Vol.2020-DPS-182 No.55
Vol.2020-CSEC-88 No.55
2020/3/13

OSの自動インストール設定の対話型作成と統一記法

小山 智之[1]　串田 高幸[1]

概要：同時に複数マシンのセットアップを行う場面では，作業時間の削減や設定ミスを防ぐ目的でOSインストールの自動化が行われる．しかし，自動化のための手順や設定記法はOSごとに異なるため，オペレータにOSごとの手順や設定記法を覚える負担が生じる．本研究では，設定記法の統一化と対話型インターフェースにより，この課題を解決する．実験では仮想マシン上で，統一設定に基づく自動インストールが正常に実行可能か異なる2種類のOSで検証した．さらに，既存技術の組み合わせと作成する設定ファイル数，記述する言語数，記述するパラメータ数を比較した．その結果，記述するパラメータ数を約77%削減した．

1. はじめに

クラウド技術の普及によりこれまで以上にマシンのライフサイクルが活発な状況にある．クラウド技術の利用により従来型のオンプレミスによる課題が解消されている．例えば，ハードウェアの管理やそれに伴うソフトウェアの管理があげられる．ハードウェアはその性質上，故障や耐用年数による定期的な交換が発生する．オンプレミスではこれに対応するため，エンジニアの雇用や保守契約の費用が発生する．また，ハードウェアの交換に伴いソフトウェア設定の変更が発生する [1]．クラウド技術はこうした付加価値を生まない作業にかかる時間を削減する．

クラウド技術の構築・運用を行う立場では，複数台の物理マシンをセットアップする状況は依然として発生する．特に提供サービスの規模が拡大するたびにマシンの台数は線形に増加する．これらマシンは必ず同一の環境 (ハードウェアスペックやOS) とは限らず，マシンが増えるにつれて個々のマシンの性能や環境のばらつきが生じ，それに伴うオペレータの設定の手間も増加する [2]．特に，これらマシンのセットアップを個別に手動で行う場合，オペレータの作業量はより増加する．個々に異なるパラメータをオペレータが手動で設定する場合，設定パラメータの種類と設定パターンが増えるにつれて設定不備の発生する可能性が高まる．設定不備は不正アクセスを許し，セキュリティホールにもつながる [3]．オペレータの作業量削減とセキュリティ対策の観点から，オペレーションの自動化は必要不可欠である．

こうした自動化を実現するツールには，ベンダーが提供するものがある．これらは，自作やサードパーティ製ツールとの拡張性の低さや自動化しにくいGUI，一部で手動作業が必要となる不完全さの問題を抱えている．オペレータは，システムのインストールやアップデートのたびにこうした課題と向き合う必要がある [4]．つまり，ベンダーの製品では十分に柔軟な自動化が実現できない状況にある．

大規模システムを運用するオペレータほど，自動化ツールを独自で開発し，修正を繰り返しながら運用を行う．こうしたツールは，システムごとに作成されるため，汎用性がない．また，こうしたツールは処理内容を把握した作成者のみ知る，属人化の温床となる恐れがある [5]．すなわち，ツールは簡単なインストールや将来の変更やシステム独自のカスタマイズを行える簡単な仕組みである必要がある [6]．しかし，こうした条件を十分に満たすツールや仕組みは存在しない．

マシンのセットアップにおける作業はいくつかある．本研究はその中でもOSのインストールに着目した．その理由は，著者自身が複数台の仮想マシンへのOS自動インストールにおいて，OSごとに異なる互換性のない設定記法の作成に直面した為である．

本研究では，記法の違いを吸収する統一設定記法の提案を行う．さらに，この統一設定記法の設定ファイルの作成を簡単にするために設定プログラム (以下，設定UI) を提案する．提案をもとに実装を行い，設定ファイルの項目数やパラメータ名を比較することで評価を行った．2章では関連研究を取り上げる．関連研究の課題をふまえ3章では新たな提案を示す．4章では提案をもとに実装を行う．実装をもとに5章では検証を行う．6章では評価を，7章および8章では議論と結論，今後の課題を示す．

