的モデリングの２通りがある．本稿では，これら２つの手法をマイクロサービスの性能モデル構築に向けて検討する．

統計的時系列モデリング

時系列データの解析をして予測するために，自己回帰モデル (AR モデル) などが広く用いられているが，中でも自己回帰移動平均モデル (ARIMA モデル) は，よく知られた予測モデル手法の一つである．過去のデータに対する自己相関と移動平均での線形和として表現される時系列データの階差をとったものであり，線形性のある時系列データに対しては非常によく当てはまることが知られている．また，非線形性を考慮しながら観測値をもとに予測する拡張カルマンフィルタなど，多くの解析手法が研究されている．

機械学習的時系列モデリング

一方，近年ではニューラルネットワークを用いた時系列データの解析手法も幅広く使われている．時系列データに対して再帰的な回帰を行うことが可能な RNN や GRU などが提案されてきたが，なかでも LSTM(Long Short-Term Memory) は，様々な分野で応用されているモデルの一つである．線形・非線形を問わずに適用できる点や，周期性を有するデータも柔軟に吸収してモデル化できるメリットがあり，様々な ML/DL フレームワークの登場により，比較的容易に開発できるようになったため，非常に多くのモデルが研究されている．また，オンラインラーニングの分野においても LSTM を用いる研究が登場してきている [8]．

3. 機能および実装

3.1 機能の検討

本稿で実現したい機能として挙げられのは，(1) 継続的なメトリクスの収集，(2) 時系列データのモデル化・利用・管理，(3) イベント発生時にアクションをトリガーするためのモジュールである．モデリングを行うためには，まず第一に，絶えずコンテナやシステムから生成される様々なメトリクスを継続的に収集しておく必要がある．Kubernetes を前提とした場合，様々なメトリクスが時系列データとして Prometheus に保存される．cAdvisor によるコンテナメトリクス，Istio による Request Rate 等のネットワークメトリクス，Dapper や Jaeger のような分散トレーシングデータなど，多くのデータが再利用可能なエコシステムが出来上がっており，必要に応じてカスタムメトリクスも取得可能である．本稿では，自動的に収集されるデータをベースにして，モデリングが可能かどうかを検討する．

次に時系列データのモデリングであるが，Prometheus 上に保存されたメトリクスは一次データであり，そのままでは学習には適さない．正規化を行ったり，欠損データやエラーを適切に調整し，モデリングに適したデータに変換することが必要となる．また，継続的に正しく予測するためには，学習済みモデルの適合率が悪くなった場合に，再度モデリングを行うことも必要となる．

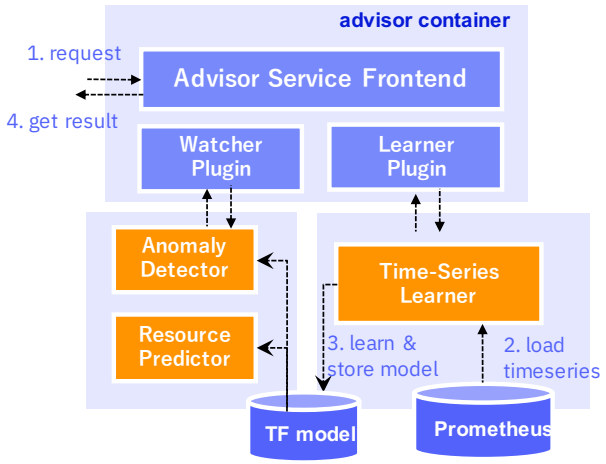
最後に，生成されたモデルデータを用いて，Elastic な最適化を行うための様々なタイミングを適切に判断し，実際のアクションとして結びつけることが必要となる．例えば，コンテナに割り当てられたメモリリソースを例に考えると，近い将来設定されたリソースクォータを超えることが検知された場合，クォータを適切に拡張するのか，コンテナ自体のレプリカを増やして対処するのかなどの意思決定が必要となる．これらのイベントを適切に検知して，モデルから得られた結果をもとに最適化を行うイベントハンドラが必要である．

