コンテンツコピーと設定ファイル変更の一括処理による仮想マシンからコンテナへのWordPress移行時間削減

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，管理者にとって移行の時間が増加してしまう．この課題を解決し移行作業の実行時間を短縮するため，移行するデータを移行先に対して並列でコピーし，設定ファイルとコンテンツデータの編集を一括で行える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，そして提案ソフトウェアと作業を並列で行わない場合をそれぞれ 5 回実験し比較した．提案ソフトウェアの実行時間は平均で約 99 秒，手作業は平均約 1208 秒，非並列が平均約 116 秒であった．提案ソフトウェアは手作業から約 12 分の 1 に短縮し，非並列とは約 17 秒削減することができた．

1. はじめに

背景

ブログサイトや EC サイトを運営する際，ウェブサイトへのトラフィックが増加しても，ウェブ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 [1]．トラフィックの増加に対処できない場合，ウェブサイトをホストするサーバーはアクセスの処理に追いつかず，サイトの応答が遅くなる．

トラフィックの増加に対する解決策の一つとして，ウェブサイトをコンテナで仮想化し，Kubernetes によって管理するアプローチが存在する．コンテナは，仮想マシンと比較して軽量であるため，コンポーネントの迅速な展開や更新，スケーリングが容易に行える [2–4]．さらに，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 を採用することでスケーリングを自動的に行える [5]．そのため，ウェブサイトのトラフィックが増加し，CPU やメモリの不足が発生しても，自動的にスケールアップされる．ユーザーは，トラフィックが増加しても，管理者を介さずにウェブサイトの閲覧を継続できる．

これらの事から，個人や企業，研究機関が仮想マシンで稼働させ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増加している [6][*1]．アプリケーションを移行する際，その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使用していたデータも同様に移行する必要がある．ブログや Web サイトを作成し運用できる WordPress というオープンソースソフトウェアでは，Web サイトで表示する写真や動画といったコンテンツデータや，拡張機能であるプラグイン，サイトデザインを変更するテーマも一緒に移行する必要がある．

WordPress で用い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 サーバーのディレクトリへデータの書き込み・読み取りを可能にする．NFS を用いることで，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

WordPress を仮想マシンで動作させる場合，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 のように保存される．これは WordPress のデータが PV_W に，MySQL のデータは PV_M に分別されて二つのボリュームに保存される状況を表している．これによって，NFS や Kubernetes に詳しく

[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

ない管理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．

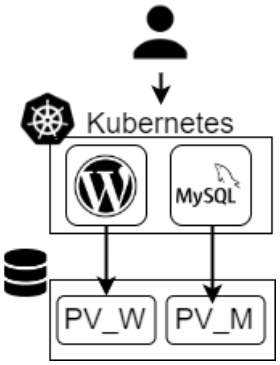

図 2 Kubernetes で運用する場合のデータ構成

それに加え，管理者はデータ移行時に，移行先の環境に応じて設定ファイルを編集する必要がある．これに関しても WordPress に明るくない管理者にとって，どの部分をどのように編集すればいいのかわからず，調査時間が増えて結果として移行にかかる時間が増加する．

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存在する埋め込み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 サイトの状況に依存している．また，変更が必要なリンクはデータベースである SQL に保存されており，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，実験の内容と，その評価について述べる．第 6 章では，本稿の議論を述べる．第 7 章は，本稿のまとめである．

2. 関連研究

イタリアにおける情報システムの移行について調査した論文がある [8]．イタリアの IT 企業 59 社を対象にシステム移行経験の有無や用いた手法，移行の動機をアンケートでヒアリングした．結果として筆者らは移行活動における適切な自動化ツールが不足していると結論付けた．本稿はこの論文の指摘を解決できるものである．

ソフトウェア移行の際に発生するソースコード移行を自動化した研究がある [9]．この研究では特定ライブラリのモデルを定義するために使用できるドメイン固有言語（DSL）を提案した．ソースコード移行のプロセスをトレース抽出，トレースマッピング，トレース計算，トレースバック，コード変換の 5 ステップに分けステップごとにアルゴリズムを開発した．人工的なコード例とオープンソースプロジェクトの二つで実験し，コードの移行を正常に自動化できると評した．しかし本稿は WordPress を対象としており，コード変換だけでは移行が正常に完了しない．

