社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

更新やカスタマイズが可能なクラスタ設定のパッケージ化手法

高宮　安仁† 　栄　 純明† 　山形　育平† 　松岡　 聡†,††

† 東京工業大学　〒 152–8552 東京都目黒区大岡山 2–12–1
†† 国立情報学研究所　〒 101–8430 東京都千代田区一ツ橋 2–1–2
E-mail: †takamiya@matsulab.is.titech.ac.jp

あらまし　クラスタ用自動インストーラの有用性が広く知られる一方, 依然として一般のユーザに普及していない主因として, クラスタ設定のカスタマイズにはシステム内部に関する知識が要求される点やインストールするパッケージの選択が初心者にとって難しいことが考えられる. 近年提案された手法ではこうした問題をある程度解決しているものの, 詳細な設定が依然として難しい点や設定のパッケージ化には専門的知識を必要とする点などがある. そこで, 我々はメタパッケージと呼ばれる概念を導入し, クラスタ上である機能を実現するパッケージ群および設定ファイルをひとつのパッケージとしてパッケージ化し, インストール処理中でのファーストクラスエンティティとして扱う手法を提案する. また, メタパッケージと共に提供されるツール群によってインストール性能や記述力を損なうことなくメタパッケージを柔軟に, 依存関係を厳密に扱いつつ, エンドユーザによって容易にカスタマイズ可能にできることを示す. また, 我々の自動クラスタインストーラである Lucie 上にメタパッケージの機能を実装することによってその有用性を確認する. 実験では, 必要なメタパッケージを選択するだけで適当な依存性や衝突関係のチェックが行われ, インストール自体が 5-6 分といった従来ツールよりも短い時間で完了することを確認した. 設定内容の低レベルデバッグにおいても, メタパッケージのサポートツールを用いることに寄ってリスト中から必要なクラスタ機能を選択し, 起こりうる衝突関係の情報を取得しこれを解決することで, 初心者でもクラスタを 1 時間以内に構築できることを期待する.

キーワード　クラスタ, システム管理

A Flexible Configuration and Packaging Method for Cluster Installers

Yasuhito TAKAMIYA†, Yoshiaki SAKAE†, Ikuhei YAMAGATA†, and Satoshi MATSUOKA†,††

† Tokyo Institute of Technology　Oookayama 2–12–1, Meguro-ku, Tokyo, 152–8552 Japan
†† National Institute of Informatics　Hitotsubashi 2–1–1, Chiyoda-ku, Tokyo, 101–8430 Japan
E-mail: †takamiya@matsulab.is.titech.ac.jp

Abstract　Although automated cluster installers are becoming better known, it has not attained widespread popularity for several reasons, one of which is that customization of cluster configurations according to the needs of the underlying environment as well as configuring multiple user-level packages are quite difficult for the layman. Recently proposed solutions may relieve expertise at a certain level, but are incomplete that detailed customization and/or packaging will again require expert knowledge. Instead, we propose the notion of *metapackages* that treats a set of packages that define a certain functionality and their mutual configurations as a templatable package in itself, and treated as a first-class entity in the installation process. We show that, with associated tools support metapackages provide very high flexibility, rigorous dependency management, ease of end-user customizability, without sacrificing performance or expressive power in cluster configurations. We demonstrate the effectiveness by implementing the metapackage feature on top of our automated cluster installer Lucie. Experiences have shown that cluster installation itself will only take 5-6 minutes after a set of necessary metapackages have been selected, with appropriate dependency and conflict checks performed. Even with low-level debugging with our support we expect that a layman can pick the necessary features from a list, get full account of possible conflicts, and build a cluster in less than an hour by resolving such dependencies with alternate picks.

