WordPressのコンテナ化に伴うコンテンツ送信と設定ファイル変更の一括処理による移行の自動化

圖齋 雄治[1]　大野 有樹[2]　串田 高幸[1]

概要：仮想マシンで動作している WordPress をコンテナに移行することはシステムの可用性や拡張性の面で重要である．この時，WordPress の移行に伴ってコンテンツを移行する場合がある．しかし，管理者が知識のない初心者だと WordPress のコンテンツデータをどのように移行し，移行後に何をすれば変わらずにコンテンツデータを使用できるのか分からない．そのため，開発者にとって負担となる．この課題を解決するため，移行するデータを移行先に対して送信し，データを使用できるようなファイル設定にする作業を自動化し一括で行えるようなソフトウェアを提案した．提案ソフトウェアと手作業での作業時間を比較すると，提案ソフトウェアの実行時間は約 105 秒，手作業は約 165 秒で提案ソフトウェアは約 60 秒短縮することができた．

1. はじめに

背景

ブログサイトや EC サイトを運用する際に，サイトへのアクセス数が増加しても Web サイトのコンテンツ閲覧を可能にすることは重要である．アクセス数の増加に対応できないと，Web サイトを動かすサーバーがアクセスの処理に追いつかなくなり，サイトのレスポンスが遅くなる．また，サイトを運用するサーバーのリソースが足りなくなり，サーバーがダウンしてしまう [1].

アクセス数の増加に対する解決策の一つとして，Web サイトをコンテナで仮想化し，Kubernetes 上で管理するという手法がある．コンテナは，仮想マシンと比較して軽量である [2,3]．そのためコンポーネントの展開が素早く更新やスケーリングが容易である [4]．また，Kubernetes は自動スケーリングメカニズムを提供しているため，Kubernetes を使用する事でスケーリングを自動で行える [5]．すなわち，サイトのアクセス数が増加し，CPU やメモリが足りなくなっても自動でスケールアップされる．ユーザーは，アクセス増加の際も管理者を介すること無く，Web サイトを閲覧し続けられる．

この事から，個人や企業，研究機関が仮想マシンで運用しているレガシーアプリケーションをコンテナに移行する動きが高まっている [6][*1]．アプリケーションの移行に伴って，アプリケーションの機能だけでなく，アプリケーションが扱っていたデータも移行させる必要がある．オープンソースのブログソフトウェアである WordPress では，Web サイトで表示する写真や動画を含むコンテンツデータや，インストールしたプラグインやテーマも移行させる必要がある．

WordPress で使用するコンテンツデータを保存する方法として NFS(Network File System) がある．NFS は，コンピュータがネットワーク上でファイルを共有するための分散ファイルシステムの一つである [7]．共有ネットワークを介して NFS サーバーを起動しているディレクトリへデータを保存・取得することができる．これを使用することで，Kubernetes クラスター上の全てのノードが同じデータを共有して使用できる．

課題

仮想マシンで WordPress を動作させる場合，図 1 のようにコンテンツデータは仮想マシン内の同一ボリュームに保存されている．一方で，WordPress をコンテナ化して Kubernetes 上で運用する場合は図 2 のようにそれぞれのデータを別々のボリュームに分けて保存する．これによって，NFS や Kubernetes に詳しくない開発者はどの位置にどのデータをどのように配置すればいいのかわからず，アプリケーションの移行に時間がかかってしまう．

それに加え，データ移行時に設定ファイルやプラグイン

[1] 東京工科大学コンピュータサイエンス学部
〒 192–0982 東京都八王子市片倉町 1404-1
[2] 東京工科大学大学院 バイオ・情報メディア研究科コンピュータサイエンス専攻〒 192–0982 東京都八王子市片倉町 1404-1