3.2 実装

各コンテナ・マイクロサービスの振る舞いを，各コンテナから生成される種々の時系列データを推測・学習することでモデル化することを目指しているが，我々の先行研究として，Kubernetes 上で動作するコンテナ内のミスコンフィグを自動的に検知して，ユーザおよびコンテナ自身に最適なコンフィグをフィードバックして性能を改善するためのフレームワークを提案 [9] している．共通して利用したい機能が実装されているため，本稿では，このフレーワムワーク上で動作するプラグインとして，時系列モデリング・予測を行う機能を実装した．

図 1 は，このフレームワーク上にプラグインとして実装した時系列データモデリングとイベントハンドラである．ユーザやコンテナから直接アクセス可能となるように，REST API をフロントエンドに用意した．ユーザリクエストやモデルの乖離を検知するモジュールの要求に応じて，各コンテナやサービスのモデリングを行い，その結果を Persistent Strange に保存する．Prometheus に保存されている時系列データは，Pandas に変換した後，Keras および Tensorflow を用いてモデルを生成する．リソースアロケーションに関するアドバイスや，メトリクスの予測値，スケジューリングポリシーの変更等を行うイベントハンドラが，生成されたモデルを用いて最適値を計算しユーザに結果を返すように実装した．また，自動的に Kubernetes Object に対して反映することで，最適化も自動的かつ継続的に行えるようにするよう実装した．

図 1 時系列モデリングおよびイベントハンドラアーキテクチャオーバービュー



4. 評価

4.1 評価環境および実験

Softlayer Cloud 上に 4 台のマシン (8 vCPU, 16GB RAM, 1TB Disk, Ubuntu 16.04.05, kernel 4.15.0-33) を用意し，IBM Cloud Private を用いて Kubernetes 環境をセットアップした．また，ソフトウェアのバージョンは，Kubernetes 1.13.5，Prometheus 2.3.1，Docker 18.06.1-ce である．その上で Helm を用いて多くのアプリケーション・ミドルウェアをセットアップした．

多くのコンテナ，サービスがセットアップ後に起動されるが，それらのサービスがどのように振る舞い，どの程度のリソースを消費するのかをユーザが事前に判断することは難しいため，各コンテナやサービスが要求するリソースは実際の利用状況とは乖離してしまうことが多い．その結果，実際よりも少なく，または，多く見積もってしまうことで，非効率的なリソース利用率となる．本稿では，デプロイ後にそれぞれのサービスに対してメモリ利用率のモデルを作成して，LSTM および ARIMA で作成したモデル自体の評価を行い．さらに作成したモデルをもとに Kubernetes

図 2 ARIMA による DB サーバのモデリング

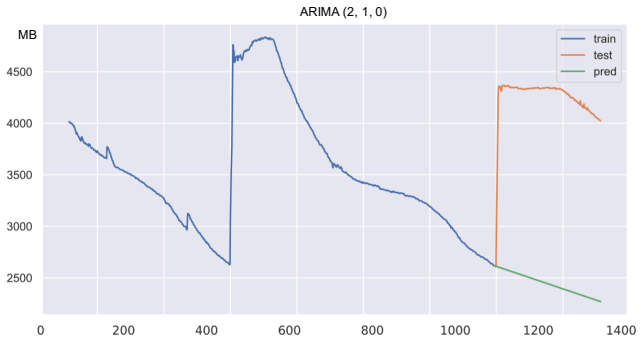
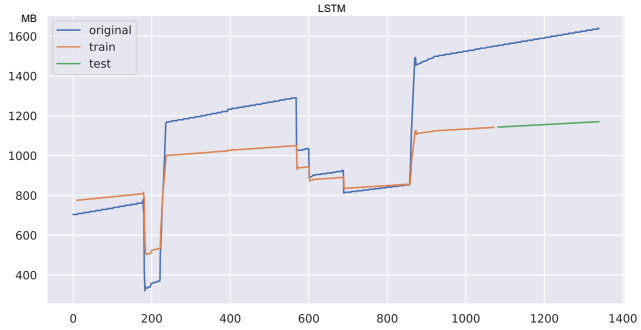