Key words　Clustering, System Administration

1. はじめに

OS やミドルウェア/ソフトウェアのインストールや設定を複数のノードに渡って自動的に行う自動インストーラは，クラスタシステムのようなほぼ設定内容が均質で多数のノードから成るシステムでは特に有効である．全ノードへのインストール作業を完全に自動化することができるため，対話的インストーラを用いて一台ごとに手作業でインストールする場合に比べ，セットアップ作業を短時間で確実に行うことが出来る．このように大きな利点のある自動インストーラであるが，以下の要因により熟練者以外にとっては効果的な運用が難しい．

- クラスタを機能させるためにインストールしなければならないソフトウェアパッケージの数が膨大であり, クラスタ利用者の要求に対して過不足の無い「正しい」パッケージ集合を作成することが難しい．
- インストールを成功させるためには，インストールするソフトウェアパッケージ間の依存関係を解決しなければならない. またインストール時に依存関係を満たせなかった場合にはインストーラのエラー停止とともに複雑なエラーメッセージが出力されるため, 修正が困難である．
- インストールされるパッケージの選択やソフトウェア毎の設定処理内容をインストール前にすべて洗い出し, あらかじめ設定スクリプトに記述しておく必要がある. これはソフトウェアやシステムに関する詳細な知識を必要とし, またインストール処理が対話的でないためデバッグに時間がかかる．
- 一般にクラスタはさまざまな機能を受け持つノード — 管理ノードや計算ノード, ストレージノードなど — の集合から構成されるが, こうしたさまざまなタイプのノードを構築するためには経験やノウハウが必要である．

こうした点を解決するために, インストールするソフトウェアの選択や設定内容をテンプレート化することによって必要な設定ノウハウやデバッグのコストを軽減し, グラフトラバーサルアルゴリズムによってパッケージ間の依存関係を解決するなどして, 自動インストーラの設定を容易に生成できるシステムがいくつか提案されている [3] [4]. しかし, これらのシステムではテンプレートのカスタマイズ性やテンプレート間に発生する競合関係が未解決であるなど, 依然としていくつかの問題がある．

そこで我々は, 自動インストーラの運用に関する労力を大幅に軽減するための一般的なフレームワークとして ”メタパッケージ” の概念を提案し, 我々の開発した自動クラスタインストーラである Lucie 上に実装することでその有用性を示した. メタパッケージを用いることによってユーザはそれぞれのクラスタやソフトウェアに特有な知識をほとんど要さずに, 容易に自動インストーラ設定を記述し, セットアップやデバッグ, 運用することが可能となる. メタパッケージの重要アイデアは, 設定テンプレート自体をインストーラ中でのファーストクラスオブジェクトであるパッケージとして扱い, このパッケージの操作によってインストーラ設定を構成できるようにした点である．

メタパッケージによって,（1）ソフトウェアパッケージと設定ファイルのテンプレート化のための標準化された方法の提供（2） エンドユーザが簡単にカスタマイズ可能なテンプレートカスタマイズ機構の提供 （3） パッケージ間依存情報管理データベース及び Lucie によって生成される動的な依存情報を用いたパッケージ間の厳密な依存関係の定義と管理 （4） 依存関係エラーの簡潔な表示や迅速な開発/デバッグサイクルを可能とするためのテスト手段やさまざまなサポートツールの提供, が可能となる．

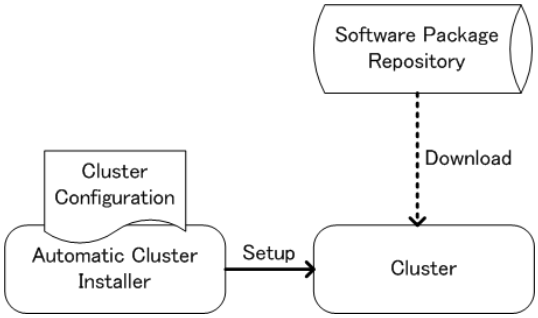

図 1　自動クラスタインストーラの一般的な方式

```
%packages
am-utils
dmalloc
gperf
j2sdk
rsh-server
sysstat
sysreport
...
```

図 2　パッケージ選択の設定例

2. 設定ファイルのテンプレート化および (メタ) パッケージ化

ここでは設定ファイルのテンプレート化手法について述べ, それらをファーストクラスのパッケージとして扱う方法について説明する．

3. 自動インストーラ

図 1 は自動クラスタインストーラの一般的な方式を表している. クラスタ管理者は自動クラスタインストーラへの入力としてそれぞれのクラスタの設定ファイルを記述する. 設定後インストーラが起動されそれぞれのクラスタノードで並列にインストールを実行する. インストールではディスクの初期化からパーティションの設定,OS やミドルウェア, アプリケーション等を含むさまざまなソフトウェアのインストールを行い, それぞれのソフトウェアに対応する設定 (e.g. /etc ディレクトリの設定) を行う．