あるソフトウェアにおけるライブラリ依存関係を自動で特定し半自動でライブラリを移行するアプローチを提案した研究がある [10]．java ソフトウェアのソースコードに記述されてるライブラリを抽出し，各ライブラリで定義されているシンボルを参照する事でソフトウェアとライブラリの依存関係を表現した．GitHub に公開されているリポジトリを対象にこのアプローチを検証していた．しかしこのアプローチはライブラリの変更が前提であり，ライブラリのバージョン更新がカウントされていない．また，対象が java ソフトウェアに限定されているため，データベースを扱う本稿では使用できない．

3. 提案

提案方式

本稿では，WordPress のコンテンツデータ移行に掛かる時間を削減するために，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，/var/www/html 直下にある写真やプラグイン，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扱うことで，コンテンツデータが混ざることなく移行できる．このことでコンテンツデータの形式を保つことができる．図 3 が提案の概要である．また，図 4 が提案ソフトウェアのシーケンス図である．提案ソフトウェアの流れとしては以下の通りである．

(1) 各仮想マシンにデータ格納用のディレクトリを作成する．

(2) (1) で作成したディレクトリを互いに同期させる．

(3) 移行するデータを準備する．

(4) 同期したディレクトリにデータをコピーする．

(5) コピーされたファイルの内容を編集する．

(6) SQL のデータを編集する．

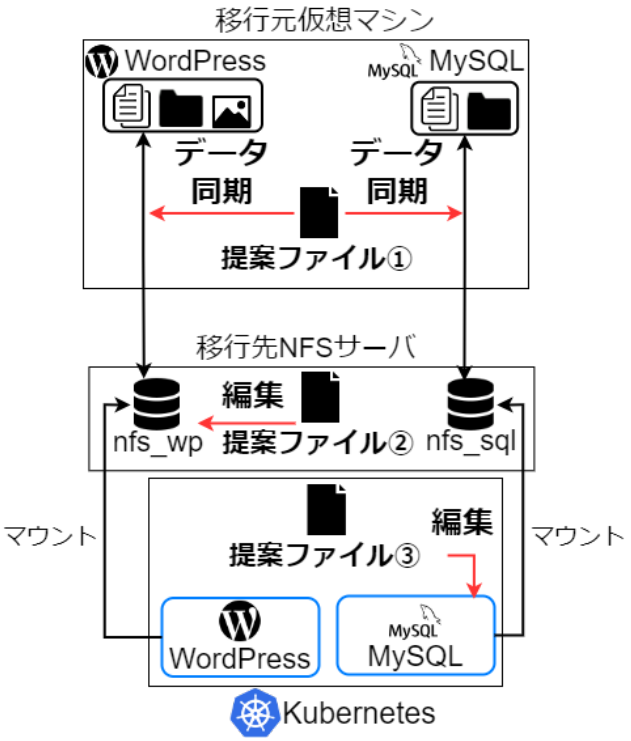

図 3 提案の概要

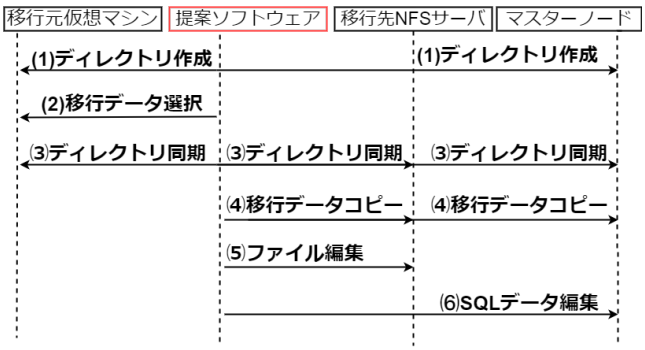

図 4 提案ソフトウェアのシーケンス図

各要素を詳細に説明する．(1) では移行元の仮想マシンと NFS サーバ，Kubernetes クラスタのマスターノードの 3 つの仮想マシンにディレクトリを作成する．まず移行用の NFS サーバにコンテンツデータの保存先を作成する．移行元仮想マシンに同期するためのディレクトリ wpfiles と sqlfiles を作成する．

(2) ではディレクトリ同士を同期させる．NFS サーバに存在する WordPress データ格納用のディレクトリ nfs_wp と，(1) で作成した wpfiles を同期する．また，Kubernetes クラスタのマスターノードに存在する MySQL のデータ格納用のディレクトリ dir_sql と (1) で作成した sqlfiles を同期する．