[1] 東京工科大学コンピュータサイエンス学部
CDSL, TUT, Hachiouji, Tokyo 101–0062, Japan

2. 関連研究

OS のセットアップを自動化する取り組みは，独自にツールを開発する手法や既存ツールをカスタマイズする手法が用いられてきた．

数百台のマシンを対象に自動による Linux インストールと設定を行うシステム ”LCFG” は，Solaris Jumpstart を Bootstap として使用した [3]．これにより，OS ごとに異なる初期の作業手順の統一を提案した．また，設定をオブジェクト指向形式で表現することで，ハードウェアごとに異なるパラメータの差分の吸収を提案した．この提案では，単一の記述記法を用いた複数の異なる OS 用の設定ファイル生成を実現した．しかし，設定ファイルの記述形式が独自のものであるため分かりやすさに課題がある．

大規模クラスタを対象に高速なセットアップと管理を実現するツール ”Lucie” は，既存ツール (ISC DHCP, Kickstart) と独自ツール Lucie を使い，250 台のマシンへ OS インストールとソフトウェアパッケージの導入を提案した [7]．Lucie では独自の設定形式の設定ファイルを NFS サーバへ配置し，それを Lucie クライアントが取得して自動的にインストールを行う．この手法では OS インストールのために軽量 OS を起動することでハードウェアをはじめとするプラットフォーム依存を解消した．欠点は，設定ファイル記法が独自であるため新たな学習が必要となることである．

システム管理ツール Puppet と自動インストールツール Kickstart，DHCP サーバ ISC DHCP の利用した研究では，Linux インストールと設定の自動化を提案した [8]．この手法では，DHCP サーバの設定を Puppet により生成し，OS の自動インストールを Kickstart でおこなった．さらに，ソフトウェアパッケージの導入を Puppet により実施した．この提案では，設定ファイルの自動生成により設定作成の手間を削減した．しかし，OS のインストール自動化では Kickstart を利用したため，RedHat 系 OS 以外では使用することができない．そのため，汎用性に課題がある．

独自のプログラムと既存ソフトウェア (Kickstart, ISC DHCP) による自動インストールでは，インストールの高速化を提案した [9]．この研究では，設定する対象マシンごとの設定ファイルの生成をプログラムで並列に行った．これにより，従来のツールに比べ高速な設定を実施した．この手法でも Kickstart を使用するため，RedHat 系 OS 以外では使用することができず汎用性に課題がある．

LCFG と Lucie は設定の記述形式が独自であるため，設定の作成には独自の設定形式の学習が必要となる．オペレータがテキストエディタで設定を作成・編集する場合，独自の設定形式は一般的な形式に比べ習得に手間がかかる．この方式では，設定ファイルの作成はドキュメントを読みながら対応する設定パラメータを記述する．特にパラメータ同士の関連やパラメータのバージョン差を理解して記述するには，十分な知識を必要とする．また，従来の手法では設定の自動化ツールが一部の OS にのみ対応していた [8][9]．これでは，限られた条件のもとでしか，設定の自動化を実現できない．本来は効率化を図るためのツールが，OS 選択時の技術的な制約を生む．

3. 提案

本研究では，対話型の設定インターフェース（以下，設定 UI）と設定記法（以下，統一記法）を提案する．提案の概要を図 1 に示す．オペレータは設定 UI を使用して CUI から対話型で設定を投入する．設定 UI で作成された設定は変換サーバへ送信される．変換サーバでは，受け取った設定を OS ごとに対応する形式へ変換を行う．また，同時に統一記法にも変換を行う．この統一記法は設定 UI へ読み込ませ編集することができる．

