リソース量推論モデル向けテスト実行自動化ツールの提案と評価

水野 和彦† 佐伯 裕治‡

†日立製作所研究開発グループ 〒185-8601 東京都国分寺市東恋ヶ窪港 1 丁目 280

E-mail: †{ kazuhiko.mizuno.pq, yuji.saeki.fz}@hitachi.com

あらまし アプリケーションを最適なロケーションに配置する場合、各アプリの要求性能を満たすリソース量を推論モデルで求め、リソース割り当て可能なデプロイ先を選定することが一般的である。しかし、推論モデルの作成には種々の条件下でのアプリケーションの稼働情報を得る必要があり、そのためのテスト工数が膨大になることが課題となる。そこで本研究では、著者らの先行研究であるリソース量推論モデルを対象として自動的に種々の条件下に合わせたテスト実行を行うテスト実行自動化ツールを提案した。解析システムを開発環境でテスト実行するユースケースに対して当該ツールによるテスト工数の削減効果を机上評価した結果、テスト工数を４７％削減できる見込みを得た。

キーワード 時系列データ処理、評価・データセット、テスト実行自動化

1. はじめに

近年，ビッグデータならびに分析ソフトウェアを扱う市場は急速な拡大を続けており，2018 年に World Wide で約$60B の市場規模に達し，2019-2023 年の CAGR (Compound Annual Growth Rate) は 12.5%と予測されている[1]。この市場の伸びは，日々の業務で発生する大量のデータの蓄積と，様々な観点での分析結果の活用による新しい付加価値の創出で裏打ちされている。また，近年特に規模が拡大している IoT 市場[2]においても同様のことが言える。

また，パブリッククラウドサービスプロバイダが提供するパブリッククラウドサービスの利用も広がってきている。これに伴い，工場の製造ラインと連携した現場における装置監視やデータ収集システムと，オンプレミス環境で利用していた IT(Information Technology)インフラ設備で稼働していた IT システムと，パブリッククラウドサービスを連携させたハイブリッドクラウド環境によるデータ利活用環境を構築し，利用するケースも増えてきている。パブリッククラウドサービスが提供する柔軟かつ迅速なリソース提供を活用することで，新規の各種サービス向けのシステムを過大な設備投資をすることなくスモールスタートすることも可能になっている。

パブリッククラウドサービスは，pay-per-use の課金モデルに基づいて，比較的短期間で多量のリソースを一時的に利用するケースにおいては費用対効果の高い環境である。しかし，中長期にわたって一定量のリソースを継続的に利用するケースにおいては割高になるケースもある。このために，現有の IT インフラのリソースとパブリッククラウドサービスのリソースを効率よく使いこなすことも求められるようになってきている。

著者らは、このような状況下において，ハイブリッドクラウド環境におけるデータストレージリソースやコンピューティングリソースを賢く使いこなすサービスを提供することが，利用者に対して様々な価値を提供することができ有益と考えており、当該サービスをデジタルインフラ面で支えるサービスやソリューション実現のための検討を進めている。本検討では，データストレージ環境ならびにコンピューティング環境のインフラ情報ならびに利用アプリケーション（以下，アプリと呼ぶ）などの稼働構成情報をもとに，アプリやインフラをモデル化し，アプリの要求性能を満たすリソース量を求め、アプリ実行環境となるコンテナやデータの最適なロケーションへの配置を立案し，最適配置制御できるようにする。また、アプリコンテナやデータストアが利用するリソースの割り当て場所やサイズの最適化による IT システム運用効率化とその IT システムを利用した各種業務の効率化の両立をめざす。

本研究では、ハイブリッドクラウド環境において、各アプリの要求性能を満たすリソース量を求める推論モデルを作成するために必要な情報を収集するテスト実行の検討を推進し、特に様々な条件下でのアプリの稼働情報を取得するテスト実行手順の自動化の検討を推進した。本稿では，テスト実行手順の自動化として提案するテスト実行自動化ツールの原理検証に関して報告する。

2. テスト実行自動化の課題

2.1 リソース量推論モデル

ハイブリッドクラウド環境において，各アプリが稼働するコンテナ実行環境向けのコンピュートリソースとデータストア環境におけるデータ配置を適正に制御することが必要不可欠となる。

著者らは、ハイブリッドクラウド環境において、各アプリの要求性能を満たすコンピュータリソースやデ

ータ配置に関するモデル化を行い、当該モデルによりアプリを最適なロケーションに配置する最適配置制御の検討を推進している。

著者の先行研究では、アプリの要求性能を満たすコンピュータリソースを求めるリソース量推論モデルを提案した。当該モデルで推論したアプリの要求性能を満たす CPU コア数を実際に対象アプリに割り当て, 対象アプリの処理時間を実機環境で実測した所，推論した CPU コア数であれば処理時間を維持することが確かめられ, 推論した CPU コア数未満を割当て際に処理時間が CPU 不足で大きくなることを確かめられたことを報告している[3]。