図 3 LSTM による DB サーバのモデリング



Cluster 全体のキャパシティプランニングの最適化を行う．

4.2 モデル構築および評価

いくつかのコンテナ対して，ARIMA および LSTM の 2 つの手法でモデリング行い，未知の時系列に対して正しく予測できるかを確認する．図 2 は，ARIMA モデルを用いて Kubernetes 上で可動している DB サーバのメモリ使用率をモデル化したものである．1 分ごとに取得された過去 24 時間分のメモリの利用率のうち，80%のデータを学習に用いてモデルを作成し，残りの 20%のデータで検証している．本稿では，ARIMA モデルにおけるハイパーパラメータ (階差など) を自動的に調整して最適なモデルを探索する autoarima を用いてモデルの作成を行った．定期的に負荷が高まってメモリ負荷が上昇し，その後ガベージコレクションによってメモリが減少する周期性が見て取れるが，作成した ARIMA モデルでは実際の周期性をモデルに取り込めていないことがわかる．その結果，実際には利用率があがるタイミングにおいても，メモリが減少するというような予測となっている．線形性・非線形性を考慮してモデルを構築しないとうまくいかない例となっている．

一方，LSTM を用いてモデルを作成する例を紹介する．同様に 80%のデータを学習に用いており，LSTM としては 10 time steps に対して 1 step 先のデータを予測するように 1 層の LSTM(ユニット数 128，活性化関数 Relu，最適化関数 Adam，忘却率 0.1) を構築して学習した．学習が

図 4 認証サーバの LSTM によるモデリング

収束したと判断された場合に途中で学習を打ち切る Early Stopping を設定し，最大でも 100 epoch で終了するように設定している．また，その他のハイパーパラメタに関しては，すべての LSTM モデルで固定としている．図 3 は，同様の DB サーバのメモリ使用率を LSTM を用いてモデル化したものである．観測値と予測値の間に一定量のギャップがあるものの，全体として利用率の上昇・下降の特徴を捉えることができている．

さらに，Kubernetes 内の各サービスに対して認証を提供するコンテナのワークロードを LSTM でモデル化，および，その際の学習状況を示した図である．DB サーバのケースと同様，細かい変動のあるデータに対しても十分に対応できていることがわかる．

4.3 キャパシティプランニング

最後に，モデル化したデータを用いて，Kubernetes 上に可動するコンテナ群全体でのキャパシティプランニングを行う例を紹介する．Kubernetes においては，コンテナをどこで起動するノードを決定するときに，設定されたラベルやデータの配置，GPU の有無等を考慮してスケジューリングされるが，ユーザが設定するコンテナのリクエスト値も重要なファクターの一つである．リクエスト値は，コンテナが使うことが期待されるリソースを予約する意味合いを持つため，そのノード上のコンテナの多くが本来であれば不要なほどにリソースを予約していてる場合，実際にはスケジュール可能であっても，そのノードにはスケジュールされなくなってしまう．