[*1] https://onl.tw/1Acit12

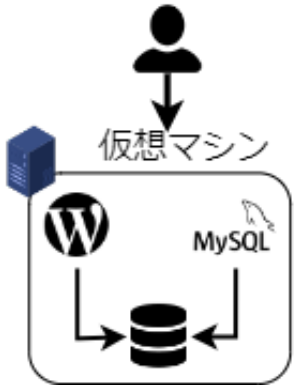

図 1　仮想マシンで運用する場合のデータ構成

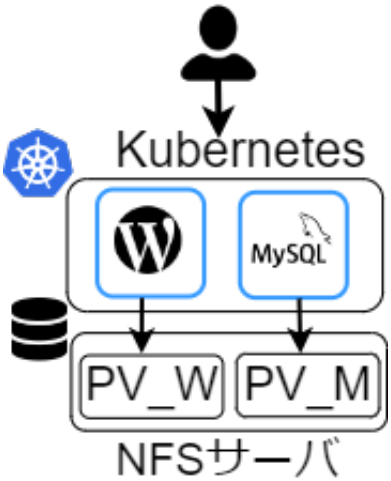

図 2　Kubernetes で運用する場合のデータ構成

を編集する必要があるが，これに関しても WordPress に詳しくない開発者にとっては，どの部分を編集すればいいのかわからず，移行に時間がかかってしまう．

また，WordPress の場合，コンテンツデータの移行が完了してもコンテンツに埋め込んだリンクや設定でトップページとしている URL を変更する必要がある．しかし，変更しなければならないリンクの数はその Web サイトに依存している．また，変更しなければならないリンクは SQL に保存されている場合があり，SQL のテーブル一つ一つを確認する必要が発生し，管理者にとって煩雑である．

各章の概要

本テクニカルレポートは以下のように構成される．第 1 章では，本稿の背景と課題について述べる．第 2 章では，本稿の関連研究について述べる．第 3 章では，本稿での課題を解決するための提案手法について述べる．第 4 章では，提案した手法の実装について述べる．第 5 章では，基礎実験の内容と，その評価について述べる．第 6 章では，提案手法の議論を述べる．第 7 章は，本稿のまとめである．

2.　関連研究

クラウド移行に関する 23 件の研究を分類し比較する事で系統的文献レビュー (SLR) を行った研究がある [8]．この論文では，クラウドへの移行に関連する主要なプロセスとタスクを特定し，選択された研究がどの領域に焦点を当てているのか分類した．結果としてこの論文では，クラウド移行を解決する提案が活発に研究されているが，そのほとんどが検証段階にあり，まだ大規模に使用されていないことが分かった．また，この論文では移行に伴うプロセスの自動化ツールが不足していると指摘している．よって本稿は，この論文の指摘を解決できる提案である．

クラウド環境でレガシーアプリケーションを移行するための新しいモデル駆動型アプローチと，すべての移行プロセスをサポートする統合フレームワークを開発した研究がある [9]．この論文では，移行対象となるアプリケーションが移行可能か分析して判断する．移行可能と判断されたなら，リバースエンジニアリングによってレガシーアプリケーションの分析モデルを生成する．その後アプリケーションを移行していく．この論文ではアプリケーションのデータについて「変換される」と記述されているが具体的な手法が書いていない．

また，GUI 認識・再構築技術によりレガシーアプリケーションを Web アプリケーションに効率的に移行する全く新しいアプリケーションマイグレーションソリューション（AMS）を提案した研究がある [10]．しかしこの論文では提案手法が詳細に書いておらず，データの取り扱いについても一切言及が無い．

3.　提案

提案方式

本稿では，WordPress のコンテンツデータ移行を簡略するために，コンテンツデータ移行の自動化を提案する．本提案ではこのコンテンツデータを二つに分けて定義する．WordPress が使用しているコンテンツデータのうち，写真やプラグイン，設定ファイルを WordPress のデータとし，SQL に保存されているデータを SQL のデータとする．この二つのデータをそれぞれ別々に送信・編集することで，コンテンツデータを混ぜることなく移行できる．図 3 が提案の概要である．また，図 4 が提案のシーケンス図である．流れとしては以下の通りである．