リソース量推論モデルでは，様々な条件を変更しながら利用アプリの稼働情報を取得するテスト実行によりモデル作成を行う。例えば, 利用アプリが Kubernetes の Pod 上で稼働する場合，テスト実行時のパラメータとして Pod に割り当てる CPU コア数やメモリ量を対象とし，このパラメータを変更しながらテスト実行を行って，利用アプリの実行時間を計測する。この計測結果に基づいて，CPU コア数やメモリ量に対する利用アプリの実行時間を推論するモデルが作成される。

著者らは、様々な条件を変更しながら利用アプリの稼働情報を取得するテスト実行手順を容易に行うことを目的としてテスト実行手順の自動化に関する検討を推進している。

2.2 テスト実行の想定ユースケース

テスト実行手順の自動化を効率よく適用するためには、各手順を具体化し，どの手順において自動化により利用者の作業負担を軽減できるか確認する必要がある。そこで，テスト実行手順を具体化するための想定ユースケースを立案した。想定ユースケースでは，テスト実行対象となる利用アプリとして解析システムを想定し，事前にプライベートリポジトリに実装された利用アプリが登録されていることを想定する。以降、想定ユースケースについて説明する。なお、図 1 に想定ユースケースの概要を示す。

業務システム開発者は，利用アプリをテスト環境にデプロイするためのパッケージ化を行う（図 1 - a）。ここでは，利用アプリのデプロイに Kubernetes を利用することとして, Kubernetes の Pod/コンテナ化として，利用アプリのデプロイ用設定ファイル(マニフェスト)の作成，デプロイ用のイメージファイルの準備，および利用アプリの各処理を実行するクライアントプログラムの準備を行う。利用アプリで想定される各処理としては，データ登録処理，学習処理，推論処理などが考えられる。

IT システム管理者がインフラ整備やデプロイ準備を行ったテスト環境に、業務システム管理者が準備した利用アプリのマニフェストファイルやイメージファイルパッケージを利用して，業務システム運用者がテスト環境に利用アプリをデプロイする（図 1 - b）。利用アプリは，テスト環境の Kubernetes の Pod／コンテナ上にデプロイされ，業務システム運用者により動作確認が行われる。

業務システム運用者が利用アプリの実行条件となるテストパラメータ，およびテストパラメータの組合せ(テストケース)を立案して利用アプリのテスト実行準備を行う（図 1 - c）。このテストパラメータには，例えば，CPU コア数やメモリ量などの割当リソース量、各コンテナなどを設定するデプロイ情報、利用アプリの実行処理内容などを設定するアプリ固有情報などが利用される。例えば, 割当リソース量などについては，利用アプリに高負荷がかかるように上限値，下限値，刻み幅(サンプル数)を設定し，利用アプリの処理内容などには，リソース量推論モデルの作成要件に合わせて対象コンテナや対象処理を設定する。

テスト実行準備完了後、業務システム運用者がテストケースに合わせて利用アプリのテスト実行をテスト環境で行う（図 1 - d）。このテスト実行では，テストケースに合わせた利用アプリの Pod/コンテナの構成変更，クライアントプログラムによる利用アプリの各処理の実行，および利用アプリの実行時間や稼働情報の取得を，全てのテストケースが終了するまで繰り返し実行する。

業務システム運用者が利用アプリのテストケース，および利用アプリの実行時間や稼働情報などのテスト実行結果をコントロールプレーンに提供する（図 1 - e）。これには，例えば，Gitlab を利用することで，利用アプリのテスト実行結果をコントロールプレーンで所定のフォルダに提供することができる。

リソース量推論モデルの作成に利用する情報を抽出するために、テスト実行結果に対してデータ前処理を行う（図 1 - f）。このデータ前処理で作成されたテスト実行結果の加工・統合データが所定フォルダに格納される。また，データ前処理には，各モデル作成に合わせて事前に登録された加工・統合ルールが利用される。

テスト実行結果，およびテスト実行結果の加工・統合データからリソース量推論モデルを作成する（図 1 - g）。ここで作成されるモデルは回帰式であり，例えば，リソース量推論モデルであれば，利用アプリの割当リソース量に対する実行時間を求める回帰式が作成される。

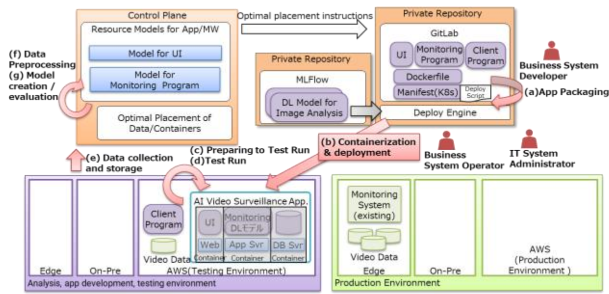

図 1　想定ユースケース概要

2.3 テスト実行の課題とアプローチ

想定ユースケースで具体化したテスト実行手順の（c）、（e）の手順に着目した課題，および課題に対する自動化へのアプローチついて説明する。

利用アプリのテスト実行準備（図 1‐c）において、第 1 に、利用アプリに高負荷がかかる状態でテスト実行を行い、そのときの実行時間や稼働情報を取得したいが、利用アプリに高負荷をかけられるテストケースを業務システム運用者が把握できないため，適したテストケースを選定できない可能性があり，テスト実行の作業時間が増加する課題が考えられる。