3.1 インストール設定ファイルの例

自動インストーラの設定例として, 図 2, 図 3 に自動インストーラの一種である RedHat Kickstart [6] の設定ファイルの一部 を挙げる. ここからわかるように, 一般に設定ファイルはインストールするソフトウェアパッケージの選択部分 (図 2), およびインストール処理の一部として実行されさまざまな管理コマンドの実行などを行うポストインストールスクリプト (図 3) から成る．

この例はあくまで設定ファイルのごく一部の例である. 実際には小規模なクラスタでも数百から数千のソフトウェアパッケージをインストールする必要があり, それぞれのソフトウェアに

```
%post
exec >/dev/tty5
echo "RedHat 7.3(kickstart)" >/etc/motd

# configure start up daemons
chkconfig --level 0123456 bootparamd off
chkconfig --level 0123456 gated off
chkconfig --level 0123456 innd off
...
```

図 3　ポストインストールスクリプトの設定例

```
%metapackages
lmp-mpich-runtime # MPICH Runtime Env.
lmp-c-dev         # C Development Env.
lmp-java          # Java Env.
lmp-nis           # Account Management

%packages
lv
w3m
kterm
```

図 4　メタパッケージを用いたパッケージ選択の設定例

固有のポストインストールスクリプトが必要である. ユーザが自らポストインストールスクリプトを記述する負担を軽減するために, 出来合いのポストインストールスクリプトをソフトウェアごと提供する方法も考えられるが, 通常はクラスタごとに設定内容が異なるため現実的ではない.

インストーラ設定の作成には以下のような難しい点もある. パッケージ間に競合関係が発生した場合, ほとんどのインストーラは単純にエラーを表示して停止してしまう. エラー状況を把握できないユーザは ad-hoc に設定ファイルの修正を行うため, ますます保守が困難となる. ほとんどのポストインストールスクリプトはシェルスクリプトや低レベルのスクリプト言語を用いて記述する必要があり, また記述のためにはパッケージに含まれるソフトウェアの内部についての深い知識を要求するためにデバッグが難しい.

3.2　要件とアプローチ

3.2.1　パッケージ選択のパッケージ化

パッケージ選択では, それぞれがお互いに競合しないようなパッケージのリストを作成する必要がある. しかし先ほど述べたように, 選択するパッケージの数は数千程度になりうるので, 必要なパッケージを選択しつつインストーラが正しくインストールできるようにすることはユーザにとってむずかしい.

我々のアプローチでは, 機能や概念的に関連のあるパッケージ同士をグループ化し, これをインストール・設定するためのインストーラ設定一式を標準ソフトウェアパッケージであるかのようにひとつのパッケージとしてパッケージ化する. これを “メタパッケージ” と呼び, 他のソフトウェアパッケージと同様にパッケージ選択可能とする (図 4). メタパッケージの一例として具体的には,C コンパイラや C デバッガおよびさまざまな開発用ライブラリなど C 言語用のプログラム開発環境を構成するメタパッケージを ’c-devel’ と名づけ, 設定ファイル中でパッケージとして選択可能とする.

加えて, ひとつのメタパッケージを構成するそれぞれのコンポーネントはお互いの依存関係を満たすことをあらかじめ保証しておくことによって, 余分な依存問題の発生を防ぎ依存問題のチェックを簡単にする (外部メタパッケージとの依存関係チェック方法は後に説明する).