(1)　移行先の NFS サーバにデータ格納用のディレクトリを作成する．

(2)　送信するデータを選択する．

(3)　(1) で作成したディレクトリにファイルを送信する．

(4)　送信されたファイルの内容や所有者を編集する．

(5) 移行が完了したか確認を行う．

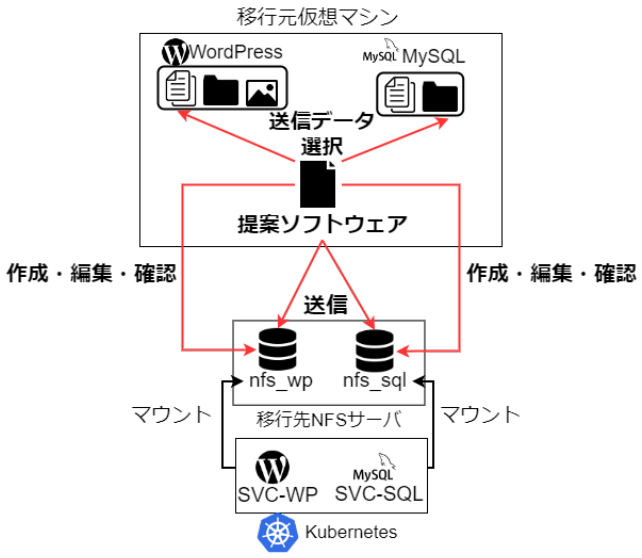

図 3 提案の概要

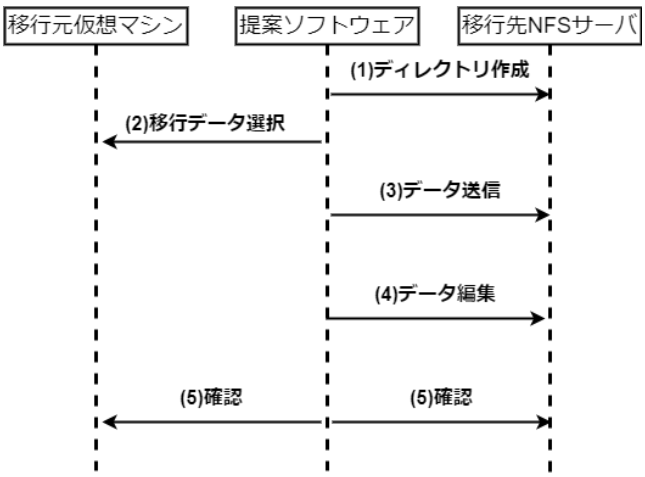

図 4 提案のシーケンス図

各要素を詳細に説明する．(1) では移行用の NFS サーバにコンテンツデータの送信先を作成する．提案ソフトウェアが移行先のサーバに対して命令を飛ばし，移行元の仮想マシン内から移行先のディレクトリ nfs_wp と nfs_sql を作成する．

(2) では (1) で作成したディレクトリに向けて送信するデータを選択する．今回は WordPress のコンテンツデータのため，WordPress のデータが保存されている/var/www/html と，MySQL のデータが保存されている/var/lib/mysql を選択する．

(3) では (2) で選択したデータを (1) で作成したディレクトリに向けて送信している．WordPress のデータは nfs_wp へ，MySQL のデータは nfs_sql へそれぞれ送信される．

(3) ではコンテンツデータの編集が行われる．送信されたファイルの所有者とグループを WordPress のデータは www-data に変換する．これは，Apache や Nginx のような Web サーバが通常の操作に使用するデフォルトユーザが www-data だからである[*2]．また，MySQL のデータは「lxd:999」に変換する．所有者の変更後，設定ファイルを編集する．編集する設定ファイルは「wp-config.php」である．このファイルの MySQL に関する記述を変更する．WordPress のデータと違い MySQL のデータに必要なことはリンク変換である．WordPress で MySQL を使う場合，各テーブルに以前使用していたサイトのリンクが格納されている．このテーブル一つ一つに対してリンクで検索をして，ユーザが指定したリンクに置換していく．