当該課題については、事前に行われる利用アプリ本体の性能検証や動作検証で得られた知見からテストパラメータを選定し、自動的にテストケースを作成することで、テスト実行の作業時間の短縮が期待できる。例えば，デプロイ情報となる割当リソース量の場合には，上限値に高負荷がかかる状態であっても十分にアプリの要求性能を保証できる値を設定し，下限値に利用アプリが最低限動作可能な値を設定する。テストパラメータのサンプル数については，リソース量推論モデルの作成で得られた知見から妥当な値を設定する。これら手順によるテストケースを自動作成させることで，業務システム運用者にテストケースの作り方などの必要な手順を意識させる必要がない。

第 2 に、業務システム運用者が手動で Pod や Replicaset の構築，ネットワーク(Service)設定，永続ボリューム(Persistent Volume)の指定など様々な設定項目が定義されるデプロイ用マニフェストファイルから、対象 Pod/コンテナの割当リソース量を定義する項目の特定、その項目に対する割当リソース量に制限をかけるように定義や設定値の変更、および不要な定義の削除といった一連の作業を全テストケースに対して行うことになるため，テストケースの総数に比例して，業務システム開発者の作業工数が増加する課題が考えられる。

当該課題については、マニフェストファイルの変更作業を自動化することで、業務システム開発者は，テストケースと利用アプリのデプロイ用マニフェストファイルを準備するのみとなり，それ以外のデプロイ用マニフェストファイルを変更する一連の作業が自動的に行われることになるため、業務システム開発者の作業工数の短縮が期待できる。

第 3 に、利用アプリのテスト実行では，デプロイ情報やアプリ固有情報などのテストパラメータ，および個々のテストパラメータのサンプル数などがテストケースとなるため，テストケースの増加に比例して，業務システム運用者が手動でテストケースに合わせた利用アプリの Pod/コンテナの構成変更，クライアントプログラムによる利用アプリの各処理の実行，および利用アプリの実行時間や稼働情報などのテスト実行結果の取得などの実行時間が増加するためテスト実行時間が長時間となる課題が考えられる。

当該課題については、テスト実行時に変更するパラメータを自動的に設定させることで各手順の手間を省けるため業務システム運用者の負担を軽減でき､かつ、テスト実行時間の短縮が期待できる。更にテスト実行時間については，個々のテストケースに合わせたテスト実行を並列に処理することでテスト実行時間の削減が期待できる。

テスト実行結果のデータ収集・格納（図 1‐e）において、第 4 に、テスト実行方法や稼働情報などの出力方法によって，利用アプリのテスト実行結果の出力形式が異なることが想定されるため、対象となる利用アプリ，テスト実行に利用したテストケース，および利用アプリのテスト実行結果の関連付けを適切に行えずリソース量推論モデルの作成に利用したい情報を容易に検索できない課題が考えられる。

当該課題については、テスト実行方法や出力方法を共通化し、テスト実行に関する各種情報(利用アプリ，テストケース，テスト実行結果)を紐づけてテスト実行履歴情報として自動管理することによりデータ検索を容易に行えることが期待できる。

3. テスト実行自動化ツールの提案

3.1 テスト実行自動化ツールの提供機能

テスト実行手順の 4 つの課題に対して自動化によるアプローチにより課題解決を実現するテスト実行自動化ツールを提案する。当該ツールは自動化のアプローチを 5 つの機能で提供する。

第 1 の機能として、自動的にテストケースを作成してテスト実行の作業時間を短縮させるテストパラメータの組合せ一覧(テストケース)作成管理機能を提供する。当該機能では，テストパラメータとしてアプリのデプロイ時に設定する CPU コア数やメモリ容量などの割当リソース量となる共通パラメータ，および，学習データ数や推論データ数などのアプリ固有パラメー

タを設定する。それぞれのパラメータには，上限値，下限値，サンプル数を業務システム運用者が指定することとし，この指定値に合わせてテストケースを自動作成する。当該機能により，テストケースを指定する手間を簡略化できテストケースの設定を容易に行うことができる。

第 2 の機能として、デプロイ用マニフェストファイルを変更する一連の作業を自動的に行うことで業務システム開発者の作業工数を短縮させるマニフェストファイル作成機能を提供する。当該機能では，アプリのデプロイ用マニフェストファイルを取得し，テストパラメータの組合せ一覧(テストケース)作成管理機能で作成したテストケースの内，共通パラメータを制限値として当該マニフェストファイルの割当リソース量に変更したマニフェストファイルをテストケース毎に自動作成する。当該機能により，マニフェストファイルの変更作業を簡略化でき作業工数を削減できる。