図 5 は，コンテナごとに設定されたメモリリクエスト値，リミット値のノードごとの値を，初期状態 (before) と実際

図 5 Resource Quota 最適化前後でのノードごとの Resource Requests/Limits 値の比較

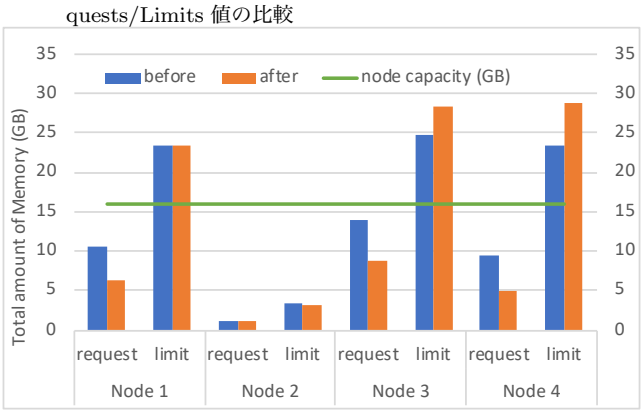
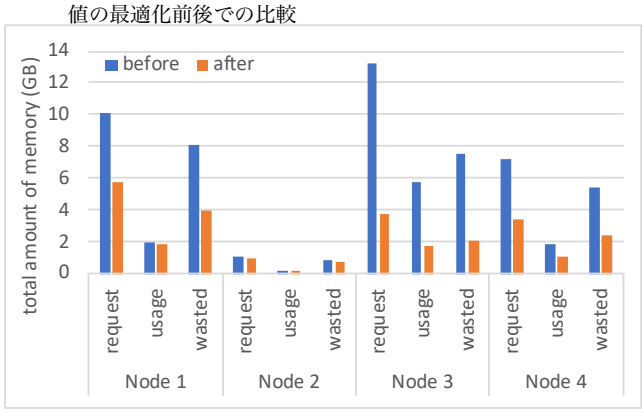


図 6 オーバーコミット状態のサービスに対する Resource Requests 値の最適化前後での比較



の状況に合わせてそれぞれの値をモデルを用いて最適化した状態 (after) を比べたものである．リクエスト値を比較するとと，Node 2 以外の全てのノードで必要以上にリクエスト値が設定されていたことが分かる．実際のモデルにあわせると，30-45%ほどのリクエスト値を減少させることができ，とりわけ Node 3 のようなケースでは，14GB から 8GB まで減ったことにより，リソースの空きを生み出すことができるため，さらにコンテナをデプロイすることが可能となる．

図 6 は，メモリリクエスト値のみに注目し，実際のメモリ利用状況よりも多くリクエスト値が設定されている状態（オーバーリクエストコミット状態）のサービスに対して，どの程度実際にはリクエスト値を減らすことができたのかにフォーカスした結果である．Node 3 においては，非常に多くのサービスが多めにメモリを確保しているが，実際に使われているメモリは非常に少なく，トータルで 8GB ほどのオーバーコミット状態となっているが，最適化後では，2GB 程度まで減らすことができている．

図 5 のリミット値については，Node 3 および Node 4 において上昇している．これは，実際には多くのコンテナがリミット値をセットしていない，すなわち，実質ノードの物理メモリを全て使い切る可能性があることを意味しており，これらのノードに備わった 16GB が実質リミット値としてセットされている状況と同義である．そのため，初期状態のリミット値の総和よりも実際の最適化されたリミット値のほうが，見かけ上多くなっている．リミット値はスケジューリングそのものには影響しないが，適切にセットしない場合，ノードのメモリを使い果たして，OOM Killer や Swap の発生の原因となり，そのコンテナのみならず他のコンテナに対しても著しい性能低下を及ぼす可能性があるため，リクエスト値同様重要である．

5. 関連研究

ML/DL をベースにクラウド上アプリケーションや VM のモデリングを行い，オーバープロビジョニング [10] やリソーススケジューリングに応用したり，SLA や QoS を達成する最適化を行う研究がいくつか行われている．文献 [11] では，コンテナの CPU 利用率を予測して SLA を達成するために，モデルに用いる特徴量を選択してニューラルネットワークベースのモデルを構築する手法を提案している．また，文献 [11] においても，VM における SLA Violation を防ぐために，リソース利用率を LSTM ベースのモデルで予測し，VM リソースアロケーションを最適化する手法を提案している．文献 [12] では，VM やジョブのリソースアロケーションを最適化するために，過去の実行履歴やトレース情報を特徴量として，Random Forest や SVM，XGBoost など様々な機械学習アルゴリズムを用いてスケジューリングポリシーを決定する分類器を作成し比較を行っている．

文献 [13] では，個々のマイクロサービスに対する負荷テストを通じて生成されたリクエスト量と CPU 利用率から最適なレプリカ数を求めるための回帰モデルを作成し，キャパシティプランニングに用いる手法を提案している．文献 [14] では，QoS Violation を検知するためのアプローチとして CNN と LSTM を用いており，マイクロサービス全体のレスポンスタイムや性能を改善するため，アプリケーションのトレースデータを用いて構築されたモデルにより将来の QoS Violation を予測する手法を提案している．