3.2.2　設定テンプレート

本システムでは, ユーザがポストインストールスクリプトを簡単に記述できるようにするために, あらかじめ十分に一般化・テンプレート化されたポストインストールスクリプト集を開発者側で提供し, これをエンドユーザが一貫した方法でカスタマイズし, 完全な設定ファイルを生成できるような仕組みを提供する. 具体的には, パッケージ開発者側でテンプレート化した

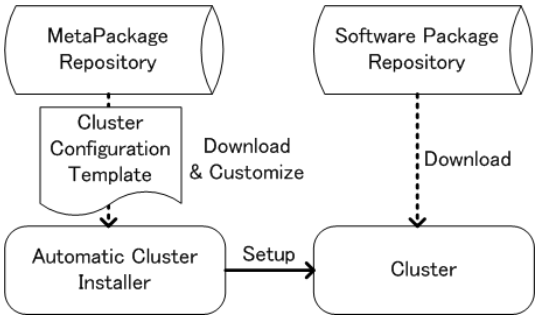

図 5　設定テンプレートを用いた自動インストーラの設定

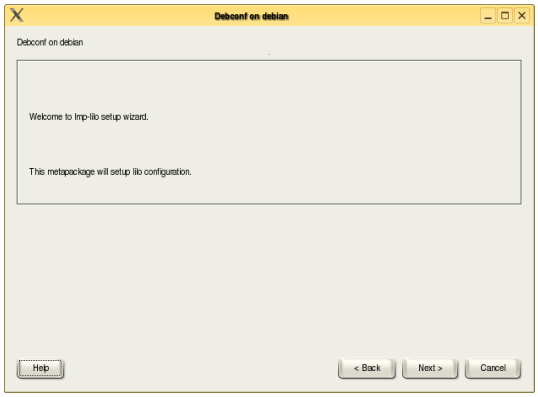

図 6　テンプレートのカスタマイズ用 GUI

典型的なスクリプトをメタパッケージ中に含め, 公開サーバ上にアップロードすることによって，スクリプトのネットワーク経由でのダウンロード・インストール・更新を可能とする．また，ダウンロードしたテンプレートをカスタマイズするための設定ダイアログ (図 6) を提供することによって，エンドユーザが簡単に標準化された方法でスクリプトをカスタマイズできるようにする (図 5)．この仕組みは,OSCAR [7] のように設定内容のカスタマイズ機構がインストーラの管理ツール自体に組み込まれているような場合とはまったく異なり, インストーラと独立して実現されている点に注目されたい.

3.2.3　メタインストール環境による設定内容のデバッグ

選択されたパッケージが実際に正しくインストールされるかどうかは, インストーラが実行されるノードにすでにインストールされているパッケージ集合や, パッケージサーバの状態に依存する. 一般的に, ノード上にすでに衝突するパッケージがインストールされており依存関係エラーが起こる場合にはインストーラはエラー停止する. また, パッケージサーバが要求されたパッケージを提供していない場合や, なんらかの要因でダウ

ンしている場合にもエラー停止する. 実際にエラーが起こると, 全インストール処理を繰り返し起動することによって設定内容をトライアルアンドエラー的にデバッグする必要がある. さらには, 起こりうるすべての設定集合にわたってデバッグを実施する必要がある.

この問題を解決するために我々のアプローチでは “メタインストール環境” と呼ばれる設定デバッグ環境を提供する. この環境ではインストール処理を高速に dry-run 実行することができる. これによって高速なデバッグ・テストサイクルを実行することができ, パッケージ間の依存関係チェックを効率的に行うことができる.

3.2.4 エラーメッセージのメタ集約

パッケージ間の依存関係に問題が生じた場合, インストーラは問題が生じたパッケージと周辺パッケージの依存関係を数レベルに渡って次のように表示する.

```
The following packages have unmet
  dependencies:
libmutexm-ruby: Depends: libruby (<=1.7)
 but 1.8.0-1 is to be installed
mhc-utils: Depends: ruby (<=1.7)
 but 1.8.0-1 is to be installed
racc-runtime: Depends: libruby (<=1.7)
 but 1.8.0-1 is to be installed
...
```

しかし, 多くのユーザにとってこのような冗長なメッセージは役に立たないだけでなく, デバッグを困難にする. そこで本システムではこの点を改善し, メタパッケージの依存関係をメタパッケージ間にあらかじめ設定しておくことによって, 依存問題を引き起こしたパッケージそれぞれがメタパッケージ中に含まれ, かつメタパッケージ間の依存関係に還元できる場合には, エラーメッセージをメタパッケージ同士の簡潔な依存関係エラーとして表示する.

```
The following packages have unmet
  dependencies:
lmp-ruby: Conflicts: lmp-c-dev but 0.2-1
is to be installed.
```

これによって依存問題を起こしているパッケージの見通しが良くなり, デバッグを容易にすることが期待できる.