第 3 の機能として、テスト実行時にテストケースに合わせて利用アプリのデプロイを自動的に行うことで業務システム運用者の負担を軽減させる割当リソース量変更機能を提供する。当該機能では、マニフェストファイル作成機能で自動作成されたマニフェストファイルにより自動的に利用アプリのデプロイを行い、利用アプリのテスト実行が完了した後に利用アプリの削除を行う。当該機能により、業務システム運用者のデプロイ作業に要する手間を軽減できる。

第 4 の機能として、クライアントプログラムによる利用アプリの各処理の実行自動的に行うことで業務システム運用者の負担を軽減させるアプリ処理実行機能を提供する。当該機能では、アプリ固有パラメータに合わせて利用アプリの各処理の実行、各処理の実行時間の計測、および利用アプリの稼働情報の取得などを自動的に実行する。当該機能により、業務システム運用者のテスト実行作業に要する手間を軽減できる。

第 5 の機能として、テスト実行に関する各種情報を自動管理することでデータ検索を容易化するデータ収集・管理機能を提供する。当該機能では、利用アプリの実行時の稼働情報や実行時間などの実行結果をテストケースの識別子(ID)に紐づけて所定フォルダに出力する。当該機能により、テストケースの識別子に紐付いて各情報が管理されるためデータ検索を容易に行うことができる。

3.2 テスト実行自動化ツールのアーキテクチャ

テスト実行自動化ツールでは，テストパラメータの組合せ一覧(テストケース)作成管理機能，マニフェストファイル作成機能，割当リソース量変更機能，アプリ処理実行機能，およびデータ収集・管理機能が提供される。テスト実行自動化ツールのアーキテクチャを図 2 に示す。

テスト実行自動化ツールは，テスト環境に実装されて、Kubernetes のデプロイエンジンで，プライベートリポジトリに登録される利用アプリのイメージファイルやマニフェストファイルを読み込み、テスト環境にテストケースに合わせた利用アプリをデプロイする（図 2－ａ）。

この利用アプリの各処理は、テスト実行自動化ツールがクライアントプログラムで提供される API で実行され、その時の利用アプリの稼働情報や実行結果が取得される（図 2－ｂ）。このテスト実行で取得した稼働情報や実行結果については，コントロールプレーンに提供してリソース量推論モデルの作成に利用される（図 2－ｃ）。

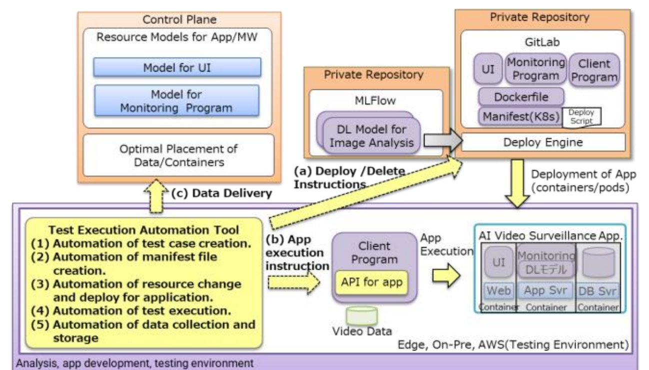

図 2　テスト実行自動化ツールの想定アーキ

テスト実行自動化ツールの実行処理について、以降説明する。なお，当該ツールの処理フローを図 3 に示す。

テスト実行自動化ツールの事前準備（図 3－①）では，まず，業務システム開発者が，利用アプリのデプロイ用マニフェストファイルの準備，およびテスト対象となる利用アプリの処理内容を確認する。

次に、業務システム開発者や業務システム運用者が，最も負荷が掛かる利用アプリの利用方法を想定したテストシナリオの検討、テストシナリオに合わせたテストパラメータの組み合わせ(テストケース)の策定、およびテストケースに合わせて利用アプリを実行するクライアントプログラムの作成と動作確認を行う。

最後に，業務システム運用者が，テスト環境で利用アプリのテスト実行を行えるように，インフラ環境の構築，テスト実行自動化ツールのセットアップ，利用アプリの稼働情報を取得する OSS の準備，クライアントプログラムやテスト実行自動化ツールの動作確認を行う。また，テスト実行自動化ツールのセットアップでは，利用アプリ，利用するテストパラメータ，利用

アプリのデプロイ用マニフェストファイルなどを登録した設定ファイルの準備，および当該ツールの利用フォルダの準備を行う。

テストパラメータの組合せ一覧(テストケース)の自動作成処理（図 3－②）の入力として，テスト実行自動化ツールの設定ファイルを読み込み，利用アプリのデプロイ時に設定する CPU コア数やメモリ容量などの割当リソース量となる共通パラメータ，および，学習データ数や推論データ数などのアプリ固有パラメータを取得する。また，各テストパラメータの情報として上限値，下限値，サンプル数などを合わせて取得する。

