2004－HPC－99　(10)
2004／7／30

クラスタ設定のパッケージ化の設計と実装

高宮 安仁†　　松岡 聡‡

本研究は,自動インストーラが一般のユーザに普及しない主因が設定の困難さにあると考え,典型的なインストーラ設定のテンプレートをパッケージ化・ネットワーク配布し,これをエンドユーザがカスタマイズ用フロントエンドを通じてカスタマイズすることによって,ローカルサイトごとに用いる設定ファイルを自動生成することを試みた.また,ソフトウェアパッケージ管理システムのデータベース情報と自動インストーラの機能を利用することで,設定ファイル中に起こるインストール対象のソフトウェアパッケージ同士の衝突の高速な解決を試みた.このほか,テンプレートパッケージ開発者支援機構として,同データベースを用いてパッケージ化されたソフトウェア同士の依存関係の問題点を指摘する仕組みや,カスタマイズ用フロントエンド定義の自動生成などの,パッケージ作成支援機能を提供した.

Design and Implementation of Configuration Packaging Methods for Cluster Installers

YASUHITO TAKAMIYA†　SATOSHI MATSUOKA‡

While the wide spread of commodity clusters, fully automatic cluster installers haven't become the setup tool of choice for many users because of its complexity and difficulty in configuration. This paper introduces methods of 1) by packaging a set of typical configurations of cluster installers into common software package format (*MetaPackaging*) and make it downloadable for end-users with standard package managers, allows automatic generation of customized configurations for each local sites over integrated template customization GUI, and 2) by making use of package dependency information stored at underlying software package management system and pseudo installation environment built by automatic installer, allows sanity check of contentment of dependencies between user selected software packages without actual executions of installer. Moreover, for MetaPackage developers, we deploy a helper toolkit to enable detecting package dependency problems occurs when building metapackages and code generation of metapackage customization front-end.

1. はじめに

OS やソフトウェアの設定を自動的に行う自動インストーラは, クラスタリングシステムのようなほぼ同一のセットアップが施された多数のマシンから成るシステムでは特に有効である. 全マシン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で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. また, この性質をソフトウェア障害の復旧手段として利用し, システム全体を再インストールすることによって簡便に障害復旧を行う手法が試みられている[1][2].

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, このシステムはあまり一般的には利用されていない. この原因としては, (1) 利用可能なソフトウェアパッケージが多岐にわたるため, 実際にインストールするソフトウェアパッケージの選択が難しい点, (2) 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点, (3) ソフトウェア毎の設定内容をインストール前に把握し, あらかじめ設定しておく必要がある点, (4) クラスタノードとスタンドアロンのホストとでは設定内容が異なり, クラスタ特有のさまざまな設定ノウハウが必要である点, などが考えられる.

そこで本研究では, 自動インストーラの設定作業

† 東京工業大学大学院情報理工学研究科 数理・計算科学専攻

‡ 東京工業大学大学院学術国際情報センター / 国立情報学研究所

を大幅に簡略化する仕組みを提案し，そのアイデアの大部分を有用なプロトタイプシステムとして実装した．本システムは，クラスタシステムやソフトウェア固有の情報について特段の知識をもたない一般ユーザにも，自動インストーラの有効な運用を可能にするために，（1）典型的なソフトウェアパッケージの選択やインストーラ設定をユーザによるカスタマイズ可能なフォーマットでテンプレート化およびパッケージ化し，（2）テンプレートのエンドユーザによるカスタマイズ機構を自動インストーラ設定ツールに組み込むことで，事実上，エンドユーザが自動インストーラやソフトウェア固有の細かい知識を意識することなく運用できるように工夫した．また，ソフトウェアパッケージ管理システムのデータベース情報と自動インストーラ Lucie [2]の機能を利用することで，生成される設定ファイルの正当性の保障を可能とした．さらに，テンプレートパッケージ開発者向けのパッケージ開発支援ツールを実装した．