3.2.5 メタパッケージ作成支援

メタパッケージに含まれるパッケージがお互いに衝突しないことを保証し, かつメタパッケージ間に適切な依存関係 (注2) を設定するのはメタパッケージ開発者の作業である. 膨大な数のパッケージ情報に精通していなくともメタパッケージを作成できるようにするためには, パッケージ間依存関係の解析を自動的に行う仕組みが必要である. さらに, さまざまな Linux ディストリビューション間の相違に関する知識や `make`,`rpm`, `dpkg` ツールなど, メタパッケージの生成に必要な知識を開発者から隠蔽する必要がある.

このために, メタパッケージを生成するための最低限の入力: (1) メタパッケージの提供するパッケージリスト (2) Lucie 設定ファイルのテンプレート (3) 設定ダイアログ定義 (後に述べる状態遷移表およびポストインストールスクリプトのテンプレート), のみを与えることによって, 自動的にパッケージ間依存情報を生成し, 潜在的に起こりうる依存関係問題を指摘するだけでなく自動的に適切なメタパッケージ依存情報を生成するためのツールを提供する. また, 生成されたこれらの依存情報を元にそれぞれのパッケージフォーマットに固有のファイルを生成し, 開発者の PGP 署名や MD5 情報を付加したメタパッケージを生成する.

(注2)：実際のパッケージ間依存関係はインストール時にチェックされる

4. メタパッケージの設計と実装

本節では, 我々の開発している自動インストーラである Lucie 上に実現したメタパッケージの設計と実装について述べる. ここでは Lucie を対象としているが, 理論的にはメタパッケージは Rocks などの類似システムにも適用可能である.

4.1 メタパッケージを用いたクラスタ設定例

まず, ユーザはあらかじめ利用可能なホストやネットワーク情報 (各ホストの NIC の MAC アドレスや, クラスタが設置されるネットワークの DNS サーバやゲートウェイアドレス情報など), また Lucie で構築したいクラスタのホスト構成など, 物理的リソースに関する情報等をリソースファイルと呼ばれる設定ファイルに登録する. 次にクラスタノードのソフトウェアリソースを設定するために, リソースファイルで定義したそれぞれのクラスタについてインストーラを構築しメタパッケージを用いてインストール設定を行う. 以下のコマンドで基本インストーラがインストールサーバ上に構築される:

```
sudo lucie-setup \
  --installer-name=presto_installer
```

次に, ユーザは設定に必要なメタパッケージ群を Lucie ホームページ [1], もしくはメタパッケージ公開サーバで検索し, 構築した基本インストーラに適用する.

```
sudo lucie-setup \
  --installer-name=presto_installer\
  --lmp-install='mpi-runtime,c-dev,monitoring'
```

この例では,MPI 実行環境 (`mpi-runtime`) と C 開発環境 (`c-dev`) をセットアップし, 各ノードをモニタリング対象とする (monitoring) ことを設定している. この際, それぞれのメタパッケージが依存する他のメタパッケージ (default, accounting, lilo メタパッケージ等) はメタパッケージに設定された依存関係から自動的に解析され, ダウンロード・インストールされる. また図 6 で示したような設定ダイアログ GUI がメタパッケージ毎に必要に応じて表示され, テンプレートのカスタマイズがユーザによって実行される. すべてのメタパッケージが設定された後, 各ノードが PXE ブートし Lucie によるインストール処理が自動的に完了する. インストールが完了後, 全ノードがリブートしクラスタ設定が完了する. これらの Lucie によるインストール機能は現在 RedHat や Fedora, Debian といったさまざまな Linux ディストリビューションへ対応している.

4.2 メタパッケージ間依存関係の生成

本システムでは, 単一の Linux ディストリビューションに特有なパッケージフォーマットだけでなく, 複数フォーマットのパッケージ情報を統合的に利用する. 実際にはすべてのパッケージフォーマットに必ず含まれている依存情報メタデータのうち Depends:, Conflicts:, Provides: と抽象化される情報を利用している.