(3) では移行するコンテンツデータを準備する．WordPress のデータは 4 つに分けて Linux の tar コマンドでアーカイブする．WordPress のコンテンツデータは wp-admin, wp-content, wp-includes の 3 つのディレクトリと php ファイルや txt ファイルが存在する．これを図 5 のようにディレクトリを除いたファイル群と 3 つのディレクトリそれぞれを分割してアーカイブする．MySQL のデータはダンプしてエクスポートする．

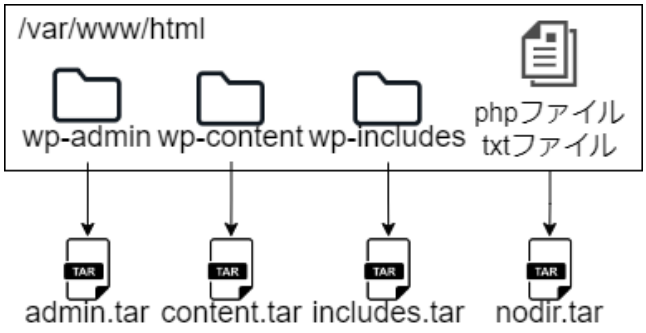

図 5 アーカイブ方法

(4) では同期したディレクトリに (3) で準備したコンテンツデータを展開し，コンテンツデータをコピーする．tar でまとめた WordPress のデータは wpfiles に展開し，エクスポートした MySQL のデータは sqlfiles に展開する．その後，コピーした MySQL のデータを Kubernetes の MySQL の Pod にインポートする．データの展開が終わったら，ディレクトリ同士の同期を取りやめる．

(5) では WordPress の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 に接続する設定を変更する．

(6) では MySQL のデータを編集する．WordPress でデータベースとして MySQL を使用する場合，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．このままでは移行後にサイト内を移動する際に，移行前のリンクにアクセスしてしまい，サイトのコンテンツが使用できなくなってしまう．そのためサイトのコンテンツデータが保存されているデータベースのテーブルに対して，一つずつ移行前のリンクを検索し，該当する場合は置換を行う．

(7) ではコンテンツデータの移行が正常に完了したかのチェックを行う．チェックの基準として以下の二つの手法でチェックする．

- 移行元の仮想マシンとコピーしたファイルとで差異が無いか．

● 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．移行前とコピー直後のファイルのハッシュ値をそれぞれ取得し，それを比較する事でファイルの内容や権限に変化がないかを確認する．後者は，移行対象全てのコンテンツを取得して比較する行為は時間がかかるうえ，一つ目の比較で使用するファイルの確認を終えているので，確認するコンテンツを絞って行う．確認するのは移行する Web サイトの人気上位 10%のコンテンツである．

ユースケース・シナリオ

本稿では，WordPress が動作している仮想マシン上のブログサイトを，Kubernetes 上にコンテナ化して移行する事例を考える．移行前のブログサイトが図 6 である．WordPress や MySQL といったアプリケーションと，アプリケーションが使うコンテンツデータが同一の仮想マシンに配置されている．図 6 の構成では柔軟性が不足しており，リソース不足が発生した時に迅速な対応が出来ない．また，アプリケーションとコンテンツデータが同じ仮想マシンに格納されているため，片方のメンテナンスを行ったとしても両方を再デプロイしなければならない．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．

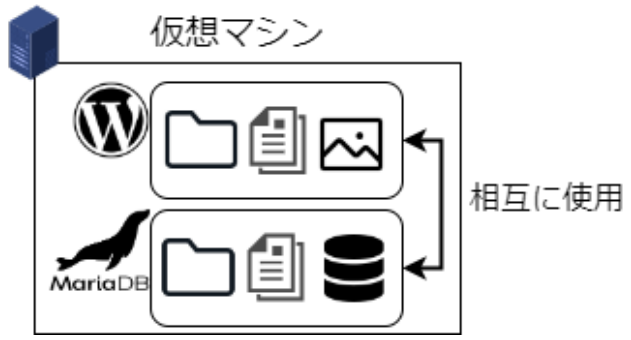

図 6 移行前の状態

背景で前述したように，アプリケーションをコンテナ化し Kubernetes で管理することは，デプロイをより早く行えるようにし，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 7 である．図 6 から図 7 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，移行を素早く行える．

