Vol.2015-DPS-163 No.16
Vol.2015-MBL-75 No.16
2015/5/28

インストール依存関係があるコンポジットアプリケーションサーバーの高速デプロイメント

勝野 恭治[1,a)]　高橋 ひとみ[1,b)]

概要：IaaS クラウドでは、アプリケーションがインストールされていないオペレーティングシステムのみの状態のサーバーが基本的にはデプロイされるため、ユーザーはサーバーが自動的にデプロイされた後に、アプリケーションを手動でインストールしなければならない。コンポジットアプリケーションデプロイメント方式は、アプリケーション毎にインストールスクリプトを事前に用意して、サーバーがクラウド上で起動・初期化された後に、ユーザーが選択したアプリケーションに該当するスクリプトをサーバー上で自動的に実行することによって、アプリケーションがインストールされたサーバーの自動デプロイメントを実現している。アプリケーションの中には、複数のサーバー間でインストール依存関係を持つものがある。従来の研究では、既存スクリプトや自動インストールプラットフォームへの影響を考慮して、インストール依存関係の順番で逐次的にアプリケーションサーバーをデプロイすることによって、依存関係があるアプリケーションサーバーのデプロイメントを実現している。しかし、サーバーの台数が増えるに従って、デプロイメント時間が線型的に増加するという問題点がある。
本論文では、既存スクリプトやプラットフォームへの影響を最小限に抑えて、インストール依存関係がある複数アプリケーションサーバーを並列でデプロイすることによって高速にデプロイする手法を提案する。既存スクリプトやプラットフォームに対する変更を最小限に抑えられることを示すために、自動インストールプラットフォームとして幅広く用いられている Chef を検証対象としてプロトタイプシステムを実装する。さらに、パブリッククラウド上でのデプロイメント性能評価を通して、本論文のアプローチが実用的であることを示す。

1. はじめに

1.1 背景

クラウドインフラストラクチャーサービス (Infrastructure as a Service, IaaS) では、自動的にデプロイするサーバーは基本的に OS のみがインストールされているサーバーである。クラウド上でアプリケーションがインストールされている状態のサーバー、アプリケーションサーバーが必要な場合には、クラウドが OS のみのサーバーを自動的にデプロイした後に、ユーザー自身が手動でアプリケーションをインストールする必要がある。

アプリケーションのインストールをサーバーデプロイメントのように自動で行う研究として、コンポジットアプリケーションデプロイメント方式 [1][2][3][4] が注目されている。コンポジットアプリケーションデプロイメント方式は、アプリケーションを自動的にインストールするスクリプト、アプリケーションスクリプトを各アプリケーション毎に用意し、アプリケーションスクリプトをダウンロードして実行するエージェントスクリプトを搭載したプライベートサーバーイメージを 1 つ用意する。ユーザーは、サーバーをデプロイメントする際に、インストールしたいアプリケーションを選択する。クラウドがプライベートイメージからサーバーを自動的に作成すると、サーバーが初期化された後、サーバー上のエージェントスクリプトはユーザーが選択したアプリケーションに対応したアプリケーションスクリプトをコンポジットアプリケーションデプロイメントサーバーからダウンロードし、アプリケーションスクリプトを自動的に実行する。この仕組みによって、ユーザーが選択したアプリケーションがインストールされたサーバーを自動的にデプロイすることを可能としている。

アプリケーションサーバーを自動的にデプロイする方式として、他には、アプリケーションを事前にインストールしたプライベートイメージを用意する、バーチャルアプライアンスデプロイメント方式 [5][6] が良く使われている。コンポジットアプリケーションデプロイメント方式は、バー

[1] IBM 東京基礎研究所
IBM Research – Tokyo, 135–8511, Japan
[a)] katsuno@jp.ibm.com
[b)] hitomi@jp.ibm.com

チャルアプライアンスデプロイメント方式と比べて次の 3 点で優れていると考えられている [1][2][3][4]。

- プライベートサーバーイメージの管理の容易さ
  バーチャルアプライアンスデプロイメント方式では、想定される全てのアプリケーションの組み合わせに対応するプライベートサーバーイメージを用意する必要があり（例：アプリケーションの数が 4 つの場合、15 個のプライベートサーバーイメージが必要）、プライベートサーバーイメージの管理が困難である。それに対して、コンポジットアプリケーションデプロイメント方式では、事前に用意するプライベートサーバーイメージが 1 つのため、管理が容易である。
- アプリケーションサーバーパターンの自動デプロイメントの実現
  複数のサーバーから構成されるアプリケーションサーバーパターンのデプロイメントを行う際には、サーバー間接続の設定を如何にして行うかが課題となる。なぜならば、サーバー間接続情報には、通常、クラウドによって動的にアサインされる IP アドレスが含まれるからである。コンポジットアプリケーションデプロイメント方式は、IP アドレスがアサインされ初期化が完了したサーバー上でスクリプトが実行されるので、動的にアサインされる IP アドレスを引数としてスクリプトに渡すことで、サーバー間接続を自動的に行うことが可能である。
- アプリケーションコード変更に対する柔軟な対応
  バージョンアップなどでアプリケーションのコードが変更された際に、バーチャルアプライアンスデプロイメント方式では該当するアプリケーションを含む全てのプライベートサーバーイメージを変更する必要があり、大変手間がかかる。それに対して、コンポジットアプリケーションデプロイメント方式は該当するアプリケーションのアプリケーションスクリプトのみを変更するだけなので、変更が容易である。