当該処理では，テストパラメータの組合せ一覧(テストケース)作成管理機能を呼び出し、まずは、それぞれのテストパラメータの値に，テストパラメータの上限値から下限値までの範囲で，サンプル数で刻んだ値を設定する。次に、この設定したテストパラメータを組合せ、個々のテストケースを示す識別子，テスト実行の間隔時間，利用アプリのデプロイ用マニフェストファイル情報，デプロイ時のネームスペース情報，および各種情報の出力先フォルダ情報がテストケースとして登録される。

当該処理の出力として，当該テストケースを所定フォルダに JSON 形式のテストケースファイルとして出力する。

利用アプリのデプロイ用マニフェストファイルの自動作成処理（図 3－③）の入力として，テストケースファイルを読み込み，利用アプリのデプロイ条件となる CPU コア数やメモリ容量となる共通パラメータ、および利用アプリのデプロイ用マニフェストファイルを取得する。

当該処理では，マニフェストファイル作成機能を呼び出し、利用アプリのデプロイ用マニフェストファイルをベースに，CPU コア数やメモリ容量に関する定義部分に,デプロイ条件となる CPU コア数とメモリ容量を利用上限値として定義することでデプロイ条件以上のコンピュートリソースを利用できないように制限をかける。

当該処理の出力として，全テストケースに合わせて作成した利用アプリのデプロイ用マニフェストファイルを所定フォルダに出力する。

全てのテストケースのテスト実行を完了するまで繰り返し処理（図 3－④）では，以降の手順を繰り返し実行する。

利用アプリのデプロイ処理（図 3－⑤）の入力として，テストケースファイルを読み込み，デプロイ先となる Kubernetes のネームスペース情報、および利用アプリのデプロイ用マニフェストファイルを取得する。

当該処理では，割当リソース量変更機能を呼び出し、利用アプリのデプロイ用マニフェストファイルを利用して，Kubernetes のネームスペースに利用アプリをデプロイする。また，デプロイの完了確認後に，利用アプリの実行処理を呼び出す。

利用アプリの実行処理（図 3－⑥）の入力として，テストケースファイルを読み込み，利用アプリの実行時に指定する学習データ使用率などのアプリ固有パラメータを取得し，利用アプリの実行結果を出力する所定フォルダの情報を取得する。

当該処理では，アプリ処理実行機能を呼び出し、まず,利用アプリの実行前に必要なセットアップを行う。例えば,利用アプリの各処理を API で実行するために，利用アプリが稼働する Pod に対してポートフォワード設定を行う。次に，学習データ使用率などのアプリ固有パラメータを利用して，利用アプリの各処理を実行する。各処理の実行時にはタイムスタンプを取得して各処理の実行時間を計測する。最後に，利用アプリの稼働情報の取得処理を呼び出す。

当該処理の出力としては，利用アプリの実行結果として，利用アプリの各処理の実行時間を計測し登録したファイルを所定フォルダに出力する。

利用アプリの稼働情報の取得処理（図 3－⑦）の入力としては，テストケースファイルからテストケースの識別子を取得し，⑥の処理で得られるタイムスタンプから利用アプリの実行開始時刻と実行終了時刻を取得する。

当該処理では，データ収集・管理機能を呼び出し、利用アプリの実行開始時刻から実行終了時刻に取得した稼働情報を収集する。具体的には，Kubernetes の稼働情報を取得する Prometheus[4]を事前に稼働させておき，この Prometheus から実行開始時刻から実行終了時刻までの稼働情報を取得する。稼働情報では，Pod の CPU 実行時間やメモリ利用量などを取得できる。

当該処理の出力として，取得した稼働情報を登録したファイルをテストケースの識別子(ID)に紐づけて所定フォルダに出力する。

利用アプリの削除処理（図 3－⑧）の入力としては，テストケースファイルを読み込み，デプロイ先となる Kubernetes のネームスペース情報、および利用アプリのデプロイ用マニフェストファイルを取得する。

当該処理では，割当リソース量変更機能を呼び出し、利用アプリのデプロイ用マニフェストファイルを利用して，Kubernetes のネームスペースにデプロイされた利用アプリの削除を行い，利用アプリの削除が完了するまで待機する。