2. 設定ファイルのパッケージ化

本節では，自動インストーラの設定ファイルのテンプレート化やパッケージ化手法の概要を述べる．

2.1 設定ファイルの例

自動インストーラの設定ファイル例の一部を図 1，図 2に示す．この設定ファイルは，自動インストーラとして広く用いられている RedHat Kickstart[5] の設定ファイルである．設定ファイルは基本的に，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 1）と，インストーラの終了処理として実行され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部分（図 2）から成る．

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

図 1：パッケージ選択の設定例（一部）

```
%post
exec >/dev/tty5
echo "Kickstart-installed Red Hat Linux `/bin/date`" > /etc/motd
# configure start up daemons
chkconfig --level 0123456 bootparamd off 2>&1 > /dev/null
chkconfig --level 0123456 gated off 2>&1 > /dev/null
chkconfig --level 0123456 innd off 2>&1 > /dev/null
…
```

図 2：ポストインストールスクリプトの設定例（一部）

これら図に挙げた例は，あくまで設定ファイルのごく一部の例である．実際には，たとえ小規模なクラスタシステムでも，1 ノードあたり数百～数千のソフトウェアパッケージがインストールされ，それぞれについてポストインストール処理が必要であることから，数百～数千行もの設定ファイルを作成する必要があり，ユーザへの負担は極めて大きい．また，設定ファイルの作成には以下のような難しい点もある．選択したパッケージ中に衝突し合うパッケージがあった場合，インストーラはエラーで途中停止してしまう．また，ポストインストールスクリプト部の記述にはシェルスクリプトやソフトウェアごとの設定に関する詳しい知識を前提としており，記述ミスはインストーラの異常終了や思わぬ誤設定につながる．これらを考えると，人手に頼ってこれらのような設定ファイルを作成することが現実的でないことが分かるであろう．

2.2 要件とアプローチ

ここでは，多くのユーザにとって自動インストーラを効率よく運用できるための要件と，それを満たすための本研究のアプローチを述べる．

2.2.1 パッケージのグループ化

パッケージの選択にあたって，選択したパッケージに含まれる任意の 2 つのパッケージがお互いに衝突しないようなパッケージリストを作る必要がある．しかし，利用可能なソフトウェアパッケージの数は数千にのぼっており，またパッケージ間の依存関係が複雑に入り組んでいるため，システムにとって本当に必要なパッケージをもれなく選択し，かつそれぞれの依存関係を満足させることは難しい．

本研究では，パッケージを機能や概念別にいくつかのグループに分類し，それぞれをひとつのソフトウェアパッケージとしてインストール単位とする手法を提案する．本研究ではこれを，**メタパッケージ**と呼ぶ．具体的には，C コンパイラ，C デバッガ，開発系ライブラリパッケージなど，C 言語開発環境に必要なパッケージ一式を `c-devel’ メタパッケージなどと 1 つのメタパッケージにまとめ，これをパッケージ選択部分に指定可能とする．

```
# メタパッケージの選択
lmp-mpich-runtime # MPICH ランタイム環境一式
lmp-c-dev         # C 開発環境一式
lmp-java          # Java 環境一式
lmp-nis           # NIS アカウント管理一式

# 追加的な通常パッケージの選択
lv
w3m
kterm
```

図 3：メタパッケージを用いた場合のパッケージ選択例

また，1つのメタパッケージ中に含まれるパッケージは互いに依存関係を満足することをあらかじめ保障することで，余分な依存問題の発生を防ぎ，ソフトウェアパッケージ選択部分を簡潔に短く記述できるようにする（図 3）.

2.2.2 設定ファイルのカスタマイズ

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は，あらかじめ出来合いのポストインストールスクリプト集をベンダ側で提供し，これをエンドユーザがローカルサイトごとにカスタマイズできるような仕組みが好ましい.