(4) ではデータの移行が正常に完了したかの確認を行う．移行の確認は主に以下の 2 つの観点から行う．

- 移行元の仮想マシンから送信したファイルに欠落が無いか．
- 移行前と移行後で同様のコンテンツを使用できるか．

前者はハッシュ値の比較で行う．移行前のファイルと移行後のファイルをハッシュ値で比較する事で，データ送信でファイルの内容に変化がないかを確認する．後者に関しては，全てのコンテンツを網羅的に取得して比較するのは時間がかかるうえ，ハッシュ値の比較でファイルの確認は終えているので，確認するページを絞って行う．確認するのはトップページと管理者ページ，そして移行するサイトの人気上位 10%のページである．

ユースケース・シナリオ

本稿では，仮想マシンにインストールされた WordPress で動作しているブログサイトを，コンテナ化して Kubernetes 上に移行する状況をユースケースとする．移行前の状態が図 5 である．WordPress や MySQL といったソフトウェアと，ソフトウェアが使うデータが同じ仮想マシンに配置されている．図 5 の構成では拡張性が低く，リソース不足が発生した時にすぐに対応が出来ない．また，アプリケーションとデータが同一の仮想マシンに保存されているため，どちらか一方のメンテナンスを行っても両方を再デプロイする必要がある．これを解決するためこのブログサイトをコンテナ化する．

背景でも述べた通り，アプリケーションをコンテナ化して Kubernetes で管理することで展開を高速にし，スケーリングを自動的に行えるようになる．移行後の状態が図 6 である．図 5 から図 6 の状態に移行する際に本提案を使用することで，WordPress や Kubernetes の知識が浅くても，移行が問題なく行われる．

[*2] https://onl.sc/SB5K2GB

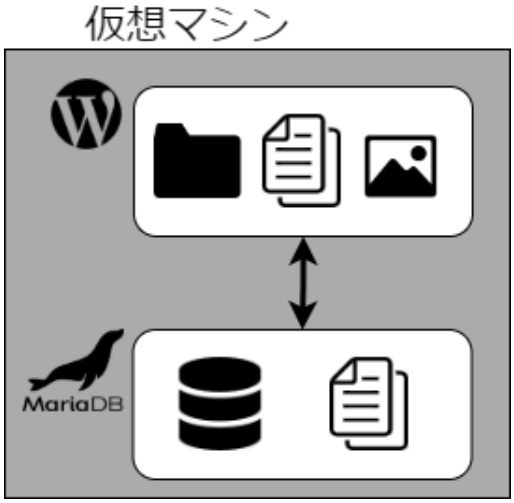

図 5　移行前の状態

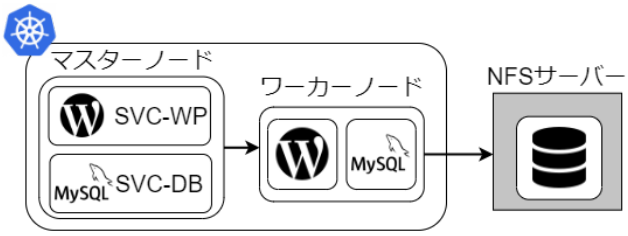

図 6　移行後の状態

4.　実装

実装では，データを移行元の仮想マシンから移行先の NFS サーバに送信するソフトウェアを作成する．図 7 が実装するソフトウェアの概要図である．実装するソフトウェアは，移行元の仮想マシンと移行先の NFS サーバの二つに渡って動作する．以下で各動作を説明する．後述されている ssh のコマンドをわかりやすくするために，NFS サーバーのユーザー名は user2，アドレスは NFS-Server とする．

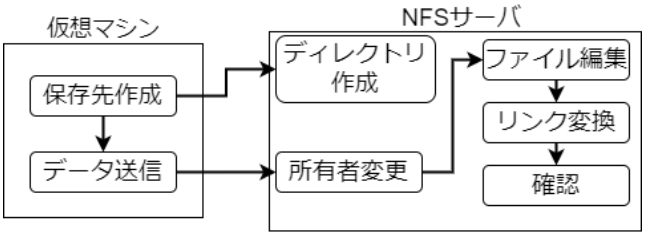

図 7　ソフトウェアの概要

4.1　保存先作成・ディレクトリ作成

新しい保存先の作成は移行元の仮想マシン側から移行先の NFS サーバに向けて命令する．以下がそのコマンドである．今回は移行先 NFS サーバのホームディレクトリ直下に新しい保存先を作成した．