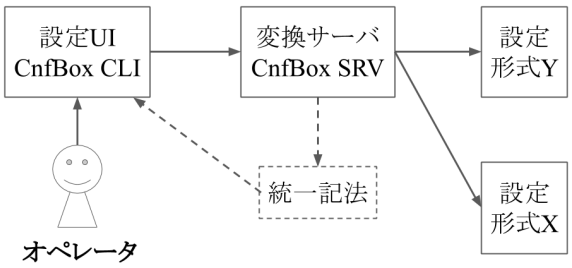

図 1　提案の概要

3.1 設定 UI

設定インターフェースは，ネットワーク機器でみられる CUI による対話型や，設定ファイルをテキストエディタで作成してコマンドプログラムへ渡すファイル記述型，Web やデスクトップアプリケーションで設定を行う GUI 型がある．設定インターフェースは記法やツールに依存しやすい．オペレータはツールにあわせて設定記法を選ぶ必要がある．GUI はコマンドを覚える必要がないため，コマンドを知らないオペレータでもオペレーションが行える．代表的な GUI インターフェースに Web がある．これは機器の設定コンソールへ Web ブラウザからアクセスし，グラフィカルなインターフェースで設定が行える．一方で，GUI は自動化が行いにくくオペレータの手動による設定が必要になる．これは，同時に複数台の機器へ設定を行う場合に手間となる．CUI はオペレータがコマンドを覚えれば高速な設定が行える．GUI では画面遷移やクリックが発生するが，CUI ではキーボード操作のみで設定が完結する．また，SSH をはじめとする汎用的なプロトコルを採用している為，ツールによる自動化が行いやすい．一方，設定にはコマンドを覚える必要があるため，オペレータの知識が要

求される．また，コマンドがベンダーやツールにより異なる場合，新たな学習が必要となる．

本提案では，設定 UI として自動化のしやすさと他のツールとの組み合わせやすさから CUI を使用する．この CUI には従来の CUI の課題を改善するため，設定値入力時の補完機能を提案する．

設定パラメータは設定項目 (Key) と設定値 (Value) のペアで構成される．設定を作成する場合，設定項目をもとにそれに対応する設定値を記述する．オペレータがドキュメントや Web サイトを見ながら設定項目から，対応する設定値を探す作業は手間がかかる．本提案では，設定パラメータを入力する際の補完を導入する．これにより，設定パラメータ指定にかかる手間を削減する．

3.2 統一記法

これまでの研究では，設定記法に独自のオブジェクト指向記法や関数記法を採用している [3][7]．こうした独自の設定記法は，新たなツールを使うたびにオペレータに学習を必要とする．こうした課題をふまえ，設定の記述を統一化する記法を提案する．本提案では，設定記法として木構造データによる記法を使用する．木構造を採用した理由は，設定パラメータが増加した場合にグループ化することで設定パラメータを整理しやすくするためである．提案記法の構造の例を図 2 に示す．設定は根 (root) から設定パラメータごとの節 (network, os, storage) に分類される．節は，根と同様に葉 (language, keyboard layout, timezon) に分岐する．設定パラメータへは根 (root) から節へ下り，さらに節から葉へ下ることでアクセスする．

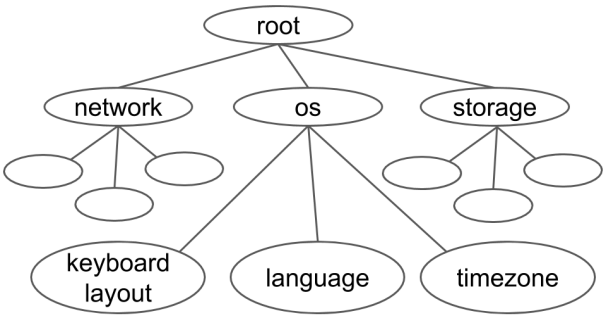

図 2　統一記法の構造

設定ファイルを読む際，設定項目が何を意味しているか分かりづらい場合がある．例えば設定項目 pwd はパスワードを表す password か present working directory か判別ができない．同様に temp も Temporary か Temperature か判別ができない．こうした事前に定義された設定パラメータは設定項目の曖昧な表現はオペレータの理解を妨げる．本提案では，新たに設定を記述するための統一記法を提案する．統一記法では設定項目の名前の省略を排除する．これにより理解のしやすさを向上させる．