本研究では，ベンダ側でテンプレート化した典型的なスクリプトをメタパッケージ中に含めることによって，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・更新を可能とした. 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 4）を提供することによって，エンドユーザが簡単にスクリプトをカスタマイズできるようにした.

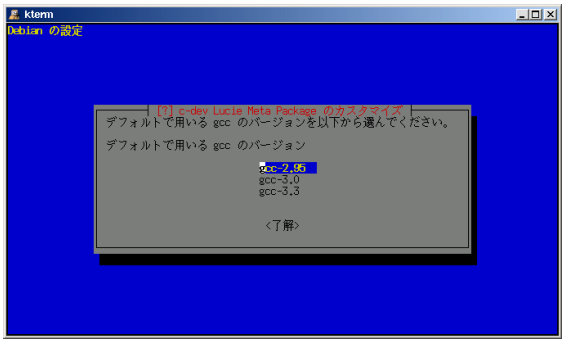

図 4：設定ダイアログのスクリーンショット（c-dev メタパッケージ）

2.2.3 選択したパッケージの正当性チェック

選択したパッケージが実際にインストール可能であるかどうかは，インストール先ホストのソフトウェア構成や，パッケージ取得元サーバの状態に依存する. 一般に，インストール先ホストに選択したパッケージと衝突するパッケージがすでにインストールされていた場合，依存関係が解決できないためインストーラはエラー終了する. また，パッケージ取得元サーバが選択したパッケージを提供していない場合，やはりインストールは失敗する.

このためパッケージ選択部分などの設定ファイルのデバッグには，設定ファイルを修正し，実機上でインストーラを実際に動作させるといったトライアンドエラー的な作業が繰り返し必要であり時間がかかる. 自動インストーラを普及するには，実際にインストーラを起動させることなく，インストール可能かどうかを短い時間でチェックするための仕組みが必要であるといえる.

本研究では，インストーラ実行前にあらかじめインストーラを実機とほぼ同等の擬似環境で dry-run する機構を提供しており，インストーラを実際に実行するのに比べ，極めて短い時間でインストール可能性をチェックすることができる.

2.2.4 メッセージ出力

インストーラはパッケージ間の依存関係に問題が生じたときに，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，数レベルに渡って表示する（図 5）. しかし，多くのユーザはこのようなメッセージに対応することが難しいであろう.

```
以下のパッケージには満たせない依存関係があります:
   libmutexm-ruby: 依存: libruby (< 1.7.0-0) しかし,1.8.0-1 は
   インストールされようとしています
   mhc-utils: 依存: ruby (< 1.7) しかし,1.8.0-1 はインストール
   されようとしています
   racc-runtime: 依存: libruby (< 1.7.0-0) しかし,1.8.0-1 は
   インストールされようとしています
                        ・・・
```

図 5：依存関係によるエラーメッセージの例（一部）

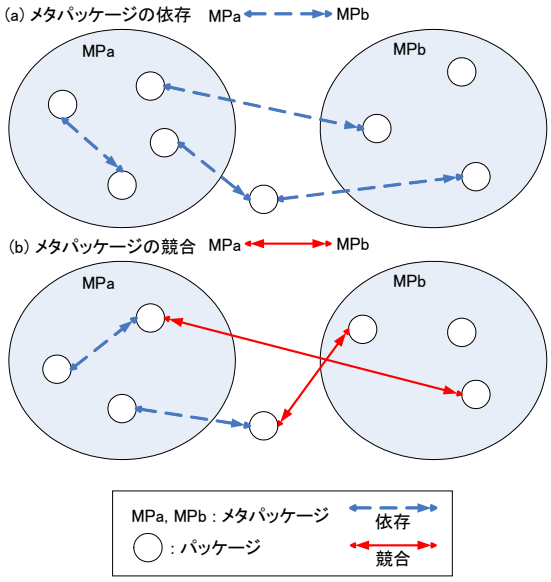

図 6：メタパッケージ間への依存関係の導入

そこで本システムではこの点を改善し，あらかじめ依存関係をメタパッケージ間に設定しておくことによって（図 6），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，メタパッケージ間の依存関係に還元できる場合には，エラー表示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（図 7）. この場合，競合するパッケージの数が，高々それらを含むメタパッケージ