アプリケーションサーバーパターンの中には、サーバー間にインストール依存関係があるものがある。Web サーバーとデーターベースサーバーから構成される Web アプリケーションが典型的な例である。Web サーバーの Web アプリケーションがデーターベースサーバー上のデータベースにアクセスするために、Web サーバー、データベースサーバーそれぞれで接続設定を行う必要がある。接続設定を行う順番は接続先でありデータベースサーバー側が先、接続元である Web サーバー側が後になることを保証しなければならない。

既存研究 [8][9][10] では、サーバーを逐次的にデプロイメントすることによってこのようなサーバー間のインストール依存関係を持つアプリケーションサーバーパターンのデプロイメントを安全に行うことを提案している（図 1）。

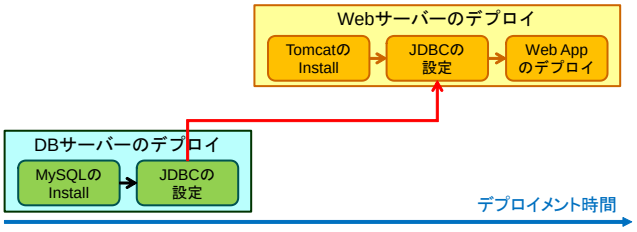

図 1　逐次型デプロイメント方式

Web アプリケーションの例では、データベースサーバーのデプロイメントを先に行い、データベースサーバーのインストール・設定が全て完了した後に、Web サーバーのデプロイメントを行う。

このような逐次型デプロイメントは、アプリケーションスクリプトが、単一サーバーのインストールを想定している、シェルスクリプトや、Chef [11] や Puppet [12] といった自動インストールフレームワークを利用していることを考慮すると当然のアプローチである。しかし、逐次型デプロイメントは、サーバーの台数が増えるに従って、デプロイメント時間が線型的に増えるという問題がある。DevOps [13] といったサーバーデプロイメントの機会が増加している現状を考慮すると、デプロイメント時間を短縮する、高速デプロイメントの要望が高まっている。

1.2　本論文の貢献

本論文では、既存スクリプトや自動インストールフレームワークへの影響を最小限に抑えて、インストール依存関係があるコンポジットアプリケーションサーバーを高速にデプロイメントする、自動並列型デプロイメント方式を提案する（図 2）。[*1] 提案手法は、まず、アプリケーションスクリプトを解析して、アプリケーションスクリプト単位のインストール依存関係を検出する。続いて、検出した依存関係の順番でアプリケーションスクリプトが実行されるのを保証するため、アプリケーションスクリプト実行の同期を管理する。既存スクリプトやフレームワークに対する変更を最小限に抑えられることを示すために、自動インストールフレームワークとして幅広く用いられている Chef を検証対象としてプロトタイプシステムを実装する。さらに、IaaS 型パブリッククラウドの 1 つである、ソフトレイヤー [14] 上で 2 種類のアプリケーションサーバーパターンを従来の逐次型アプローチと自動並列型アプローチでデプロイした実測時間を測定し、提案手法がサーバーデプロイメント時間を短縮できることを示す。

本論文の構成は次の通りである。第 2 章で設計、第 3 章でプロトタイプ実装について述べる。第 4 章でプロトタイプシステムの性能評価を行う。関連研究について第 5 章で議論する。最後に、第 6 章で結論と今後の課題について述

[*1] 本論文の概要は、3rd IEEE International Conference on Cloud Engineering (IC2E 2015) で発表された [7]。

Vol.2015-DPS-163 No.16
Vol.2015-MBL-75 No.16
2015/5/28

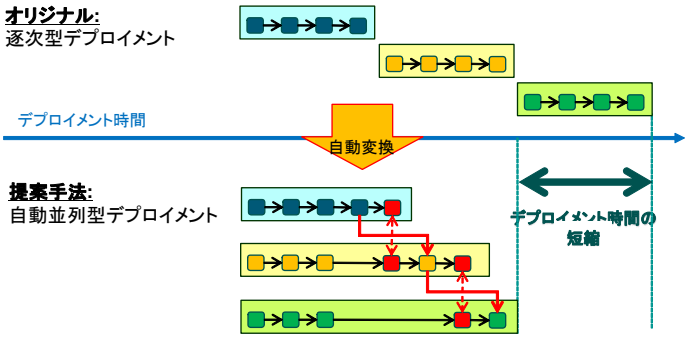

図 2　提案手法 – 自動並列型デプロイメント方式

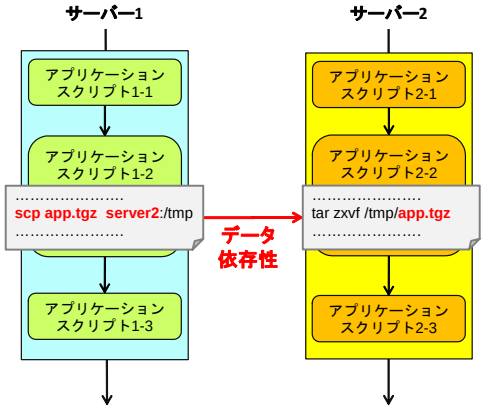

図 3　アプリケーションスクリプト間の暗黙的なデータ依存関係検出の例

べる。

2. 設計

提案手法は、同期ポイントの自動検出と、インストールの同期管理という 2 つの技術から構成される。

2.1 同期ポイントの自動検出

本技術は、まず、アプリケーションスクリプトを解析して、サーバー間のインストール依存関係を自動的に検出する。そして、インストール依存関係の保証、つまりアプリケーションスクリプト実行順序の保証を実現するために、何処に同期管理スクリプトを挿入すべきかを決定する。