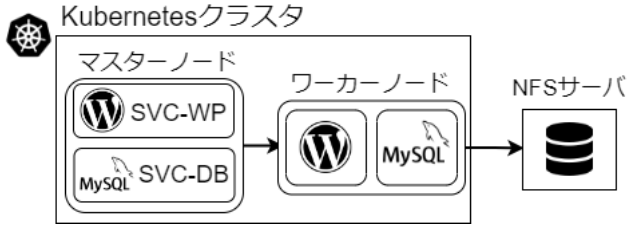

図 7 移行後の状態

4. 実装

実装では，コンテンツデータを移行元の仮想マシンから移行先の NFS サーバにコピーし，コンテンツデータの編集を行うソフトウェアを作成する．図 8 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，移行先の Kubernetes クラスタのマスターノードと三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．また，マスターノードのユーザー名は user3，アドレスは Master とする．

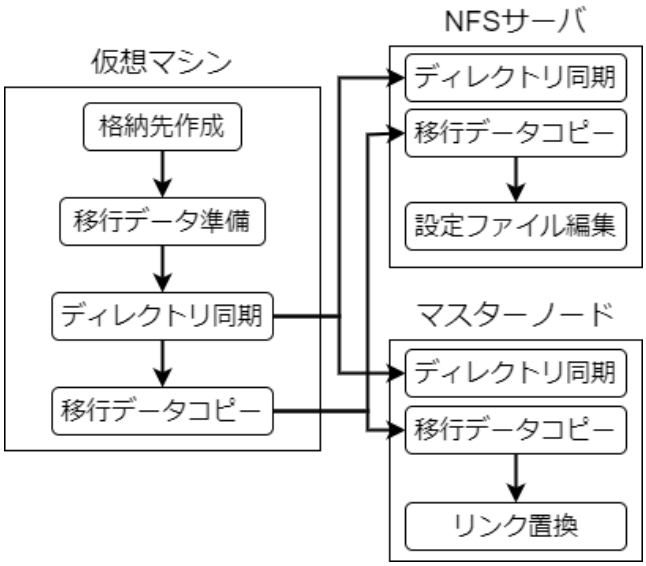

図 8 ソフトウェアの概要

4.1 格納先作成

移行元の仮想マシンに，コピーするコンテンツデータを格納しておくディレクトリを作成する．mkdir コマンドを使用して移行元の仮想マシンに wpfiles と sqlfiles を作成する．

4.2 移行データ準備

移行元仮想マシンに存在するコンテンツデータを WordPress データと MySQL データの二つに分けて纏める．WordPress のデータは Linux の tar コマンドを使用してアーカイブする．提案で前述した 4 つのアーカイブは，Linux の&コマンドを使用することにより並列で行う．MySQL のデータは mysqldump コマンドでダンプしておく．実行を自動で行うためにパスワードをコマンドに記述している．

4.3 ディレクトリ同期

作成したディレクトリを移行先のディレクトリと同期

する．wpfiles は NFS サーバと同期し，sqlfiles はマスターノードと同期する．

4.4 移行データコピー

同期したディレクトリに移行するデータを展開して，移行先にデータをコピーする．WordPress のデータは wpfiles に展開していく．&コマンドを使用することで，4 つの tar ファイルの展開を並列で行う．MySQL のデータは sqlfiles に移行した後，マスターノードのルートディレクトリに移動させる．データの展開が終了したらディレクトリ同士の同期を解除する．その後，マスターノードにコピーした MySQL のデータを，Kubernetes で動作している MySQL の Pod にエクスポートする．

4.5 コンテンツデータ編集

移行されたコンテンツデータを編集する．まずは WordPress の設定ファイルを編集する．提案でも述べた通り，編集するファイルは wp-config.php である．オンプレミスで WordPress を動作させる場合と Kubernetes で動作させる場合では wp-config.php の記述方法が変化する．そのため，Kubernetes で動作させる場合の wp-config.php のテンプレートを作成しておき，そこに移行前の wp-config.php に記載されているデータベースに関する記述を代入していく．次に MySQL のデータを編集する．移行したデータベースの全てのテーブルに移行前のリンクが無いか検索し，該当したならば移行先のリンクに置換する．

コンテンツデータの編集はそれぞれ別の Python ファイルを NFS サーバとマスターノードに用意し，そのファイルを移行元仮想マシンのシェルスクリプトファイルから呼び出して並列で実行する．