の数だけに減少するため，表示されるエラーメッセージが単純化されるため，選択パッケージの修正が容易になる．

```
以下のパッケージには満たせない依存関係があります:
  lmp-ruby: 競合: lmp-c-dev しかし,0.2-1 はインストール
  されようとしています
```

図 7：メタパッケージによる改善された依存関係エラーメッセージの例

2.2.5 パッケージのネットワーク配布

セキュリティ勧告やメタパッケージのバージョンアップによって，メタパッケージは頻繁に更新される可能性がある．この場合，従来のようなCD-ROM メディア等による更新だと[1]，更新の遅れやし忘れがどうしても起こるため，セキュリティパッチが適切に適用されない可能性がある．

本システムでは，メタパッケージを通常のソフトウェアパッケージと同じフォーマット（rpm や deb）で配布し，またパッケージマネージャ apt (Advanced Package Tool) のネットワークインストール機能を用いることによって，メタパッケージの安全なネットワークダウンロードやインストール，自動更新を可能とする．

メタパッケージをネットワーク上で配布する場合，メタパッケージ開発者は作成したメタパッケージを scp 等を用いて http/ftp サーバ上に配置する．また，配置先の URL を *apt-line* と呼ばれる形式に従って Web ページ等でユーザに通知する（図 8）．

```
deb http://lucie.sourceforge.net/debian/lmp/woody/ ./
```

図 8：メタパッケージの取得元（apt-line）の例

ユーザは，必要なメタパッケージを含む apt-line を自動インストーラの設定に追加することによって，必要なメタパッケージをパッケージに含まれる PGP や MD5 情報を利用して安全にネットワークインストールできる．また，cron 等の設定によって，更新されたメタパッケージと依存するすべてのパッケージの自動更新が可能である．

2.2.6 メタパッケージ作成支援

メタパッケージを作成するにあたって，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障し，また作成したメタパッケージ間に適切な依存関係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依存情報に精通していなくともメタパッケージが作成できるようにするためには，依存関係の解析や設定を自動的に行う仕組みが必要であるといえる．また，メタパッケージの生成には make 等のコマンドに対する知識や，rpm や deb といったパッケージフォーマット固有の知識も必要である．メタパッケージの普及のためには，こうした煩雑な知識をメタパッケージ開発者から隠蔽する機構が必要といえる．

本研究では，（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（状態遷移表，ポストインストールスクリプト），といった最低限の情報のみを与えることによって，依存情報を自動的に解析し，依存関係に問題のあるパッケージの指摘やメタパッケージ自体の依存関係の適切な設定，メタパッケージのビルドに必要なパッケージフォーマット固有ファイルの自動生成，開発者情報 (PGP サイン) や MD5 情報の組み込みから，最終的なメタパッケージの生成までを自動的に行う，メタパッケージ作成支援ツールを提供する．

3. システムの設計

本節では，前節で述べた方針に基づいて作成したシステムの設計について説明する．

3.1 メタパッケージ間の依存関係

本システムでは，メタパッケージ間の依存関係を求めるにあたって，すべてのパッケージに必ず含まれているメタデータのうち *Depends:*，*Conflicts:*，*Provides:* という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，メタパッケージ間の依存関係を以下のように定める（図 6）．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，

MPa が MPb に依存する条件:

∃pa∈MPa,　depends(pa)∩MPb≠ φ,　or
∃pb∈MPb,　provided-depends(pb)∩MPa≠ φ

MPa が MPb と衝突する条件:
∃pa∈MPa,　conflicts(pa)∩MPb≠ φ

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，支援ツールによってメタデータに自動的に設定される．

3.2 テンプレートのカスタマイズ

本システムが提供する Lucie 設定ファイルのテンプレート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[6] 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 9 のようになる．