4. 実装

提案をもとに設定 UI と統一設定記法を実装する．さらに，設定ファイルの変換を行う変換サーバを実装する．以下では，設定 UI を CnfBox CLI，変換サーバを CnfBox SRV とする．

4.1 設定 UI

設定 UI を提供するクライアントプログラム CnfBox CLI を実装した．アーキテクチャを図 3 に示す．このプログラムはオペレータのための設定インターフェースを提供する．設定を送信するコマンドを実行することで，CnfBox SRV へオペレータの投入した設定を送信する．プログラミング言語には Python 3.7 を．サーバとの通信プロトコルには HTTP を使用した．設定 UI プログラムはあらかじめ実装したコードを Python コマンドの-i オプションで呼び出し実行する．動作の様子をコード 1 に示す．

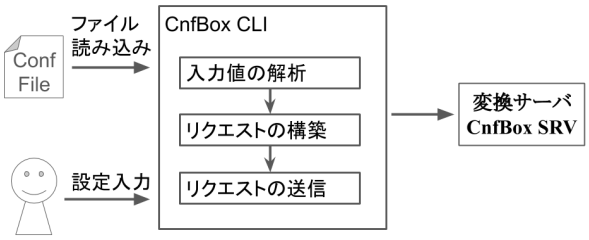

図 3　CnfBox CLI アーキテクチャ

コード 1　CnfBox CLI 動作の様子

```
>>> cb.show()  # 設定の確認
os: {'name': 'centos', 'version': 8.0, 'lang':
     'us', 'timezone': 'Asia/Tokyo', 'keyboard
    ': 'jp'}
(略)
>>> cb.os.  [Tab]  # Tabキーによる補完
cb.os.keyboard_layout
cb.os.language_supported
cb.os.timezone cb.os.language
cb.os.name  cb.os.version
>>> cb.os.keyboard_layout  # 個別設定を確認
'jp'
>>> cb.os.keyboard_layout = 'us'
    # 個別設定を変更
>>> cb.commit()  # 設定の送信
{"id":18,"status":"ok"}
```

4.2 統一記法

統一記法の実装には，木構造を表現可能な JSON 形式を採用した．Python では，json モジュールのインポートにより利用できる．記法の例をコード 2 に示す．例では，OS に関する設定を辞書型をネストすることで表現している．

コード 2 統一記法の例

```
{
  "os": {
    "keyboard_layout": "jp",
    "language_supported": "jp",
    "name": "centos",
    "timezone": "Asia/Tokyo",
    "version": 8
  },
  (略)
}
```

4.3 CnfBox SRV: 変換サーバ

設定ファイルの生成，公開を行うサーバープログラム CnfBox SRV を実装した．このプログラムは次の 2 つの機能をもつ．アーキテクチャを図 4 に示す．変換サーバは，設定 UI から受け取った設定情報を解釈し，OS ごとに適した設定に変換し生成する．また，配信機能により OS インストーラへの設定ファイルの配信を行う．配信する設定形式ごとに異なる設定パラメータがある場合，プログラムにより修正を行う．また，パスワードやトークンをはじめとする秘密情報のハッシュ化も行う．データは Database に保管し，エンドポイントを変えることで必要な設定形式へアクセスできる．