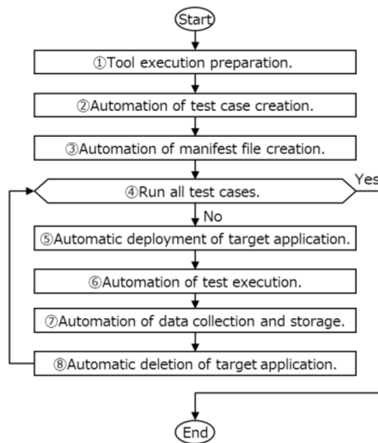

図 3　テスト実行自動化ツールの処理フロー

テスト実行自動化ツールでは，上述した一連の流れを実行することで,マニフェストファイルの変更作業，テストケースの指定，およびテストケースの設定を容易化でき，テストケースに合わせた利用アプリの実行結果の取得などの各処理を自動的に繰り返し行うことで，テスト実行手順の手間を省き，自動化によりテスト実行時間を削減できる。

4. テスト実行自動化ツールの原理検証

4.1 テスト実行自動化ツールの検証方法

想定ユースケースで具体化したテスト実行手順にテスト実行自動化ツールを適用した際の効果を検証するために，実機環境を利用した原理検証を実施する。なお，テスト実行対象の利用アプリを図 4 に示す。また，検証項目を表 1 に示す。

実機環境には，仮想サーバにテスト実行対象となる利用アプリ，テスト実行手順の自動化を行うテスト実行自動化ツール，利用アプリのメトリクス情報を取得する Prometheus を構築する。利用アプリは，Kubernetes の Pod/コンテナで起動し，Pod/コンテナのコンテナポートに対してポートフォワード設定を行うことで，クライアントプログラムで利用アプリを実行する。テスト実行自動化ツールは，Python ベースのプログラムであり，Kubernetes Client[5]のライブラリを利用することで，Kubernetes の Pod/コンテナのデプロイや削除を行うことできる。Prometheus では，コンテナのメトリクス情報を取得する cAdvisor[6]とノードのメトリクス情報を取得する Node_exporter[7]から各情報を収集し，API を実行することで任意の時間帯のメトリクス情報を抽出できる。

利用アプリでは，scikit-learn サンプルデータである 20 の話題に関する 18000 のニュース記事データセット[8]を Elasticsearch に登録し，これらのニュース記事のデータ使用率を指定して当該データの学習処理と，トピックの分類を行う推論処理を実施する。データの登録，学習処理，および推論処理は，クライアントプログラムに実装された API により実行することができる。

上述した実機環境による検証では、まず，テスト実行自動化ツールの効果検証として，テスト実行自動化ツールが，どの程度の実行時間で完了することが可能か確認する(表 1－＃１)。当該検証では，テスト実行自動化ツールの実行時間を計測するために，利用アプリが正常に処理できるリソース量を割当て，利用アプリの処理が短時間で完了するようにパラメータを指定する。具体的には，テストパラメータには，CPU コア数に 1，2 コア，メモリ容量に 9000，10000ＭＢ，学習/推論データ使用率に 5%を指定する。

次に，テスト実行自動化ツールの適用時におけるテスト実行手順の削減効果を検証する(表 1－＃２)。当該検証では，想定ユースケース（図 1）をベースに業務システム開発者と業務システム運用者のテスト実行手順を洗い出し，テスト実行自動化ツールの適用により業務システム開発者と業務システム運用者が行うテスト実行手順がどの程度削減できるかを机上検証する。業務システム開発者と業務システム運用者のテスト実行手順の対象は，図 1 の手順（ａ）から（ｅ）までとし，手順（ｆ）以降は，リソース量推論モデルの作成処理となるため当該検証から割愛する。

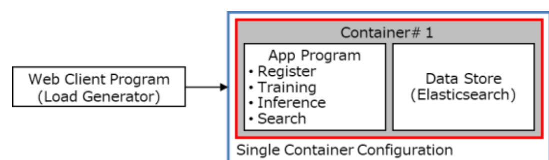

図 4　利用アプリの概要

表 1　検証項目

#	Item	Description
1	Verification of app execution time.	Measurement of application execution time by verification environment.
2	Verification of test execution procedure.	Verification of effectiveness by automating test execution procedures

4.2 テスト実行自動化ツールの検証結果

実機環境でテスト実行自動化ツールの実行時間を計測した結果を表 2 に示す。

テスト実行自動化ツールの実行時間については，CPU コア数が 1 の時に約 158 秒（表 2－＃３），CPU コア数が 2 の時に約 127 秒（表 2－＃６）となる結果を得た。以降では，テスト実行自動化ツールの実行時間の内訳について説明する。

テストパラメータの組合せ一覧(テストケース)の自

動作成処理，および利用アプリのデプロイ用マニフェストファイル作成処理については殆ど時間をかけずに処理が完了した。これらの処理については，テストパラメータの組合せに比例して増加することが推測できるが，テスト実行自動化ツール全体の実行時間に比べると非常に小さな値になると考えられる。

利用アプリの実行時間については，CPU コア数が 1 の時に約 98 秒で完了し，CPU コア数が 2 の時に約 61 秒で完了した。これは、CPU コア数の増加により利用アプリ処理能力が向上したため利用アプリの実行時間が短縮したと推測できる。また、測定結果から利用アプリの実行時間がテスト実行自動化ツールの全体の実行時間の約 62%を占めることが分かった。