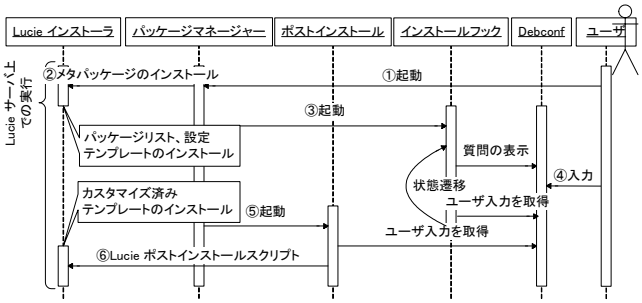

図 9：設定ダイアログによるカスタマイズの流れ

まず，ユーザがメタパッケージ公開サイトからパッケージマネージャを通じて 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 (①)，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 (②)．

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 (③)．インストールフックスクリプトは，設定ダイアログ (図 4) 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(④)した値は，Debconf 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 (図 10)を実行する (⑤)．ポストインストールスクリプトは，Debconf 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 (⑥)．

```
db_get c-dev/gcc
GCC_LINK=$RET
db_get c-dev/gpp
GPP_LINK=$RET
db_get c-dev/cpp
CPP_LINK=$RET

cat <<EOF > /etc/lucie/lmp/c-dev/scripts/symbolik_links
#!/bin/sh
chroot /tmp/target ln -sf /usr/bin/${GCC_LINK} /usr/bin/gcc
chroot /tmp/target ln -sf /usr/bin/${GPP_LINK} /usr/bin/g++
chroot /tmp/target ln -sf /usr/bin/${CPP_LINK} /usr/bin/cpp
EOF
```

図 10：ポストインストールスクリプトの例 (一部)

Lucie インストーラは，生成された各設定ファイルを入力として，クラスタノード上で実際のインストール処理を自動実行する (図 11)

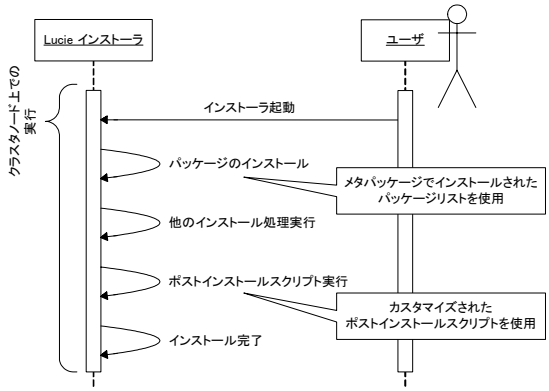

図 11：生成された設定ファイルを用いた Lucie インストーラの実行

3.3 設定フロントエンドの生成

設定ダイアログを制御するためのインストールフックスクリプトでは，パッケージごとに状態遷移マシンをシェルスクリプトで実装する必要があり，開発者の負担が大きかった．

本システムが提供する支援ツールでは，こうした状態遷移マシンを簡単に記述するための仕組みとして，ユーザが定義した XML による状態遷移表 (図 12) を入力として，オブジェクト指向言語 Ruby で記述された，デザインパターンの State パターンによる状態遷移マシン (図 13) を自動生成する仕組みを提供する．

3.4 Dry-run 機能

設定ファイルによって指定されたソフトウェアパッケージが実際にインストール可能であるかどうかをチェックするために，本システムでは自動インストーラ Lucie がインストーラサーバ上に生成する NFSroot 環境を利用する．

```
<statemap name="c-dev">
  <state name="c-dev/overview" priority="medium"
template="c-dev/overview" start="true"
next="package_information"/>
  <state name="c-dev/package_information" priority="medium"
template="c-dev/package_information" next="gcc"/>
  <state name="c-dev/gcc" priority="medium" template="c-dev/gcc">
    <case value="gcc-2.95" next="gpp"/>
    <case value="gcc-3.0"  next="gpp"/>
    <case value="gcc-3.3"  next="gpp"/>
  </state>
                              . . .
</statemap>
```