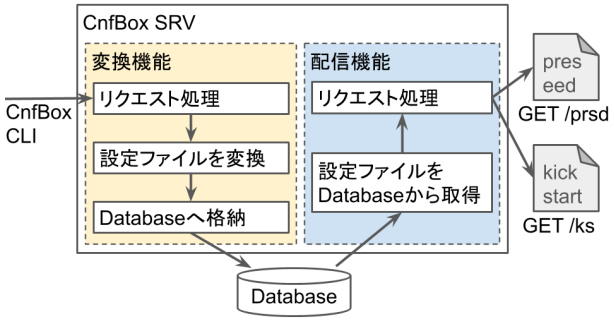

図 4 CnfBox SRV アーキテクチャ

変換機能

CnfBox SRV は CnfBox CLI から設定（例：ネットワーク，OS，認証情報）を受け取る．受け取ったデータは，データベースサーバで自動生成される ID とともにデータベースで保存する．また，プログラムにより解析してデータ構造へ変換する．これを事前にプログラム内に定義した各設定ファイルのフォーマットと個々に照らし合わせて変換を行う．

配信機能

生成した設定ファイルをデータベースから取得し，内蔵する HTTP サーバで配信する．このサーバでは RESTful API を使用する．

4.4 システムの動作フロー

インストールにおける動作フローを図 5 に示す．以下では，図 5 の手順ごとに説明を行う．

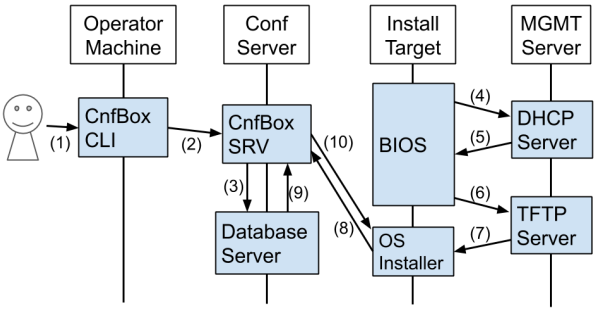

図 5 動作フロー

(1) オペレータは CnfBox CLI を利用してインストールのパラメータを投入する．
(2) CnfBox CLI は入力された設定を CnfBox SRV へ送信する．
(3) CnfBox SRV は受信した設定データから Kickstart, preseeed の設定フォーマットを生成し，Database Server へ保存する．
(4) Install Target の電源をいれ，ネットワークブートを選択．DHCP サーバへ IP 取得を試みる．
(5) DHCP Server は IP 取得のリクエストに応答し，IP アドレスを払い出す．
(6) インストール対象マシンは，DHCP Server から提示された TFTP Server へアクセスして OS インストーラを取得する．
(7) TFTP Server はインストール対象マシンからのリクエストに対応する OS インストーラを返す．
(8) OS インストーラは CnfBox SRV へ設定ファイルの取得リクエストを送る．
(9) CnfBox SRV は受信したリクエストに対応する設定ファイルを Database Server から取得する．
(10) CnfBox SRV は Database Server から取得した設定を，OS インストーラへ返す．
(11) OS Installer は取得した設定をもとに自動的にインストールを行う．

5. 検証

提案システムが実際のインストールで利用できるか検証を行った．

5.1 環境

研究室内部に設置された仮想化基盤上に仮想マシンによる実験環境を構築した．また，CnfBox SRV を Google Cloud Platform(以降，GCP とする) 上の Google App Engine と Google Cluod SQL を使い構築した．研究室環境と環境 (図 6) と GCP はインターネットを介してやり取りが可能である．実験では簡単のため，OS インストーラの起動と設定ファイルの指定を手動により行った．そのため，TFTP サーバを構築していない．

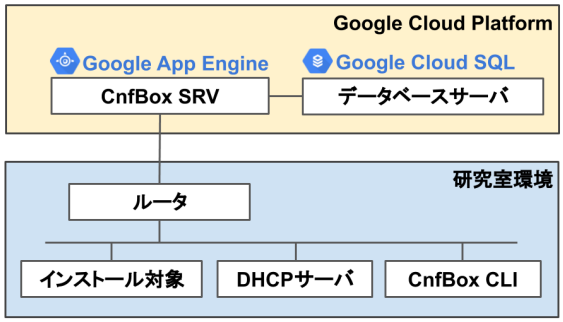

図 6 研究室環境と GCP 環境

CnfBox SRV

GCP 上の Google App Engine(以降，GAE とする) へプログラムを設置する．GAE のランタイムは Python 3.7 を使用した．ファイアウォールルールにより，大学のネットワーク以外からの HTTP および HTTPS によるアクセスを IP レンジに基づき遮断した．データベースサーバとの通信には Unix ドメインソケットを使用した．