Depends:は，そのソフトウェアパッケージが依存する他の

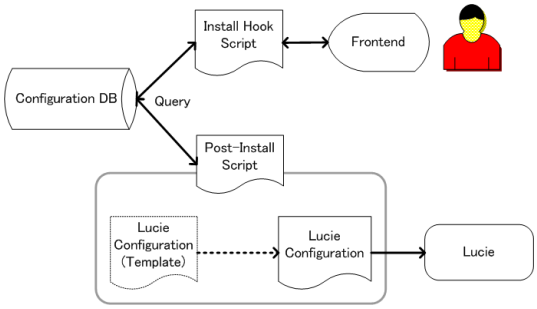

図 7　GUI によるテンプレートのカスタマイズ

パッケージ名と，そのバージョン番号を表したものである．Conflicts: は，そのソフトウェアパッケージが衝突する他のソフトウェアパッケージ名とバージョン番号を表したものである．Provides: は，そのソフトウェアパッケージが提供する機能を，仮想的なパッケージとして表したものである．以上の依存関係より，以下のようなパッケージの集合を定義する．Depends(p) はパッケージ p が直接，あるいは間接的に依存するすべてのパッケージの集合であり，依存関係を正向きにたどる関係である．Reverse-Depends(p) はパッケージ p に直接，あるいは間接的に依存するすべてのパッケージの集合であり，依存関係を逆向きにたどる関係である．Provided-Depends(p) はパッケージが提供するすべての仮想パッケージについて，Reverse-Depends をとったものの和集合である．以上より，メタパッケージを MPa，MPb，また通常のパッケージを pa，pb とすると，メタパッケージ間の依存関係を以下のように定める．

MP_a depends on MP_b:

$$\exists pa \in MP_a, depends(pa) \cap MP_b \neq \emptyset \quad (1)$$

or

$$\exists pb \in MP_b, provided - depends(pb) \cap MP_a \neq \emptyset \quad (2)$$

MP_a conflicts with MP_b:

$$\exists pa \in MP_a, conflicts(pa) \cap MP_b \neq \emptyset \quad (3)$$

以上の方法で求めたメタパッケージ自体の依存情報は，メタパッケージのビルド時に支援ツールによって自動的に検出され，メタデータ中に自動的に記録される．

4.3　テンプレートのカスタマイズ

本システムが提供する（メタ）パッケージのテンプレート化およびカスタマイズ機能は，汎用設定フレームワークである Debconf [8] を用いて構築されている．実際のテンプレートの導入とカスタマイズの流れは，図 7 のようになる．

ユーザがメタパッケージ公開サイトから Lucie インストーラ上にメタパッケージをダウンロード・インストールすると，メタパッケージに含まれるパッケージリストや，Lucie 設定ファイルの各テンプレートが Lucie インストーラ上に展開される．展開後，パッケージマネージャはメタパッケージ中に含まれインストール時のフックとして起動される，インストールフックスクリプトを実行する．インストールフックスクリプトは，設定ダイアログと通信し，カスタマイズに関する質問項目の表示要求と，ユーザ入力に応じた質問項目の状態遷移を設定ダイアログへ送信する．ユーザが設定ダイアログを通じて入力した値は，設定 DB に一旦保存される．すべての質問が終了後，パッケージマネージャは，メタパッケージ中に含まれ，パッケージ開発者によって記述されるポストインストールスクリプトを実行する．ポストインストールスクリプトは，設定 DB に保存されたユーザ入力項目の各値を読み込み，これと設定ファイルの各テンプレートをもとに，カスタマイズされた最終的な Lucie 設定ファイルを生成する．

```
question 'like_lucie' => \'
  { 'false=>' 'why_lucie_is_great} do |question|
  question.priority = Question::PRIORITY_MEDIUM
end

question 'why_lucie_is_great' do |question|
  question.priority = Question::PRIORITY_HIGH
end
```