Listing 1　保存先作成

```
1 ssh user2@NFS-Server 'mkdir ~/nfs_wp && mkdir
      ~/nfs_sql'
```

4.2　データ送信

移行元の仮想マシンに保存されているコンテンツデータを移行先 NFS サーバに作成したディレクトリに対して送信する．以下がそのコマンドである．送信には rsync コマンドを使い作成したディレクトリに送信する．

Listing 2　データ送信

```
1 sudo rsync -av /var/www/html/ cdsl@c0a20092-
      sqltest:~/nfs_wp
2 sudo rsync -av /var/lib/mysql/ cdsl@c0a20092-
      sqltest:~/nfs_sql
```

4.3　所有者変更

送信されたコンテンツデータの所有者変更は一括で行う．これは ssh コマンドを使用する事で NFS サーバのパスワードを要求される回数を減らす目的がある．以下がそのコマンドである．

Listing 3　所有者変更

```
1 ssh cdsl@c0a20092-sqltest 'sudo -S chown -R www
      -data:www-data ~/nfs_wp/*'
2 ssh cdsl@c0a20092-sqltest 'sudo -S chown -R lxd
      :999 ~/nfs_sql/*'
```

4.4　ファイル編集

展開された設定ファイルの中を編集する．提案でも述べた通り，編集するファイルは wp-config.php である．このファイルの記述を取得し，修正が必要な部分だけ置換する．

4.5　リンク変換

リンク変換は MySQL にログインした状態で行う．以下のコマンドを各テーブルに総当たりで入力する．table_name にテーブル名，column_name に列名，'search' に置換を行いたい文字列，'replace' に置換する文字列を入力する．

Listing 4　保存先作成

```
1 UPDATE table_name SET column_name = REPLACE(
      column_name, 'search', 'replace') WHERE
      column_name LIKE '%search%';
```

4.6 確認

移行作業が終了した後に，移行が正常に終了したかを確認する．提案で述べたように，移行完了の基準はファイル内容，コンテンツの共通度である．前者に関しては，移行元の仮想マシンのディレクトリにあるファイルのハッシュ値を取得しておき，移行完了後のハッシュ値と比較する．コンテンツの共通度は，実際にコンテンツを取得する事で比較する．移行前の仮想マシンと移行先の Kubernetes クラスターにそれぞれ curl コマンドで通信を行い，コンテンツデータを取得する．取得したコンテンツデータを比較することで共通度の確認を行う．

5. 実験

実験では実装したソフトウェアを稼働させ，移行したソフトウェアを稼働させ移行が完了したか確認する．また，それぞれの実装フェーズにかかった時間を計測し，手作業でかかった時間と比較する．それぞれ 5 回ずつ行い，平均を算出した．

実験環境

実験環境を図 8 に示す．移行元の仮想マシンとして実際にブログサイトが動作している仮想マシンを用意した．また，移行先として NFS サーバを用意した．NFS サーバに新しくコンテンツデータを保存するディレクトリを作成する．その後，仮想マシンに保存されている WordPress のコンテンツデータを NFS サーバに送信する．送信完了後にコンテンツデータの所有者を変更する．図 8 の破線が各工程の実行を表している．この実行時間を計測し総和を計算する．以下にそれぞれの仮想マシンの構成要素を示す．