インストール依存関係を検出するために、本論文では、プログラム依存グラフ手法 [15] を用いた。プログラム依存グラフはデータ依存関係と制御依存関係という 2 種類の依存関係を取り扱う。データ依存関係はプログラム実行に影響しないため、プログラムソースコードを解析すれば検出できる。制御依存関係はプログラム実行に影響を及ぼすため、プログラムソースコードとプログラム実行フローの両方を解析して検出される。本研究で対象とするプログラムはインストールスクリプトであり、インストールスクリプトは他のプログラム実行に影響を及ぼさないように開発されるのが一般的であるため、本論文では、データ依存関係に着目する。

本論文で提案するデータ依存関係検出では、まず、明示的なデータ依存関係を検出する。従来のアプリケーションサーバーパターンデプロイメント技術と同じ方法を採用し [9][10]、アプリケーションスクリプト開発者によって明記されたデータ依存関係を示すキーワードをアプリケーションソースコードから探し出す。本ステップによって、従来技術と同じく、データ依存関係を保証したデプロイメントが実現される。

続いて、提案手法ではさらに、明示的なデータ依存関係に加えて、アプリケーションスクリプトソースコードに明示的に記されていない暗黙的なデータ依存関係も検出することで、高速なデプロイメントを実現する。暗黙的なデータ依存関係として、本論文では、サーバー間のインストール作業で典型的な、ファイル転送・更新に着目した。暗黙的なデータ依存関係検出アルゴリズムは次の通りである。まず、スクリプト内で、IP アドレスを格納している変数を探し出す。続いて、IP アドレスを格納する変数を用いて、ファイルの他のサーバーへの転送、もしくは他サーバー上のファイルの更新を行っているアプリケーションスクリプトを探し出す。最後に、転送・更新されるファイル名から、ファイルの転送・更新を行うアプリケーションスクリプトと対となるアプリケーションスクリプトを探す。

図. 3 に、暗黙的なデータ依存関係検出の例を示す。まず、変数 server2 に IP アドレスが格納されていることを検出する。続いて、変数 server2 を用いる、アプリケーションスクリプト 1-2 が、scp コマンドによって、アーカイブファイル app.tgz を他のホストに転送していることを検出する。最後に、アーカイブファイル app.tgz を解凍している、アプリケーションスクリプト 2-2 を検出する。以上より、アプリケーションスクリプト 1-2 から、アプリケーション 2-2 に対して、データ依存関係があることが検出される。

2.2 インストールの同期管理

インストールの同期管理によって、複数のサーバーにまたがるアプリケーションスクリプトの実行順番をデータ依存関係に基づいて保証する。

本論文では、並列プログラミングでよく用いられる、チェックポイント技術 [16] に基づいて、インストール同期管理を実現する。チェックポイント技術では、作業進捗情報をミリ秒といった単位で迅速に共有するために、CoCheck [16] といった通信プロトコルが用いられる。しかし、本論文が対象とするインストール作業では、秒単位の同期で十分であり、かつ既存システムへの影響を最小限にするために、新たに新しい通信プロトコルを導入するのではなく、既に用いられている通信手法を利用する。この方式は、同期の情報共有のために新規サーバーを立ち上げず、既存サーバーを利用することができるようになる。