図 12：XML による状態遷移表の定義 （一部）

```
class PackageInformation < TextState
  def transit( aDebconfContext )
    super aDebconfContext
    aDebconfContext.current_state = DebconfContext::GCC_STATE
  end
end
class Gcc < SelectState
  def transit( aDebconfContext )
    super aDebconfContext
    aDebconfContext.current_state = ¥
    case get( @name )
    when 'gcc-2.95'
      DebconfContext::GPP_STATE
    when 'gcc-3.0'
      DebconfContext::GPP_STATE
    when 'gcc-3.3'
      DebconfContext::GPP_STATE
    end
  end
end
                              . . .
class DebconfContext
  PACKAGE_INFORMATION_STATE =
PackageInformation.new( 'c-dev/package_information', 'medium' )
  GCC_STATE = Gcc.new( 'c-dev/gcc', 'medium' )
                              . . .
  attr_writer :current_state
  def initialize
    @current_state = PACKAGE_INFORMATION_STATE
  end
  def transit
    @current_state.transit self
  end
end
# Main
title 'c-dev Lucie Meta Package のカスタマイズ'
debconf_context = DebconfContext.new
while true do
  rc = debconf_context.transit
  exit 0 if rc.nil?
end
```

図 13：本システムで自動生成された State パターンによる状態遷移マシンの例（一部）

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インストーラ実行環境と同等である．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，インストーラ中のパッケージマネージャによるインストール処理を `--dry-run' オプション付で仮想実行する．これによって，実際には 5 分程度を要するインストール処理を 10 秒程度と短時間でシミュレートし，依存問題が起きる場合には原因となるパッケージとその依存関係を表示する．

4. 関連研究

LCFG[7]や NPACI Rocks[1]，FAI[8]といった自動インストーラは，設定ファイルの断片（スニペット）とスニペット間の継承関係を提供することによって，ユーザによるインストーラ設定ファイルの生成や再利用を可能としている． メタパッケージと比較すると，スニペットの問題点として，（1）スニペット間の競合関係が考慮されていないために，生成される設定ファイルに矛盾が発生する場合がある点，（2）その操作には各インストーラ独自の専用ツールが必要である点，（3）テンプレート化されておらず，カスタマイズ性が低い点（4）PGP サインや MD5 チェックサムなどの改ざんされていないことを保障する仕組みが無い点，などがある．

5. おわりに

これまでほとんどのユーザにとって利用することができなかった自動インストーラの設定方式を改善し，ソフトウェア固有の細かい知識等を意識することなく運用できる仕組みとして，メタパッケージを実装した．また，メタパッケージ開発者用の支援ツールを開発した．今後の課題として，これらのツールを用いてどの程度設定コストが軽減されるかを実環境上で評価する必要がある．

謝 辞

本研究は，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．

参 考 文 献

[1] Philip M. Papadopoulos, Mason J. Katz, and Greg Bruno, “NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters”, Cluster 2001 , October 2001

[2] 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”, IPSJ Trans. ACS. 論文誌第 3 号, pp. 79-88 , August 2003

[3] SCore, http://www.pccluster.org/score/dist/index.php

[4] OSCAR Homepage, http://oscar.sourceforge.net/

[5] RedHat Kickstart, http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/custom-guide/ch-kickstart2.html

[6] Debconf, http://www.kitenet.net/programs/debconf/

[7] LCFG: A large scale UNIX configuration system , http://www.lcfg.org/

[8] FAI (Fully Automatic Installation) Home Page, http://www.informatik.uni-koeln.de/fai/