図 8　Ruby による設定フロントエンド記述例

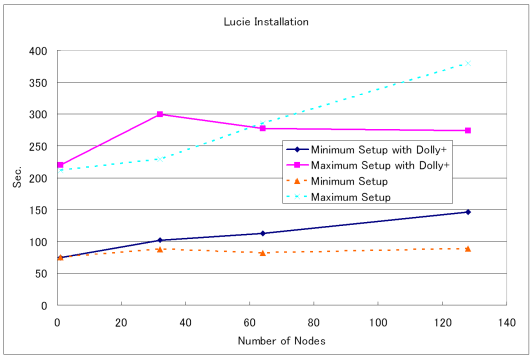

図 9　Lucie のインストール性能

4.4　GUI セットアップフロントエンド

GUI セットアップフロントエンドでは, オブジェクト指向スクリプト言語 Ruby による状態遷移定義ライブラリを提供する. 図 8 に見られるように, 画面状態の遷移をルールベースで記述することができるため, 状態マシンを簡潔に記述できる.

4.5　インストーラのメタ実行 (dry-run) 機能

メタ実行 (dry-run) 機能では, 設定ファイルに記述したソフトウェアパッケージがエラー無く完全にインストール可能か, もしくは競合を起こすかどうかをすばやく確認することができる. 我々の実装では,Lucie がインストーラの実行環境として作成する NFSroot 環境をこれに用いる.

この NFSroot 環境は物理的なネットワーク構成を除いてはソフトウェア構成や設定などが実際のクラスタノードと同等である．dry-run 機構では，この NFSroot 上へ `chroot` することで仮想的な実行環境として利用し，すべてのパッケージインストール処理を `--dry-run` オプション付で仮想実行する．我々の計測では，実際には 5 分程度を要するフルインストールの処理を 10 秒程度と短時間でシミュレートする. さらに，すべての起こりうるパッケージ間の衝突が適切に検出され, 第 3.2.3 節に述べたように報告される.

5.　評　　価

Lucie のメタパッケージを用いたセットアップでは, メタパッケージのインストールサーバへの展開後のインストーラ開始のための起動, および実際のインストール処理後の再起動と,2 度の（再）起動が行われる. 前者の処理は高々 10 秒程度で完了するため, メタパッケージによる設定が全体のインストールフェーズ (5 分程度) に与える影響は最小である.

図 9 はインストーラの第二フェーズ, すなわち Lucie を用いた実際のインストール時間が示されている. 計測では,RedHat7.3 の最小構成 (ハードディスク使用量 348MB, 基本ソフトウェアパッケージのみをインストール) および最大構成 (ハードディスク使用量 1303MB, すべてのソフトウェアパッケージをインストール) について, ノード数 1 台〜128 台のセットアップ時間を計測した. また, ソフトウェアパッケージの各ノードへのデータ転送には,Lucie と連携動作する高速データ転送ツールである Dolly+ [5] を使用した場合, しない場合について計測を行った. なお Dolly+ は大容量データのパイプライン転送やチャンク分割によるバースト的な転送手法および転送リンクの監視によって, 転送のフェイルオーバーやノード台数に対して O(1) での効率的なファイル配布を可能としている.

結果, 最小構成の場合には Dolly+ 使用, 未使用の場合についておよそ 150 秒以内でクラスタ全体のセットアップを完了していることがわかる. また, 最大構成においても Dolly+ を使用した場合にはパイプライン効果によって約 300 秒以内間でインストールを完了できていることがわかる. メタパッケージでの設定に要する時間を含めても数分程度である. より重要なのは, メタパッケージの選択・導入によってインストーラの設定が非常に簡単に行えるようになったことである. このため, インストールに不慣れな初心者であってもクラスタのインストール・設定に要する時間を従来の数日程度から 1 時間程度に短縮できることが期待できる.

5.1 関 連 研 究

SCore EIT [9] や OSCAR [7] などのツールは, クラスタシステムに必要な全てのソフトウェアをインストール対象とすることで設定の自由度やカスタマイズ性, ソフトウェアパッケージ単位での操作やアップグレードといった運用性を犠牲にした反面, 初心者でも容易にクラスタを構築できるようにしたツールである. こうしたシステムの欠点は, ノード構成にかかわらず画一的に大量のソフトウェアをインストールしてしまうためにインストール時間が長くなってしまう点や, ソフトウェアを後から追加するための手段がツールに統合されていない点である. 一方メタパッケージでは, 必要なパッケージ集合のみを選択し (追加) インストールすることができるため, 追加インストール処理を通常のインストールと同様にシームレスに行うことができ, またインストール時間も短くなる.