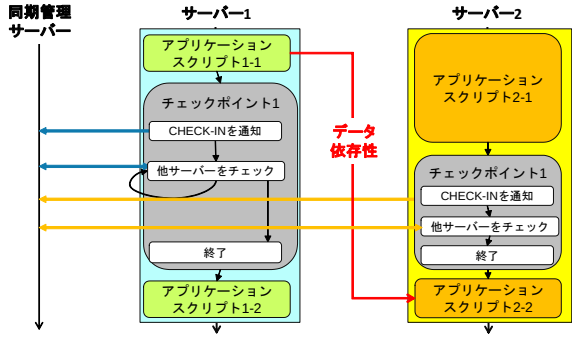

図 4　チェックポイントに基づく、インストールの同期管理

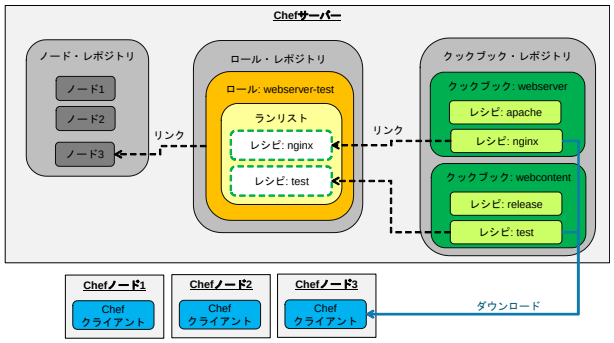

図 5　Chef クライアント・サーバーアーキテクチャ

図 4 に、チェックポイントに基づくインストールの同期管理を示す。サーバー 1 で実行される、アプリケーションスクリプト 1-1 から、サーバー 2 で実行されるアプリケーションスクリプト 2-2 に対してデータ依存関係があるため、アプリケーションスクリプトの実行順番は、1-1 が先で、2-2 が後にならなければならない。この順番を保証するために、アプリケーションスクリプト 1-1 の後と、アプリケーションスクリプト 2-2 の前に、チェックポイント 1 スクリプトが挿入される。チェックポイント 1 に入ると自身がチェックポイント 1 に入ったことを同期管理サーバーに通知し、相手のサーバーがチェックポイント 1 に入るまで待つ。

3. 実装

本論文では、既存システムに対する変更を最小限に抑えられることを示すために、自動インストールプラットフォームとして幅広く用いられている Chef [11] を検証対象としてプロトタイプシステムの実装を行った。

3.1 Chef

Chef は、Ruby と Erlang で実装された、自動インストールプラットフォームである。Chef のクライアント・サーバーアーキテクチャを図 5 に示す。

レシピは、コンポジットアプリケーションデプロイメントにおけるアプリケーションスクリプトに該当する。レシピは、使用用途・目的に応じて、クックブックとして分類される。レシピを適応するサーバーは、ノードとして管理される。ノード上では、Chef クライアントが起動して、Chef サーバーをポーリングしている。ノード上で Chef レシピを実行するには、まずレシピの実行順番をリストしたランリストを作成し、ロールを定義する。続いて、ロールを対象のノードにアサインする。すると、対象ノード上の Chef クライアントが自身にロールがアサインされたことを知り、ロール内のランリスト内のレシピファイルをダウンロードして、ランリストに書かれた順番で、レシピを実行する。

図 5 では、nginx をインストールするレシピ：nginx と、本番用 Web コンテンツをロードするレシピ：release から作成されたランリストを定義した、ロール：webserver-test を作成し、ノード 3 にアサインしている。すると、ノード 3 上の Chef クライアントは、レシピ：nginx、release をダウロードして、nginx のインストールと、本番用 Web コンテンツのロードを行う。

3.2 同期ポイントの検出

本論文では、既存の Chef システム・コードへの影響を最小限に抑えるため、同期実行の単位を Chef レシピにした。各レシピを解析して、レシピ間のデータ依存関係を検出すると、ランリスト内の該当レシピの前後にチェックポイントレシピを挿入する。

なお、明示的なデータ依存関係、暗黙的なデータ依存関係、共に、デッドロックは起きないことを仮定する。

3.2.1 明示的なデータ依存関係の検出

明示的なデータ依存関係は、レシピやクックブックを作成する開発者が記述する。

Chef では、同一クックブック内のレシピ間のデータ依存関係を明示的に記述するために、”`depends`” という予約語を定義し、クックブックのメタデータを定義するファイル：`metadata.rb` に記述する。予約語：”`depends`” は、単一サーバー上における作業のみが対象となっている。

本論文では、複数のサーバー間の作業を対象とするデータ依存性の記述に対応するために、新たな予約語：”`node-depends`” を定義し、各クックブックのメタデータ定義ファイル：`metadata.rb` のコメントヘッダーに挿入した。そのため、明示的なデータ依存関係は、ランリストにリストされている各レシピに関するクックブックの `metadata.rb` ファイルをチェックすることによって検出できる。

図 6 に、例を示す。Cookbook2 に、Cookbook3 とのデータ依存関係が記述されている。サーバー 1 とサーバー 2 のランリスト：Runlist1 と Runlist2 を見ると、Runlist1 に Cookbook2 に関する Recipe2 が、Runlist2 に Cookbook3 に関する Recipe3 が、それぞれリストされていることが分かる。そのため、チェックポイントレシピ：Checkpoint1