6. おわりに

本稿では，スケジューリングやキャパシティプランニングなどの最適化を Elasitic に行うために必要な性能モデリングに関して，Kubernetes 上で実行されるマイクロサービスから取得可能な様々なメトリクスをもとに時系列モデリングを行う手法について検討し，Kubernetes 上で動作するモデリングサービスとして実装した．また，LSTM を用いてモデリングした結果をもとに Kuberenetes 上で動いている様々なサービス全体のリソースクォータを最適化し，最大で 45%程度の余分なリソースリクエストを削減できることを確認した．今後の課題としては，モデルを用いた異

常検知や，より超長期の性能予測などを行う予定である．

参考文献

[1] Fazio, M., Celesti, A., Ranjan, R., Liu, C., Chen, L. and Villari, M.: Open Issues in Scheduling Microservices in the Cloud, *IEEE Cloud Computing*, Vol. 3, No. 5, pp. 81–88 (online), DOI: 10.1109/MCC.2016.112 (2016).

[2] Verma, A., Pedrosa, L., Korupolu, M. R., Oppenheimer, D., Tune, E. and Wilkes, J.: Large-scale cluster management at Google with Borg, *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France (2015).

[3] Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J.: Borg, Omega, and Kubernetes, *ACM Queue*, Vol. 14, pp. 70–93 (online), available from ⟨http://queue.acm.org/detail.cfm?id=2898444⟩ (2016).

[4] : Horizontal Pod Autoscaler, `https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/`.

[5] López, M. R. and Spillner, J.: Towards Quantifiable Boundaries for Elastic Horizontal Scaling of Microservices, *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, UCC '17 Companion, New York, NY, USA, ACM, pp. 35–40 (online), DOI: 10.1145/3147234.3148111 (2017).

[6] Dean, J.: Machine learning for systems and systems for machine learning, *Neural Information Processing Systems* (2017).

[7] Gao, J.: Machine Learning Applications for Data Center Optimization (2014).

[8] Ergen, T. and Kozat, S. S.: Efficient Online Learning Algorithms Based on LSTM Neural Networks, *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 29, No. 8, pp. 3772–3783 (online), DOI: 10.1109/TNNLS.2017.2741598 (2018).

[9] Chiba, T., Nakazawa, R., Horii, H., Suneja, S. and Seelam, S.: ConfAdvisor: A Performance-centric Configuration Tuning Framework for Containers on Kubernetes, *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 168–178 (2019).

[10] Zhou, H., Chen, M., Lin, Q., Wang, Y., She, X., Liu, S., Gu, R., Ooi, B. C. and Yang, J.: Overload Control for Scaling WeChat Microservices, *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '18, New York, NY, USA, ACM, pp. 149–161 (online), DOI: 10.1145/3267809.3267823 (2018).

[11] Tang, X., Liu, Q., Dong, Y., Han, J. and Zhang, Z.: Fisher: An Efficient Container Load Prediction Model with Deep Neural Network in Clouds, *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pp. 199–206 (online), DOI: 10.1109/BDCloud.2018.00041 (2018).

[12] Yang, R., Ouyang, X., Chen, Y., Townend, P. and Xu, J.: Intelligent Resource Scheduling at Scale: A Machine Learning Perspective, *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pp. 132–141 (online), DOI: 10.1109/SOSE.2018.00025 (2018).

[13] Jindal, A., Podolskiy, V. and Gerndt, M.: Performance Modeling for Cloud Microservice Applications, *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ICPE '19, New York, NY, USA, ACM, pp. 25–32 (online), DOI: 10.1145/3297663.3310309 (2019).

[14] Gan, Y., Zhang, Y., Hu, K., Cheng, D., He, Y., Pancholi, M. and Delimitrou, C.: Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices, *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, New York, NY, USA, ACM, pp. 19–33 (online), DOI: 10.1145/3297858.3304004 (2019).