データベースサーバ

GCP 上の Google Cloud SQL で MySQL 5.7 によるデータベースサーバ (vCPU: 1 コア, RAM: 614MB, HDD: 10GB) を構築した．CnfBox SRV と同様に大学以外からのアクセスを遮断した．GAE 実行ユーザに IAM ロール (Cloud SQL クライアント) を付与することで，データベースサーバへのアクセスを許可した．

CnfBox CLI

研究室環境に接続したラップトップ (CPU: Intel Core i7-8550U 1.8 GHz, RAM: 16GB, SSD: 512GB) で Python 3.7.6 の動作環境を構築して，CnfBox を実行する．

インストール対象

研究室環境の VMWare ESXi 上に VM(CPU: 1core, RAM: 2GB, SSD: 30GB) を作成した．OS はインストールされていない．

インストール OS

インストールする OS として Ubuntu 18.04 と CentOS 7 を使用した．それぞれ標準的なインストール自動化ツールには，preseed と Kickstart が用意されている．

DHCP サーバ

研究室環境の VMWare ESXi 上に VM(CPU: 1core, RAM: 2GB, SSD: 30GB) を作成し，Ubuntu 18.04 上に ISC DHCP をインストールした．

5.2 方法

次の 3 ステップを 2 種類の OS(CentOS 7, Ubuntu 18.04) により実験し，正常に自動インストールが行えるか検証した．

(1) **CnfBox を GitHub からダウンロード：** GitHub リポジトリからソースコードをダウンロードする．

(2) **GCP のセットアップ：** GCP でプロジェクトを作成し，Cloud SQL のインスタンスを前述のスペックで作成する．

(3) **CnfBox SRV を GCP にデプロイ：** GAE に CnfBox SRV をデプロイする．GAE 実行ユーザの IAM ロールに Cloud SQL クライアントを追加する．ファイアウォールポリシーで大学以外からのアクセスを遮断する．

(4) **CnfBox CLI による設定の投入：** Python 3.7.6 をインストールしたマシンで CnfBox CLI を実行する．

(5) **CnfBox SRV へのアップロード：** CnfBox CLI から commit() メソッドを実行し，設定を CnfBox SRV へアップロードする．

(6) **VM の設定と起動：** 研究室環境の VM(インストール対象) に Ubuntu インストーラ ISO(ubuntu-18.04-live-server-amd64.iso) を DVD ドライブへ設定し，VM を起動する．

(7) **起動パラメータの設定：** ブートローダの起動オプションをコード 3 に修正して OS を起動する．これにより preseed で参照する設定ファイル配信サーバを構築した CnfBox SRV にする．実験では https://cnfbox-dev.appspot.com/ に設置した．

コード 3 Ubuntu 起動オプション

```
append DEBCONF_DEBUG=5 auto=true locale=en_US.
    UTF-8 (略) interface=auto url=https://
    cnfbox-dev.appspot.com/conf/16/preseed vga
    =normal initrd=/install/initrd.gz quiet --
```

(8) **設定の配信：** OS インストーラは起動オプションに付与された設定の取得先である CnfBox SRV からの設定ファイルを取得する．

(9) **インストール状況の確認：** 自動インストールが正常に完了するか確認する．

(10) **パラメータを変えて実験：** 手順 (6) 以降を以下の変更を加えて再び実験する．

- **VM の設定と起動：** Ubuntu インストーラ ISO →

CentOS インストーラ ISO(CentOS-7-x86_64-DVD-1908.iso)