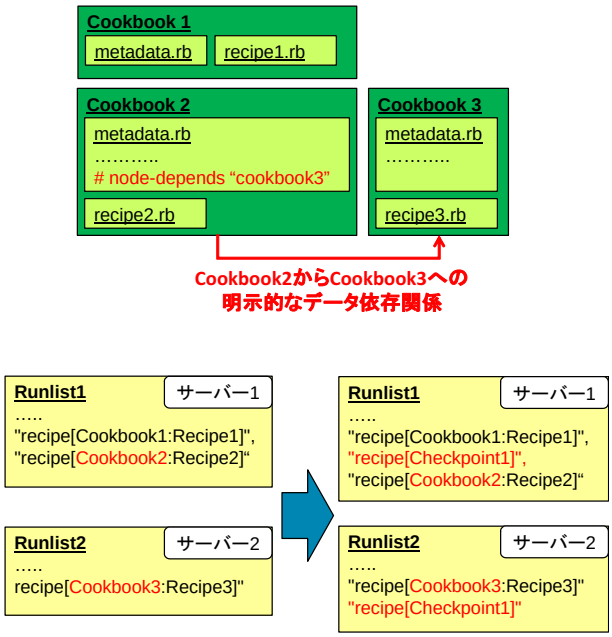

図 6　Chef における明示的なデータ依存関係の検出

を、ランリスト：Runlsit1 の Recipe2 の前、ランリスト：Runlist2 の Recipe3 の後に、それぞれ挿入する。

3.2.2　暗黙的なデータ依存関係の検出

本論文で対象としている暗黙的なデータ依存関係は、サーバー間のファイルの転送と更新であるため、最初に、サーバー間の操作を特定するため、IP アドレスを格納する変数を検出する。Chef には、レシピ起動時に値を指定でき、レシピ内で変数として取り扱うことができる、*Chef Attribute* という概念がある。クラウドでは、一般的に IP アドレスはサーバーデプロイメント時に動的にアサインされるので、IP アドレスが *Chef Attribute* に格納されていると想定し、IP アドレスを格納している *Chef Attribute* の検出する。*Chef Attribute* は、Chef ランリスト定義ファイル、Chef ロール定義ファイル、Chef クライアントコマンドのための JSON ファイル、Chef サーバーで管理されている Chef 属性データのいずれか宣言されてため、それぞれを調べ IP アドレスが格納されている *Chef Attribute* を検出する。

続いて、IP アドレスが格納されている *Chef Attribute* を使用して、かつファイル転送・更新を行う操作を行うレシピを検出する。

最後に、転送・更新される対象となるファイル名から、ファイル転送・更新を行うレシピとデータ依存関係となるレシピを、検出する。

図 7 に、例を示す。まず、*Chef Attribute* として定義されている、#{node[”vm2”][”ip”]} に IP アドレスが格納されていると検出される。続いて、#{node[”vm2”][”ip”]} を使用し、他のサーバーへのファイル転送を行う、レシピ：Recipe1-1 が検出される。最後にレシピ：Recipe1-1 で転送するファイル：*bar.tgz* を使用するレシピ：Recipe2-1 を

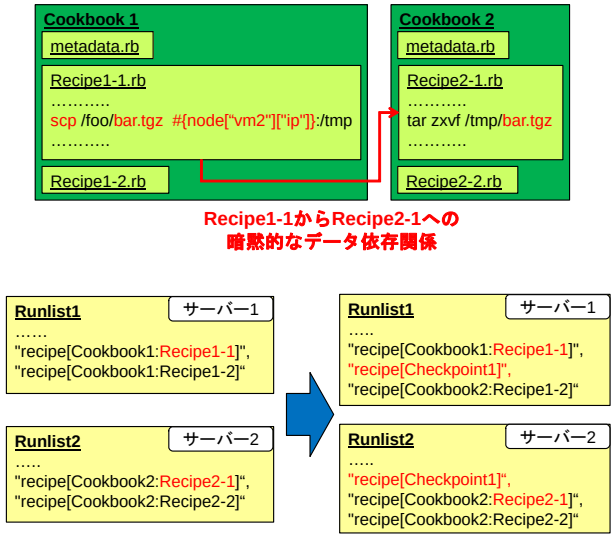

図 7　Chef における暗黙的なデータ依存関係の検出

検出する。以上より、レシピ：Recipe1-1 から、レシピ：Recipe2-1 に対して、データ依存関係があることが分かる。そのため、チェックポイントレシピ：Checkpoint1 を、ランリスト：Runlist1 の Recipe1-1 の直後と、ランリスト：Runlist2 の Recipe2-1 の直前に、それぞれ挿入する。

3.3　インストール作業の同期管理

チェックポイントに基づく、インストール作業の同期を管理するためには、インストール進捗状況を管理するサーバーが必要となる。既存環境への影響を最小限にするためには、新たにサーバーを立ち上げるよりも、既存環境で用いられているサーバーを利用する方が適している。

そこで本論文では、Chef サーバーを用いて、チェックポイントに基づくインストール作業同期機構を実装した。インストール進捗状況情報は、Chef の基本機能である *Chef Tag* を用いた。*Chef Tag* は、Chef サーバーで各 Chef ノード毎に属性の一つとして管理され、Chef コマンドを使うことによって、Chef ノードから値の更新・検索を行うことができる。そのため、各サーバーのインストール進捗状況管理に利用できる。*Chef Tag* は、名前と値のペアではなく、値のみで表現される。そこで、本論文では、インストール進捗状況を、デプロイメント ID：ID_{deploy}、チェックポイント ID：$ID_{checkpoint}$、サーバー状態：$STATUS_{server}$ を連結した、ID_{deploy}-$ID_{checkpoint}$-$STATUS_{server}$ で表現した。

図 8 に、具体例を示す。サーバー 1 の Chef レシピ：Recipe1-1 と、サーバー 2 の Chef レシピ：Recipe2-2 にはデータ依存関係があるため、Recipe1-1 の直後と Recipe2-2 の直前に、チェックポイントレシピ：Checkpoint1 が挿入されている。サーバー 1 では、Recipe1-1 の実行が完了し、Checkpoint1 を実行すると、まず、自身がチェックポイントに入ったことを Chef サーバーに通知するために、