5. 評価実験

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他手法でかかった時間とで比較する．他手法とはコンテンツデータのアーカイブやコピーを並列で行わず順番に実行した場合である．手作業は入力するコマンドを知っており，どんな作業をすれば移行が出来るのかを把握している状態である．それぞれを 5 回ずつ行い，平均を算出した．なお，提案ソフトウェアの時間を計測するにあたり Linux の date コマンドを使用し，小数点以下を切り捨てた．

実験環境

実験環境を図 9 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，移行先の Kubernetes クラスタと NFS サーバを用意した．移行先の NFS サーバと Kubernetes クラスタのマスターノードにコンテンツデータを格納するための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 データをアーカイブし MySQL のデータをエクスポートする．その後，WordPress のデータは NFS サーバにコピーし，MySQL のデータはマスターノードにコピーして Pod にインポートする．データコピー後に各データを編集する．図 9 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．

- 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
- NFS サーバ 構成要素
  OS：Ubuntu-22.04
  vCPU：2 コア
  RAM：2GB
  HDD：50GB
- Kubernetes クラスタのマスターノード 構成要素
  OS：Ubuntu-22.04
  vCPU：3 コア
  RAM：3GB
  HDD：25GB

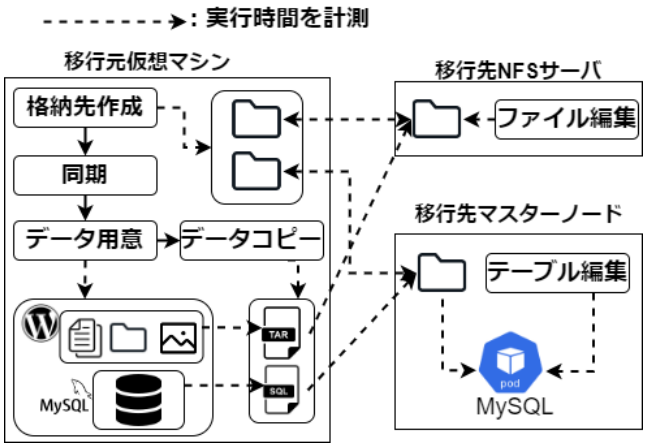

図 9 実験環境

また，コンテンツデータの移行が正常に完了したかの確認を行う．提案で述べたように，確認はコンテンツデータをハッシュ値で比較する方法と，コンテンツそのものが変化していないかの比較である．前者はコピー直前とコピー直後のファイルを Linux の md5sum コマンドで取得し，同一であるかを確認する．後者は移行前と移行後のコンテンツページを Linux の curl コマンドで取得し，取得した html コードの div クラス”content-area”内において，提案で置換したリンク以外で変更点が無いかを確認する．

実験結果と分析

各手法の時間との比較

まずは手作業からどのくらい作業時間を削減できたのか比較する．図 10 は提案手法と手作業の時間を比較したものである．提案ソフトウェアは平均約 99 秒で，手作業は平均約 1208 秒であった．手作業と比較して提案ソフトウェアは実行時間を約 12 分の 1 に短縮できた．

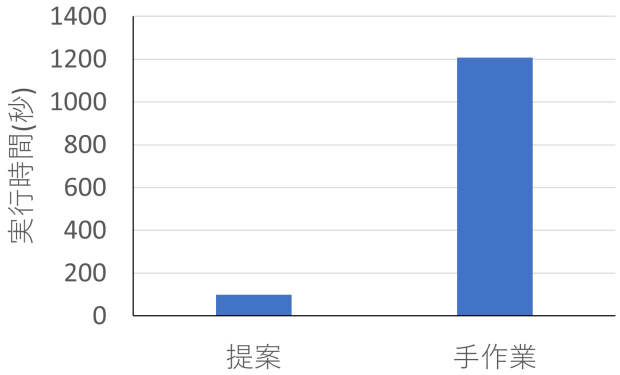

図 10 手作業との比較

次に他手法との比較を行う．図 11 は提案手法と他手法の時間を比較したものである．提案手法は平均約 99 秒で，他手法は平均約 116 秒であった．提案ソフトウェアは他手法と比較して実行時間を約 17 秒短縮している．格納先作成とディレクトリ同期は共に 1 秒未満であったことから図 11 では表示されていない．コンテンツデータのアーカイブは提案が平均約 15 秒で他手法が平均約 18 秒となり 3 秒短縮できた．データコピーは提案が平均約 82 秒で他手法が平均約 95 秒となり 13 秒短縮している．コンテンツデータの編集も他手法の平均約 3 秒から 1 秒短縮して提案が平均約 2 秒であった．コンテンツデータを直接扱う工程が軒並み実行時間を短縮できた．これは並列実行することにより，一つの大きなファイルの処理時間がボトルネックになりにくいためである．