LCFG [10], NPACI Rocks [3], FAI [4] といったより先進的で柔軟な自動インストーラは, 設定ファイルを機能や概念別に断片 (スニペット) に分割し, スニペット間の継承/依存関係を定義することによって, 自動インストーラが依存関係を生成し自動インストールを実行する.

しかし,LCFG や FAI では設定テンプレートが生のシェルスクリプトや cfengine スクリプトで提供されているために, カスタマイズのためにはスクリプトやインストール処理の詳細をエンドユーザが理解する必要がある. また, テンプレートライブラリが現状では充実しているとは言えないため, 新しいソフトウェアパッケージのためにはユーザが自らスクリプトを追加する必要があるが, これが正しい設定かどうかはチェックされない. 一方, メタパッケージではすべてのカスタマイズ項目についてわかりやすいカスタマイズ用 GUI が提供される. また依存関係は常にチェックされ, たとえエラーが起こる場合でもメタ実行による迅速なデバッグサイクルによって問題点をすばやく検出することができる.

NPACI Rocks [2] では Rolls [3] と呼ばれる設定方式をサポートしている.Rolls は実際にはテンプレートライブラリの集合であり,Condor や PBS, Sun GRID Engine などのモジュールの依存関係と設定が Configuration Graph と呼ばれる形式でまとめられている. 依存関係グラフのトラバースを行うことで各 roll が適切に配置され RedHat Kickstart の設定ファイルへ変換される. この手法の問題点として, 単純な設定の依存関係については自動的に処理できるものの, ひとつのノードに複数サーバ機能を担当させる場合にはうまく動作しない. これは, 各トラバーサルの間に生じる可能性のある競合関係が検出されないためである.

6. お わ り に

自動インストーラの設定方式を改善し，エンドユーザが容易に自動インストーラ設定をカスタマイズできる仕組みとしてメタパッケージを提案した．他のクラスタインストーラでも同様の設定手段を提供しているものの, 設定テンプレートがインストーラシステム中でファーストクラスオブジェクトとして扱われていないためにメタパッケージと比較して機能が限定されている. 一方メタパッケージではテンプレート化された設定自体がインストール可能なパッケージとして提供されており, エンドユーザが作成する任意のクラスタ設定について依存関係のチェックや dry-run を行うことができ, 設定内容の正しさが保証される.

Lucie に関する情報やドキュメント, 論文中で触れたプログラム等はすべて `http://lucie.sf.net` からダウンロードすることができる.

謝辞 本研究の一部は，独立行政法人新エネルギー・産業技術開発機構 基盤技術研究促進事業 (民間基盤技術研究支援制度) の一環として委託を受け実施している「大規模・高信頼サーバの研究」の成果である.

文 献

[1] Yasuhito Takamiya. Lucie. `http://lucie.sf.net/`.

[2] Mason J. Katz Philip M. Papadopoulos and Greg Bruno. NPACI rocks: Tools and techniques for easily deploying manageable linux clusters. In *Concurrency and Computation: Practice and Experience Special Issue: Cluster 2001*.

[3] Federico D. Sacerdoti Greg Bruno, Mason J. Katz and Philip M. Papadopoulos. Rolls: Modifying a standard system installer to support user-customizable cluster frontend appliances. In *IEEE International Conference on Cluster Computing*.

[4] Thomas Lange. FAI (fully automatic installation) home page. `http://www.informatik.uni-koeln.de/fai/`.

[5] Dolly+ home page. `http://corvus.kek.jp/~manabe/pcf/dolly/index.htm`.

[6] Kickstart installations. `http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/custom-guide/ch-kickstart2.html`.

[7] Oscar — open source cluster application resources. `http://oscar.openclustergroup.org/`.

[8] Joey Hess. Debconf. `http://www.kitenet.net/programs/debconf/`.

[9] PC cluster consortium. `http://www.pccluster.org/`.

[10] Paul Anderson and Alastair Scobie. LCFG: The Next Generation. In *UKUUG Winter Conference*. UKUUG, 2002.