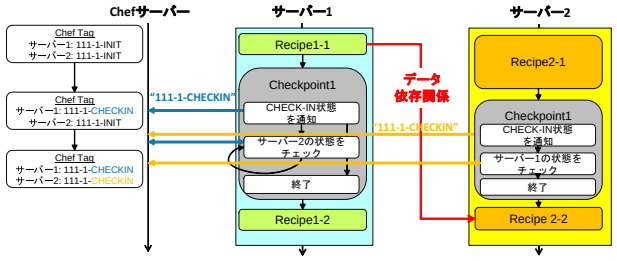

図 8　Chef におけるチェックポイントに基づくインストール作業同期管理

111-1-*CHECKIN* を Chef サーバーに送付する。すると、Chef サーバーは、サーバー 1 のインストール進捗状況：111-1-*INIT* を 111-1-*CHECKIN* に更新する。その後、サーバー 1 は、同一のデプロイメント ID とチェックポイント ID、つまり、111-1 を持つ他の全てのサーバーの状態が、*INIT* から *CHECKIN* に更新されるまで待つ。一方、サーバー 2 では、Recipe2-1 の実行が完了し、Checkpoint1 を実行すると、111-1-*CHECKIN* を Chef サーバーに送付し、Chef サーバーは、サーバー 2 のインストール進捗状況：111-1-*INIT* を 111-1-*CHECKIN* に更新する。その後、サーバー 1 とサーバー 2 共に、同一のデプロイメント ID とチェックポイント ID：111-1 を持つ他の全てのサーバーの状態が *CHECKIN* に更新されたことを認識し、Checpoint1 を終了させる。

4. 評価

本論文では、次の 2 種類のアプリケーションサーバーパターンを、Chef オリジナルの逐次型アプローチと、提案方式である並列型アプローチで、それぞれデプロイした実測時間を測定した。

- 典型的な Web アプリケーション
  Web サーバー、データベースサーバーの 2 台構成で、Web サーバー上で EAR (Enterprise ARchive) 形式のアプリケーションを動かす。
- ソーシャルメディア分析 [17]
  データサーバー、アナリティクスサーバー、インターフェースサーバーの 3 台構成で、複数の Web メディアサイトからニュースやブログ記事を収集・分析することによって、消費者インサイトを見つけ出す。

サーバーをデプロイするクラウド環境として、IaaS 型パブリッククラウドのソフトレイヤー [14] を用いた。サーバーの仕様は、4 コア・プロセッサー、48Gbyte メモリー、100GByte ディスクの仮想サーバーで、OS は、Red Hat Enterprise Linux version 6.5、である。

図. 9 と図 10 にそれぞれ Web アプリケーションとソーシャルメディア分析の結果を示す。Web アプリケーションでは 41.88%、ソーシャルメディア分析では 38.56%のデプロイメント時間の短縮を実現した。本結果によって、本論文の提案方式が高速デプロイメントに有効であることが示せた。

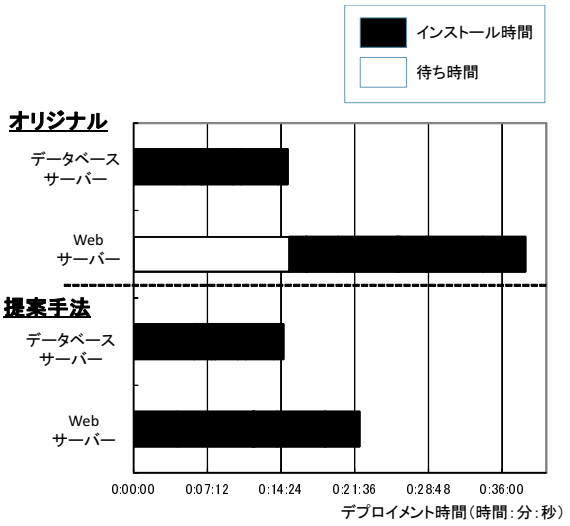

図 9　デプロイメント時間の比較 – 典型的な Web アプリケーション

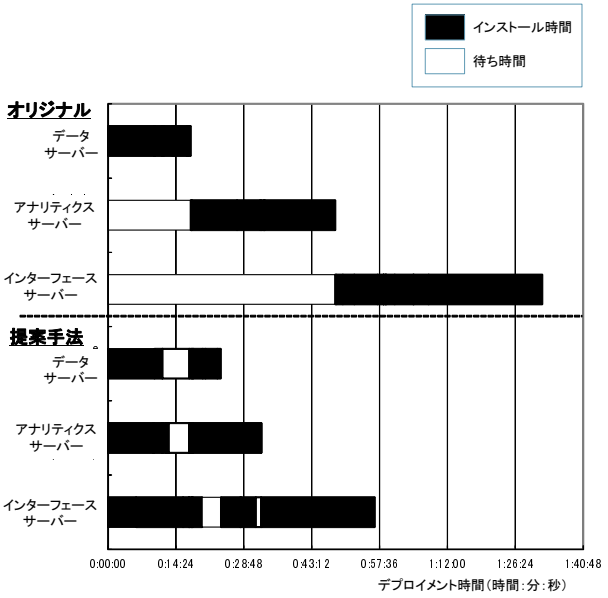

図 10　デプロイメント時間の比較 – ソーシャルメディア分析

5. 関連研究

アプリケーションサーバーの柔軟なデプロイメントを実現するために、コンポジットアプリケーションデプロイメント方式に着目した研究が幾つかある。デプロイメント記述性を高めるために、Wrangler システム [8] は XML で、OpenStack Heat [9] と AWS CloudFormation [10] は JSON や YAML を使用してパターンを記述することを提案している。これらは全て、インストール依存関係があるコンポジットアプリケーションサーバーパターンに対して、サーバー間の依存関係をパターンテンプレートに明示的に記述することによって、安全なデプロイメントを実現している。しかし、インストール同期単位がサーバーであるた