- **ブートローダの起動オプション：** Ubuntu 用コード 3 から CentOS 用コード 4 へ変更する．コード 4 では，Kickstart の設定ファイル取得先に構築した CnfBox SRV を指定している．

コード **4** CentOS 起動オプション

```
linux text ks=http://cnfbox-dev.appspot.com/
    conf/16/Kickstart
```

5.3 結果

Ubuntu, CentOS ともに正常に自動インストールが行われた．また，設定した認証情報でコンソールからログインが行えた．CnfBox SRV のアクセスログからも，正常に設定ファイルが取得できたことが確認できる（コード 5）．

コード **5** CnfBox SRV のアクセスログ

```
163.215.6.1 - - [13/Jan/2020:13:17:44 +0900] "
    GET /conf/16/preseed HTTP/1.1" 200 499 - "
    debian-installer" "cnfbox-dev.appspot.com"
     ms=2277 cpu_ms=1978 cpm_usd=5.5767e-8
    loading_request=1
163.215.6.1 - - [10/Jan/2020:17:42:27 +0900] "
    GET /conf/16/Kickstart HTTP/1.1" 200 604 -
     "urlgrabber/3.10.2" "cnfbox-dev.appspot.
    com" ms=79 cpu_ms=407 cpm_usd=6.7502e-8
    loading_request=0
```

6. 評価

検証を通じて CentOS と Ubuntu を対象とした自動インストールが確認された．これにより，記述する設定記法が 1 種類でありながら，異なる複数の OS の自動インストールを実現した．

設定記法：提案手法と既存手法との設定記法の形式を比較した．コード 6-8 はそれぞれの記法で表現したネットワーク設定例である．既存手法では独自の設定記法 (コード 6,7) を採用している．提案手法では設定記法 (コード 8) に JSON 形式を採用した実装をしている．JSON 形式をはじめとする標準化された記述形式の採用により，他のツールでの読み込みやプログラムからの取り扱いやすさが独自形式に比べ向上した．

設定 UI : 設定 UI は入力内容を既存技術と比較した．テキストエディタで設定ファイルを編集する場合に比べ，対話型の入力は設定値に注力できる．つまり，ファイルの構造や書き方に注力する必要がない．また，設定ファイルを編集する方法には無い補完によりユーザー体験が向上された．

コード **6** Lucie の設定記法

```
define host {
    host_name myhostname
    address 192.168.10.1
    mac_address 00:50:56:40:40:b6
    use hosttemplate
}
```

コード **7** LCFG の設定記法

```
dhclient.hostname myhostname
dhclient.mac 00:08:74:1A:52:7D
```

コード **8** CnfBox の設定記法

```
{
  "network": {
    "hostname": "myhostname",
    "host_ip_address": "192.168.10.1",
    "host_ip_address_type": "static"
  }
}
```

検証で使用した環境用の設定パラメータの記述量とファイル数の比較を表 1 に示す． なお，検証はプロトタイプを用いた為，実装済みの OS に関わるパラメータ (キーボード, タイムゾーン, 言語) のみを比較する．preseed と Kickstart はともに OS 標準のインストール自動化ツールである．作成する設定ファイル数および記述する形式数は，形式の統一化により 2 つから 1 つに削減された．これによりオペレータの設定ファイル作成の手間が削減された．記述するパラメータ数は $1 - 3/13 \approx 0.769$ より，従来に比べ約 77% の削減につながった．

表 1 CnfBox と既存技術の組み合わせとの比較

	CnfBox	Kickstart + preseed
作成する設定ファイル数	1	2
記述する形式数	1	2
記述するパラメータ数	3	13

7. 議論

本研究では，複数存在する設定フォーマットの差異を吸収すべく統一記法を提案した．当初は Ansible の採用した YAML 形式による記法を検討したが，ハッシュと配列の記述が入れ子になるにつれ，JSON に比べ分かりにくくなるため採用を見送った．統一記法を新たに導入することにより新たなツール依存が生じることは依然として課題である．LCFG の採用した OS インストール用環境として軽量