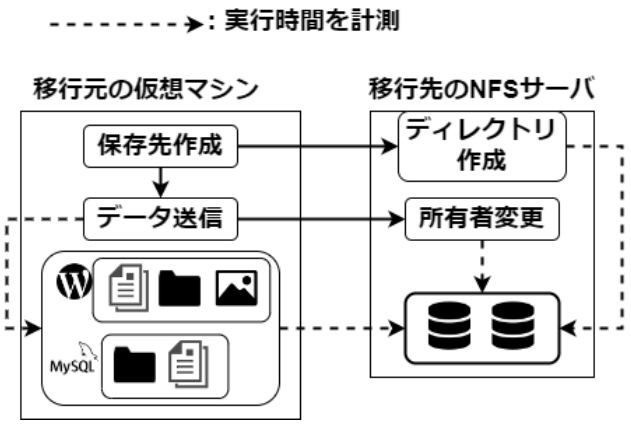

図 8 実験環境

- 移行元の仮想マシン 構成要素
  OS：Ubuntu-22.04
  vCPU：2 コア
  RAM：4GB
  HDD：50GB
- 移行先の NFS サーバ 構成要素
  OS：Ubuntu-22.04
  vCPU：2 コア
  RAM：2GB
  HDD：50GB

実験結果と分析

図 9 は提案ソフトウェアの実行時間を手作業の実行時間と比較したものである．提案ソフトウェアは平均約 105 秒，手作業は平均約 165 秒かかった．提案ソフトウェアは手作業と比較して約 60 秒短縮できた．手作業ではコマンドを入力する時間が計測されている分時間がかかっている．

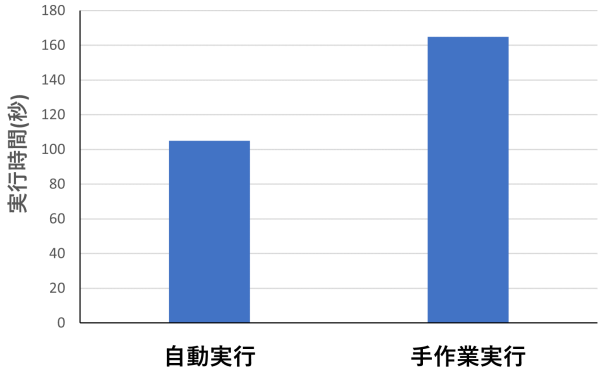

図 9 手作業との比較

6. 議論

本稿では，課題としてデータ移行の作業が煩雑とし作業の自動化を提案した．しかし，ソフトウェアの移行そのものに対しては今回の研究では触れていない．例えば，コンテナ化したソフトウェアを Kubernetes 上で管理する場合，この Kubernetes の環境を作成する必要がある．環境の作成には，クラスターの作成や NFS との紐付けを行う必要がある．この作業も Kubernetes や NFS サーバーの知識を要求されるため，開発初心者にとって負担になる．この点に関しては，テンプレートとなるクラスター構成を用意しておき，ユーザーの要求次第でクラスター構成を変化させることで開発者の負担を減少させる．例えば，ソフトウェアをダウンさせない構成にするためにワーカノードを複数台用意して 1 台が機能停止をしても維持できるようにする．他にも，移行前のアプリケーションが稼働している仮想マシンのスペックを取得し，そこからクラスターの構成やスペックを決定するようにすると，ユーザーが行う作業はより減少する．

また，今回の研究の前提として，オンプレミスの仮想マシンから，同じくオンプレミスの仮想マシンをノードとした Kubernetes クラスターに移行する場合をユースケースとした．一方でソフトウェアの環境が常にオンプレミスだ

とは限らない．例えば Google Compute Engine(GCE) に仮想マシンを作成してソフトウェアを稼働させていたり，移行先が Amazon Web Services(AWS クラウド) という状況もある[*3]．さらに，移行後の構成としてオンプレミス環境とクラウド環境の二つを併用した構成を採用する場合がある[*4]．これらの構成は今回の提案のユースケースに含めていない．提案ソフトウェアで記述した仮想マシンや NFS サーバの IP アドレスを，クラウドの IP アドレスに変更することで，本提案を使用できる．