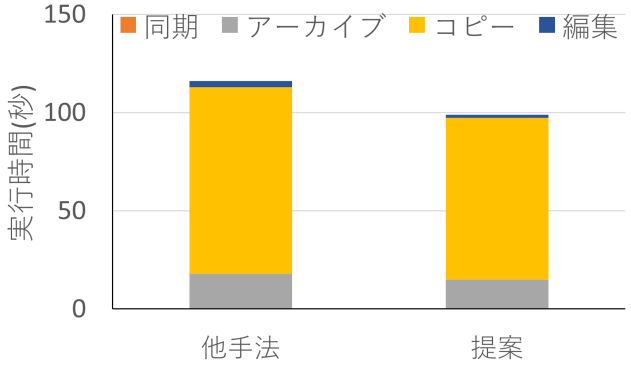

図 11 他手法との比較

移行完了の確認

まずは SQL ファイルをハッシュ値で比較した．結果として移行前の sql ファイルが”567e37d80ed26691b4062cc029e37797”で，移行後の sql ファイルが”567e37d80ed26691b4062cc029e37797”となり，一致した．次に，WordPress のファイルをハッシュ値で比較した．移行前の WordPress データを一つの tar ファイルにアーカイブし，コピーした直後と比較した．結果として移行前の tar ファイルが”89e698c594ab68f7e224d97968fcef5a”で，移行後の tar ファイルが”89e698c594ab68f7e224d97968fcef5a”となり，一致した．このことから，コンテンツデータの移行は正常に完了していることが分かった．

次に移行前と移行後でコンテンツが使用できるかの確認を行った．提案で述べた人気上位 10%のコンテンツは，WordPress のプラグインである WP Statistics を使用して取得した．その結果コンテンツ全 311 ページから上位 10%の 31 ページを取得した[*2]．取得した html コマンドをテキストファイルに書き写し，Linux の diff コマンドを使用して差異を確認した．結果として，プラグインの部分に変化点が二つあった．”data-ulike-nonce”という curl するたびに値が変化するデータと，”data-ulike-display-likers=””の” の中が 0 に変化していた．前者は curl のたびに値が変化するので差が生まれてしまう．後者は移行前から 0 であったが表示されていなかった．Web 上では変化が見られなかった．一方でプラグインを除いたコンテンツページの html そのものには変化は見られなかった．

6. 議論

本稿では，アプリケーションのコンテンツデータ移行が管理者にとって負担になるという課題に対して，移行作業の自動化を提案した．しかし移行先の Kubernetes 環境は作成されている前提としている．今までオンプレミスの仮想マシンでアプリケーションを動作させてきた管理者は Kubernetes や NFS サーバーを使用した経験がないため，環境の作成も負担になる．解決策として，ある程度一般的なクラスター構成をテンプレートとして準備しておき，管理者の移行先に対する要求を入力する事でクラスターの構成を変化させることで管理者の負担を減少できる．例えば，管理者がアプリケーションをダウンさせたくないと要求した際に，ワーカーノードを複数台作成したり，Pod の数を増加させることで，一部が機能停止してもアプリケーションを維持できるようにする．他にも移行前の仮想マシンのスペックを把握し，そのスペックからクラスターの構成や各 Pod のリソースを定めるようにすると，管理者の負担がより削減される．