利用アプリのデプロイ処理は約 5 秒、利用アプリの稼働情報の取得処理は約 9 秒、利用アプリの削除処理は約 50 秒で完了した。また、それぞれの実行時間についてはテストパラメータの値に関係なく，定常的な実行時間となることが分かった。

以上の結果からテスト実行自動化ツールの実行時間を見積もること可能と考えている。例えば，テストケースの 1 パターンを実測することで，利用アプリのデプロイ処理時間，利用アプリの稼働情報の取得処理時間，利用アプリの削除時間はテストパラメータによらず定常的であるので実測した値をそのまま当該ツールの実行時間の見積もりに利用でき，これに，利用アプリの実行時間を考慮することで，テストケースの全体を実行する際のおおよその完了時間を見積もることできる。

表 2　テスト実行時間の計測結果

#	Test Parameter			Processing of test execution automation tool	Measured value [sec]
	CPU[core]	Memory[MB]	Data Usage[%]		
1	–	–	–	Automation of test case creation.	0.000
2	–	–	–	Automation of manifest file creation.	0.001
3	1	9000	5	Automatic deployment of target application.	5.106
				Automation of test execution.	97.315
				Automation of data collection and storage.	9.216
				Automatic deletion of target application.	46.510
				Total execution time	158.148
4	1	10000	5	Automatic deployment of target application.	5.135
				Automation of test execution.	98.058
				Automation of data collection and storage.	9.144
				Automatic deletion of target application.	46.440
				Total execution time	158.778
5	2	9000	5	Automatic deployment of target application.	5.170
				Automation of test execution.	61.855
				Automation of data collection and storage.	8.393
				Automatic deletion of target application.	41.409
				Total execution time	116.827
6	2	10000	5	Automatic deployment of target application.	5.204
				Automation of test execution.	61.406
				Automation of data collection and storage.	8.437
				Automatic deletion of target application.	51.631
				Total execution time	126.679

テスト実行自動化ツールを適用した時のテスト実行手順の削減効果を図 5 に示す。図 5 では、左側に従来の業務システム開発者と業務システム運用者のテスト実行手順を示し，右側に業務システム開発者と業務システム運用者の作業に対してテスト実行自動化ツールを適用した際のテスト実行手順を示す。以降では，テスト実行手順の各ステップについて説明する。

業務システム開発者は，主に 3 ステップの手順を行い（図 5－a），テスト実行対象となる利用アプリの選定，利用アプリのデプロイ用マニフェストファイルの作成，および利用アプリの処理内容の確認を行う。

業務システム運用者は，主に 12 ステップの手順を行う。利用アプリのデプロイとして（図 5－b），利用アプリのパッケージ検索，テスト環境への利用アプリのデプロイ，利用アプリの動作検証を行う。テスト実行準備として（図 5－c），最も負荷がかかる実運用を想定したテストシナリオの検討，テストシナリオに合わせたテストケースの策定，利用アプリの各処理を実行するクライアントプログラム作成，テスト実行自動化ツールの準備を行う。テスト実行として（図 5－d），利用アプリのデプロイ、実行、および削除を行う。データ収集として（図 5－e），テストケース毎に利用アプリの実行結果を取得する。

テスト実行自動化ツールを適用した場合には，上述した各手順において，テスト実行準備，テスト実行，および,データ収集を自動的に行うことが可能であり，手動でのテスト実行手順の 15 ステップに対して，7 ステップを自動化でサポートができる。この結果からテスト実行自動化ツールの適用によりテスト実行手順を約 47%削減できる見込みを得た。

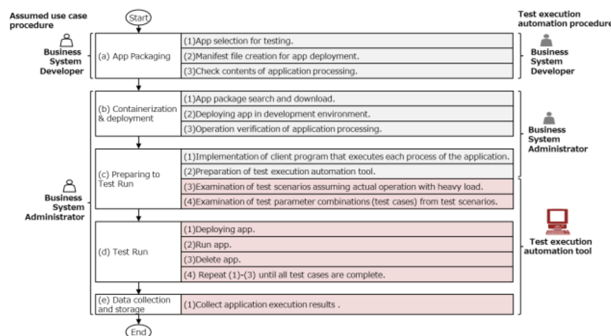

図 5　テスト実行手順の削減効果

5. 考察

本研究では、ハイブリッドクラウド環境において、各アプリの要求性能を満たすリソース量を求めるリソース量推論モデルを作成するために必要な情報を収集するテスト実行に関する検討を推進し、特に様々な条件下でのアプリの稼働情報を取得するテスト実行自動化ツールを提案した。