め、サーバーのデプロイメントが逐次的になり、サーバーの台数が増えるに従って、デプロイメント時間が長くなる。本論文では、インストール同期単位をサーバーよりも細かいアプリケーションスクリプトにすることによって、サーバーのデプロイメントを並列化し、高速なデプロイメントを実現した。

BOSH [18] など幾つかのコンポジットアプリケーションパターンデプロイメントフレームワークは、複数サーバー間のインストール同期管理機構を持たない。CloudFoundryのような複数のサーバー間でインストール依存関係を持たないアプリケーションをデプロイメントするのには向いているが、インストール依存関係を持つアプリケーションを安全にデプロイメントできない。本論文の提案方式は、既存のデプロイメントフレームワークに依存せず、既存システムアーキテクチャ・コンポーネントを変更しない。そのため、提案方式をこのようなデプロイメントフレームワークに対して適用することによって、インストール同期管理機構を持たせることが可能となると考えられる。

アプリケーションサーバーの高速デプロイメントを実現している研究が幾つかある。アプリケーションのインストールに用いられる巨大なファイルを転送するには時間がかかるため、Zhijia ら [19] や Johannes ら [20] は、ピア・トゥ・ピア・ネットワークを利用して、転送時間の短縮を提案している。Manish ら [21] は、基本的なアプリケーションがインストールされているプライベートサーバーイメージを事前に用意することによって、実行するアプリケーションスクリプトの数を減らすことを提案している。Suresh ら [22] は、CPU と GPU を組み合わせることによって、アプリケーションスクリプトの高速実行を提案している。Arijit ら [23] は、デプロイメント時に指定されるパラメーターの数を減らして、人が行うデプロイメント作業の時間を短縮することによって、高速なデプロイメントを実現することを提案している。本論文の提案手法を、これらと組み合わせることによって、更に高速なデプロイメントが実現できる。

本論文では、コンポジットアプリケーションサーバーの構成・設定が正しいことを前提としている。しかし、サーバーをデプロイメントする前に、構成・設定が正しいことを検証するのは重要である。Tamar ら [24] は、モデル駆動アーキテクチャに基づいて構成・設定を検証することを提案している。アプリケーションスクリプトの検証も重要で、自動的に行われることが望ましい。検証ツールを用いてスクリプトの検証を自動的に行う手法が提案されている [25][26]。

本論文では、アプリケーションスクリプト間のインストール依存関係はアプリケーション実行環境に関わらず不変であり、スクリプトソースコードを分析することによって検出できることを想定している。スクリプトソースコードに加えてスクリプト実行フローも分析することによって制御依存関係 [15] をサポートし、動的な依存性検出 [27] を行うことによって、インストール依存関係が実行中に変化する、より複雑な依存関係に対応できるようになる。

本論文の並列実行単位はアプリケーションスクリプトである。プログラム自動パーティショニング技術 [28][29] やプログラムスライシング技術 [30] を用いて、アプリケーションスクリプトを分割し、並列実行単位を小さくすることによって、並列性をさらに高め、更に高速にデプロイメントできるようになると考えられる。

6. おわりに

本論文では、複数のサーバー間にインストール依存関係があるコンポジットアプリケーションサーバーを高速にデプロイメントするために、アプリケーションスクリプトを解析してアプリケーションスクリプト間のデータ依存関係を自動的に検出し、チェックポイントに基づく同期管理機構によって依存関係があるアプリケーションスクリプトを安全に実行する、自動並列型デプロイメント方式を提案した。そして、広く普及する自動インストールフレームワーク、Chef 上でプロトタイプシステムを実装することによって、既存環境に対する変更を最小限に抑えられることを示した。さらに、ソフトレイヤークラウド上で、2 種類のコンポジットアプリケーションパターンを従来アプローチと提案アプローチでデプロイした実行時間を測定し、デプロイメント時間が短縮できることを実証した。

本論文では、Chef 上でプロトタイプシステムを実装したが、Puppet といった他の自動インストールフレームワーク上でもプロトタイプシステムを実装し、提案手法がフレームワークに依存しないことを実証するのは有益である。また、コンポジットアプリケーションデプロイメント方式とプライベートイメージデプロイメント方式 [5][6] を組み合わせることによって、更に高速なデプロイメントを目指すのも有益である。

参考文献

[1] Alexander V. Konstantinou, Tamar Eilam, Michael Kalantar, Alexander A. Totok, William Arnold, and Edward Snible. An architecture for virtual solution composition and deployment in infrastructure clouds. In *Proceedings of the the 3rd international workshop on virtualization technologies in distributed computing (VTDC 2009), June, 2009.*

[2] Xavier Etchevers, Thierry Coupaye, Fabienne Boyer, and Noel de Palma. Self-configuration of distributed applications in the cloud. In *Proceedings of the 4th IEEE International Conference on Cloud Computing (Cloud 2011), July, 2011.*

[3] Vanish Talwar, Qinyi Wu, Calton Pu, Wenchang Yan, Gueyoung Jung, and Dejan Milojicic. Comparison of approaches to service deployment. In *Proceedings of the 25th IEEE International Conference on Distributed*

Computing Systems (ICDCS 2005), June, 2005.