[*2] https://ja.wordpress.org/plugins/wp-statistics/

本稿の提案において，コンテンツデータを並列で圧縮・コピーするために，ディレクトリごとにコンテンツデータを 4 つに分割した．この提案では分割したデータのそれぞれのサイズが異っており，写真が多く保存されている content.tar が他の 3 つに比べて大きくなっている．具体的には，content.tar が 2.2GB であったのに対して他の 3 つは平均約 18.7MB であった．これでは他 3 つの tar ファイルの圧縮・コピーが終了しても content.tar の実行を待たなければならない．そのため，コンテンツデータの分割方法をディレクトリごとではなく，コンテンツデータ全体のサイズを把握して均等に分割すると，より実行時間が削減できる．また，分割数は何が最適なのかを調査する必要がある．コンテンツデータを均等に分割して圧縮・コピーする際に，最も実行時間が速くなるには何個に分割するのが理想なのかは，コンテンツデータのサイズや環境によって変化する可能性がある．コンテンツデータのサイズが小さいと分割する必要が無くなる一方で，データサイズによっては分割数を増やしすぎると分割しない場合と変化がなくなる可能性がある．これらを考慮してコンテンツデータの分割数を決定すると，より移行にかかる時間を削減できる．

また，本稿のユースケースとして，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 クラスタへの移行を対象とした．一方で，アプリケーションが動作している環境はオンプレミスだけではない．例として，移行前は Google Compute Engine(GCE) の仮想マシンという場合もありえるし，移行先が Amazon Web Services(AWS クラウド) という状況もある[*3]．他にも，アプリケーションは変わらずにオンプレミスだが，データベースのみクラウドに移行したり，データベースを分散させるなどして環境を併用する場合がある[*4]．これらの構成に対しては，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．

実装に関しても議論がある．実装したソフトウェアはチェックポイントを採用しておらず，例えばデータコピー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コピーをまた最初から行ってしまう．これは冗長であるため，例えばすでにコピーされたファイルがあるならばそのファイルのコピーをスキップしてするようにソフトウェアを改善できる．

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，データはどこをどのように編集するかなど管理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及び並列で行わない場合を比較すると，提案ソフトウェアは手作業から約 1109 秒短縮し，非並列から約 17 秒短縮することができた．

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

参考文献

[1] Grassi, G., Teixeira, R., Barakat, C. and Crovella, M.: Leveraging Website Popularity Differences to Identify Performance Anomalies, *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1–10 (online), DOI: 10.1109/INFOCOM42981.2021.9488832 (2021).

[2] M, A., Dinkar, A., Mouli, S. C., B, S. and Deshpande, A. A.: Comparison of Containerization and Virtualization in Cloud Architectures, *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pp. 1–5 (online), DOI: 10.1109/CONECCT52877.2021.9622668 (2021).

[3] Manninen, H., Jääskeläinen, V. and Blech, J. O.: Performance Evaluation of Containerization Platforms for Control and Monitoring Devices, *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Vol. 1, pp. 1061–1064 (online), DOI: 10.1109/ETFA46521.2020.9211901 (2020).

[4] Kumudavalli, M. V. and Venkatesh, G.: Cloud Computing based Monolithic to Containerization using Elastic Container Service for Phylogenetic Analysis, *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, pp. 1540–1543 (online), DOI: 10.1109/ICICV50876.2021.9388395 (2021).

[5] He, Z.: Novel Container Cloud Elastic Scaling Strategy based on Kubernetes, *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pp. 1400–1404 (online), DOI: 10.1109/ITOEC49072.2020.9141552 (2020).

[6] Bergmayr, A., Brunelière, H., Izquierdo, J. L. C., Gorroñogoitia, J., Kousiouris, G., Kyriazis, D., Langer, P., Menychtas, A., Orue-Echevarria, L., Pezuela, C. and Wimmer, M.: Migrating Legacy Software to the Cloud with ARTIST, *2013 17th European Conference on Software Maintenance and Reengineering*, pp. 465–468 (online), DOI: 10.1109/CSMR.2013.73 (2013).

[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] Torchiano, M., Di Penta, M., Ricca, F., De Lucia, A. and Lanubile, F.: Migration of information systems in the Italian industry: A state of the practice survey, *Information and Software Technology*, Vol. 53, No. 1, pp. 71–86 (online), DOI: https://doi.org/10.1016/j.infsof.2010.08.002 (2011).

[9] Aleksyuk, A. and Itsykson, V.: AUTOMATED SEMANTICS-DRIVEN SOURCE CODE MIGRATION: A PILOT PROTOTYPE, (online), DOI: 10.31144/SI.2307-6410.2017.N10.P67-76 (2017).

[10] Teyton, C., Falleri, J.-R., Palyart, M. and Blanc, X.: A Study of Library Migration in Java Software, *ArXiv*, Vol. abs/1306.6262 (2013).