アプリをテスト実行結果からモデル化する手法は，以下のように分類できる。

(1)資源使用制限値を変えたテスト実行時のアプリ性

能を測定してモデル化

(2)アプリ運用時の資源使用量データの時系列データ解析により資源使用パターンをモデル化

(3)アプリ開発者がソースコードの構造をもとに実行時の振舞を再現する性能モデルを作成

(4)アプリ実行時の振舞を再現するシミュレータを作成

(1)では，デプロイ前にテスト実行を行ってモデルを作成し,事前にアプリ性能を見積もることによって，運用開始時に利用者の要件に合わせたアプリ配置最適化を可能とする。

(2)の例としては，IBM 等による AI4DL[9][10]，Google の Auto pilot[11]があり，実際のクラウド運用におけるコンテナ資源使用量制限の自動調整（auto-scaling）に用いられている。いずれも時系列データ解析に機械学習を用いるので，クラウド環境でアプリが長時間実行された実績があり大量の資源使用量データが蓄積されていることが必要になる。また，アプリごとに資源使用量を機械学習するため，同種のアプリが大規模に実行される専用のクラウドに適用する場合には有効だが，多様な目的で使用される汎用的なクラウドに対して適用するのは難しい。さらに，エラーや性能低下を回避したい利用者がコンテナ資源制限を大きめに初期設定して運用されている状態の資源使用量データを分析して，実際の使用量を推論してコンテナサイズを自動調整することを想定している。

(3)はソースコードの構造を反映したモデル化をおこなうために，アプリ開発者でなければモデル作成が困難であり，オープンソースを組み合わせて構成される場合が多いクラウドアプリモデル化には不向きと考えられる。

(4)の性能シミュレータを用いる方法は，アプリごとにシミュレータを作成する作業が必要となり，実行時間が問題になる特定のアプリについて性能解析を行う手段として用いられる。例として並列動作するアプリをシミュレーションする ParSim [12] があるが，ログや性能プロファイルを解析することによりタスクグラフを作成する作業が必要となる。

本研究のリソース量推論モデルの作成方法は，(2)と異なりデプロイ前に利用者が提示する性能要件をみたすコンテナサイズを導出する目的で使用することができる。さらにテスト実行およびアプリモデル作成の自動化ツールによって，(3)のようにソースコードの構造を理解している必要もなく，(4)のようなアプリごとのモデル作成作業を省力化することができる。

6. まとめ

ハイブリッドクラウド環境において、各アプリの要求性能を満たすリソース量を求めるリソース量推論モデルを作成するために、様々な条件下でのアプリの稼働情報を取得するテスト実行自動化ツールを提案した。

テスト実行自動化ツールの効果検証を行うために実機環境を利用して，テスト実行自動化ツールが，どの程度の実行時間で完了することが可能か確認し，次に，テスト実行自動化ツールを適用した時のテスト実行手順の削減効果を検証した。

検証結果からテスト実行自動化ツールの実行時間については,約 158 秒で完了する見通しを得た。また，業務システム開発者と業務システム運用者のテスト実行手順が机上検討により 15 ステップとなるが,テスト実行自動化ツールを適用した場合には，7 ステップを自動化でサポートできるため，テスト実行手順を約 47%削減できる見通しを得た。

今後、様々なアプリの種類やアプリのデプロイ構成などに合わせてテスト実行自動化ツールのエンハンスおよび当該ツールの実行結果によるリソース量推論モデルの自動化について研究開発を推進する。

参 考 文 献

[1] IDC, “Worldwide Big Data and Analytics Software Forecast, 2019–2023,” IDC Doc #US44803719, 2019.

[2] IDC,“Worldwide Internet of Things Forecast, 2019-2023,” IDC Doc #US45373120, 2019.

[3] 水野和彦,”コンテナの処理特性に応じた割当リソース量を推論するリソース量推論モデルの検討,” DEIM2022, 2022.

[4] Prometheus,“From metrics to insight Power your metrics and alerting with a leading open-source monitoring solution,” https://prometheus.io/

[5] github,”kubernetes-client/python,” https://github.com/kubernetes-client/python

[6] github, “google/cadviosr, ” https://github.com/google/cadvisor

[7] github, “prometheus/node_exporter, ” https://github.com/prometheus/node_exporter

[8] Scikit learn,” sklearn.datasets.fetch_20newsgroups, “ https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_20newsgroups.html

[9] D. Buchaca, et. al., "Proactive Container Auto-scaling for Cloud Native Machine Learning Services", 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), Beijing, 2020, pp. 475-479

[10] D. Buchaca, et. al., “Automatic Generation of Workload Profiles Using Unsupervised Learning Pipelines”, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, VOL. 15, NO. 1, MARCH 2018

[11] K. Rzadca, et. al., "Autopilot: workload autoscaling at Google", Euro Sys ‘20

[12] A. Rosa, et. al., “ParSim: A tool for workload modeling and reproduction of parallel applications,” in Proc. IEEE 22nd Int. Symp. Model. Anal. Simulat. Comput. Telecommun. Syst., Paris, France, Sep. 2014, pp. 494–497.