[4] Tamar Eilam, Michael Elder, Alexander V. Konstantinou, and Ed Snible. Pattern-based composite application deployment. In *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management 2011 (IMS 2011), May, 2011.*

[5] Constantine Sapuntzakis, David Brumley, Ramesh Chandra, Nickolai Zeldovich, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Virtual appliances for deploying and maintaining software. In *Proceedings of the 17th Large Installation Systems Administration Conference (LISA 2003), October, 2003.*

[6] Changhua Sun, Le He, Qingbo Wang, and Ruth Willenborg. Simplifying service deployment with virtual appliances. In *Proceedings of IEEE International Conference on Services Computing (SCC 2008), July, 2008.*

[7] Yasuharu Katsuno, Hitomi Takahashi. An Automated Parallel Approach for Rapid Deployment of Composite Application Servers. In *Proceeding of the 3rd IEEE International Conference on Cloud Engineering (IC2E 2015), March, 2015.*

[8] Gideon Juve and Ewa Deelman. Automating application deployment in infrastructure clouds. In *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011), November, 2011 .*

[9] Openstackheat. https://wiki.openstack.org/wiki/Heat.

[10] Amazon cloudformation. http://aws.amazon.com/cloudformation/.

[11] Matthias Marschall. Chef infrastructure automation cookbook. In *Packt Publishing, August, 2013.*

[12] Puppet. https://puppetlabs.com/.

[13] Michael Httermann. Devops for developers. In *Apress, Berkely, CA, 2012.*

[14] Softlayer. http://www.softlayer.com/.

[15] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. In *ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 9, Issue 3, July 1987.*

[16] Jim Pruyne and Miron Livny. Managing checkpoints for parallel programs. In *Proceedings of the 10th International Parallel Processing Symposium (IPPS 96), April, 1996.*

[17] Social media analytics installation and configuration guide. http://public.dhe.ibm.com/software/data/cognos/documentation/docs/en/1.3.0/inst_ci.pdf.

[18] Cloudfoundry bosh. http://docs.cloudfoundry.org/bosh/.

[19] Zhijia Chen, Yang Zhao, Xin Miao, Ying Chen, and Qingbo Wang. Rapid provisioning of cloud infrastructure leveraging peer-to-peer networks. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems Workshops, June, 2009.*

[20] Johannes Kirschnick, Jose M. Alcaraz Calero, Patrick Goldsack, Andrew Farrell, Julio Guijarro, Steve Loughran, Nigel Edwards, and Lawrence Wilcock. Towards an architecture for deploying elastic services in the cloud. In *Journal of Software Practice Experience, Volume 42 Issue 4, April 2012.*

[21] Manish Sethi, Kalapriya Kannan, Narendran Sachindran, and Manish Gupta. Rapid deployment of soa solutions via automated image replication and reconfiguration. In *Proceedings of 2008 IEEE International Conference on Services Computing (SCC 2008), July, 2008.*

[22] Suresh Boob, Horacio Gonzalez Velez, and Alina Madalina Popescu. Automated instantiation of heterogeneous fastflow cpu/gpu parallel pattern applications in clouds. In *Proceedings of the 22nd IEEE Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2014), Feburary, 2014.*

[23] Arijit Ganguly, Jian Yin, Hidayatullah Shaikh, David Chess, Tamar Eilam, Renato Figueiredo, Jim Hansom, Ajay Mohindra, and Giovanni Pacifici. Reducing complexity of software deployment with delta configuration. In *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007), May, 2007.*

[24] Tamar Eilam, Michael H. Kalantar, Alexander V. Konstantinou, and Giovanni Pacifici. Reducing the complexity of application deployment in large data centers. In *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005), May, 2005.*

[25] Waldemar Hummer, Florian Rosenberg, Fabio Oliveira, and Tamar Eilam. Automated testing of chef automation scripts. In *Proceedings of ACM/IFIP/USENIX 14th International Middleware Conference, December, 2013.*

[26] Waldemar Hummer, Florian Rosenberg, Fabio Oliveira, and Tamar Eilam. Automated testing of chef automation scripts. In *Proceedings of ACM/IFIP/USENIX 14th International Middleware Conference, December, 2013.*

[27] Todd M. Austin and Gurindar S. Sohi. Dynamic dependency analysis of ordinary programs. In *Proceedings of the 19th annual international symposium on Computer architecture (ISCA 92), May, 1992.*

[28] Mark S. Squillante. On the benefits and limitations of dynamic partitioning in parallel computer systems. In *Proceedings of the 9th International Parallel Processing Symposium (IPPS 95), April, 1995.*

[29] V. Sarkar. Automatic partitioning of a program dependence graph into parallel tasks. In *IBM Journal of Research and Development, Volume 35, Issue 5.6, September, 1991.*

[30] Keith Brian Gallagher and James R. Lyle. Using program slicing in software maintenance. In *IEEE Transactions on Software Engineering, Volume 17, No. 8, August, 1991.*

付　　録