OS を使用する手法を利用する方が，既存の設定記法との整合性を考慮する必要がなく技術的制約が削減される．

本研究では，統一記法で記述された設定を既存ツールに対応する設定記法に変換した．これにより，既存ツールへの変更が不要となる．これは，変換する設定記法が少ない範囲においては有効である．しかし，記法が増えるにつれ変換サーバでの対応の手間が発生する．また，変換処理の追加時には変換サーバでのパラメータと変換後記法との対応付けが必要となる．これらを踏まえ，本提案で採用した設定記法の変換は，記法の差異を埋める汎用的で十分な手法とはいえない．この課題には，プログラムによる OS ごとの記法に基づく自動的な設定パラメータのマッピングでの解決を検討している．

本研究では，パスワードに代表される機密情報の管理が十分に検討されていない．IP レンジによるアクセス制限に加えて，暗号化した上で保存する方法の検討が必要である．これには Vault をはじめとした秘密分散管理データストアの採用を検討している．また，アクセス時の URL にトークンを付与することで真正性を確保することを検討いしている．

8. 終わりに

本研究では，新たな統一記法と対話型の設定 UI により，異なる OS の自動インストールの効率化を提案した．設定記法の違いによるオペレータの負担を統一記法の導入により削減した．また，設定の作成を容易にするため設定 UI を提案することで，設定の手間を削減した．

RESTful API とデータベースの採用により，これまで意識されなかった設定のバージョニングが実現された．変換サーバでは，設定 UI から受信した設定情報ごとに動的に設定を配信する URL を発行する．そのため，URL を変えることで過去のバージョンの設定情報へアクセスできる．Infrastructure as Code において，設定のバージョン管理は冪等性を担保するために重要である．RESTful API により実装したことで，副産物としてバージョン管理が得られた．今後は，設定のバージョン管理を支援する機能の実装や既存のバージョン管理の強化を検討していきたい．

参考文献

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and et al.: A View of Cloud Computing, *Commun. ACM*, Vol. 53, No. 4, p. 50–58 (online), DOI: 10.1145/1721654.1721672 (2010).
[2] Burgess, M. et al.: Cfengine: a site configuration engine, *in USENIX Computing systems, Vol*, Citeseer (1995).
[3] Anderson, P. and Scobie, A.: Large Scale Linux Configuration with LCFG, *Proceedings of the 4th Annual Linux Showcase & Conference - Volume 4*, ALS'00, Berkeley, CA, USA, USENIX Association, pp. 42–42 (2000).
[4] Anderson, P.: Towards a high-level machine configuration system, *In Proceedings of the 8th Large Installations Systems Administration (LISA) Conference* (1994).
[5] 服部健太，溝江真也，三角正樹：Kompira：シンプルで軽量な IT 運用自動化プラットフォーム，第 54 回プログラミング・シンポジウム予稿集，Vol. 2013, pp. 99–106 (2013).
[6] Finley, B. E.: VA Systemimager, *Proceedings of the 4th Annual Linux Showcase & Conference - Volume 4*, ALS'00, USA, USENIX Association, p. 11 (2000).
[7] 高宮安仁，真鍋　篤，松岡　聡：Lucie：大規模クラスタに適した高速セットアップ・管理ツール，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 44, No. SIG11(ACS3), pp. 79–88 (2003).
[8] Magherusan-Stanciu, C., Sebestyen-Pal, A., Cebuc, E., Sebestyen-Pal, G. and Dadarlat, V.: Grid System Installation, Management and Monitoring Application, *2011 10th International Symposium on Parallel and Distributed Computing*, pp. 25–32 (online), DOI: 10.1109/ISPDC.2011.14 (2011).
[9] 小久保良輔，松山和広，三鴨利彰，住元真司，平井浩一：超大規模 HPC システムのインストール高速化の提案，技術報告 18 (2018).