実装に関しても幾つかの懸念点がある．まず第一にこのソフトウェアを移用する前に移行元の仮想マシンと移行先の NFS サーバの間で SSH の暗号鍵を共有する必要がある．これは SSH の操作を簡略化するために行ったが，この工程も自動化するとより管理者の作業が減るため楽になる．第二に SSH 通信と sudo コマンドを組み込んでいるため，移行元の仮想マシンと移行先 NFS サーバのパスワードを要求される．そのため実際には自動化ではない．提案ソフトウェアにパスワードを記載することで入力を省略する手法もあるが，これは関係の無い第三者が気軽に実行できてしまう観点から，この部分を自動化するのはリスクがある．第三に実装したソフトウェアはチェックポインを採用しておらず，例えばデータ送信中にエラーが出てソフトウェアの動作が停止しても，ソフトウェアを再実行するとデータの送信をまた行ってしまう．これは冗長であるため，すでに送信されたファイルがあるならばそのファイルの送信をスキップして柔軟に対応できるようにすると有効である．

7. おわりに

WordPress の移行には複数の工程がある．その中でもデータの移行はどのデータを何処に置くのか，権限はどうするかのような開発初心者にとって負担になることが多い．そのためデータ移行に伴う作業を自動化し，一括で行えるようなソフトウェアを提案した．この提案手法を基に作成した自動化ソフトウェアと手作業での実行時間を比較すると，約 60 秒短縮することができた．本稿の提案で記載したファイルの編集とリンク変換を実験していない．次のレポートではこの二つを実装させ，評価を行いたい．

参考文献

[1] Pertet, S. M. and Narasimhan, P.: Causes of Failure in Web Applications (CMU-PDL-05-109) (2005).

[2] Pahl, C.: Containerization and the PaaS Cloud, *IEEE Cloud Computing*, Vol. 2, No. 3, pp. 24–31 (online), DOI: 10.1109/MCC.2015.51 (2015).

[3] Bernstein, D.: Containers and Cloud: From LXC to Docker to Kubernetes, *IEEE Cloud Computing*, Vol. 1, No. 3, pp. 81–84 (online), DOI: 10.1109/MCC.2014.51 (2014).

[4] Watada, J., Roy, A., Kadikar, R., Pham, H. and Xu, B.: Emerging Trends, Techniques and Open Issues of Containerization: A Review, *IEEE Access*, Vol. 7, pp. 152443–152472 (online), DOI: 10.1109/ACCESS.2019.2945930 (2019).

[5] Balla, D., Simon, C. and Maliosz, M.: Adaptive scaling of Kubernetes pods, *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–5 (online), DOI: 10.1109/NOMS47738.2020.9110428 (2020).

[6] Imran, M., Kuznetsov, V., Dziedziniewicz-Wojcik, K. M., Pfeiffer, A., Paparrigopoulos, P., Trigazis, S., Tedeschi, T. and Ciangottini, D.: Migration of CMSWEB cluster at CERN to Kubernetes: a comprehensive study, *Cluster Computing*, Vol. 24, No. 4, pp. 3085–3099 (2021).

[7] Chen, H., Tang, R., Zhao, Y., Xiong, J., Ma, J. and Sun, N.: Research on Key Technologies of Load Balancing for NFS Server with Multiple Network Paths, *2006 Fifth International Conference on Grid and Cooperative Computing Workshops*, pp. 407–411 (online), DOI: 10.1109/GCCW.2006.79 (2006).

[8] Jamshidi, P., Ahmad, A. and Pahl, C.: Cloud Migration Research: A Systematic Review, *IEEE Transactions on Cloud Computing*, Vol. 1, No. 2, pp. 142–157 (online), DOI: 10.1109/TCC.2013.10 (2013).

[9] Menychtas, A., Santzaridou, C., Kousiouris, G., Varvarigou, T., Orue-Echevarria, L., Alonso, J., Gorronogoitia, J., Bruneliere, H., Strauss, O., Senkova, T., Pellens, B. and Stuer, P.: ARTIST Methodology and Framework: A Novel Approach for the Migration of Legacy Software on the Cloud, *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 424–431 (online), DOI: 10.1109/SYNASC.2013.62 (2013).

[10] Meng, X., Shi, J., Liu, X., Liu, H. and Wang, L.: Legacy Application Migration to Cloud, *2011 IEEE 4th International Conference on Cloud Computing*, pp. 750–751 (online), DOI: 10.1109/CLOUD.2011.56 (2011).

[*3] https://onl.tw/1Acit12
[*4] https://onl.tw/LmqY6cX