ユビキタス機器を対象とした プログラム配布およびデータバックアップ手法

荒木拓也† 小倉章嗣†

近年，パーソナルコンピュータだけではなく，情報家電，携帯電話，POS 端末等，高度な処理能力を持つユビキタス機器が多数登場しつつある．それにともない，利用しているプログラムの更新や新しいプログラムのインストール，さらには，ハードウェア障害時の機器交換等，これらの機器に対する「管理」の要求が顕在化しつつある．このような管理に関して，現在はシステム側で十分な対応がとられておらず，主に手動で行う必要があった．本研究では，このような問題を解決するため，プログラムの配布およびデータバックアップを自動化することで，管理を容易にするシステムを提案する．本システムでは，プログラムは実行開始時等に自動的に更新される．この際，他のプログラムの実行環境には影響を与えない．さらに，機器側の状態は自動的にサーバにバックアップされ，機器交換の際等はこれが自動的にダウンロードされる．また，これらの機能は差分を用いて通信，保存することで，効率良く実現している．我々はこのようなシステムの設計，試作，評価を行い，その効率の良い実装が可能であることを示した．

A Program Delivery and Data Backup Method for Ubiquitous Appliances

Takuya Araki† and Shoji Ogura†

Today, we are using various information devices that have large processing power, like home information appliances, mobile phones, POS terminals, as well as personal computers. This exposes the requirement of “administrating” these devices, like update of programs, installation of new application, and replacement of the device in the case of failure. However, there is no enough support for this requirement, and manual administration is required currently. In this paper, we propose a system that can administrate these devices easily by automating program delivery and data backup. In this system, programs can be updated automatically at the time of invocation. This does not affect the environments of other programs. In addition, state of the device can be backed up to the server automatically, which can be used at the time of device replacement. These functions are implemented efficiently using differences during communication and in storage. We designed, prototyped, and evaluated this system, and proved that efficient implementation of this system is possible.

1. はじめに

近年，パーソナルコンピュータだけではなく，ハードディスクレコーダ等の情報家電，携帯電話，コンビニエンスストアの POS 端末等，高度な処理能力を持つ情報機器が多数登場しつつある．これらの機器の数や種類は今後増え続け，また複雑さを増していくものと考えられる．

それにともない，これらの機器に対する「管理」の要求が顕在化しつつある．ここでいう管理とは，利用しているプログラムの更新や新しいプログラムのインストール，さらには，ハードウェア障害時の機器交換等である．

このような管理に関して，現在はシステム側で十分な対応がとられていない．たとえば，プログラムの更新に関しては，最も単純な場合，ユーザが手動で行う必要がある．自動的な更新機能がある場合でも，プログラムごとに作りこまれている場合が多い．

パーソナルコンピュータの場合は，システムが提供する自動更新機能が利用可能な場合がある（Linux における apt，yum 等）．しかし，これらの場合でも，整合性を保ちながら完全に自動的な更新を行うことは難しい．これは，複数のプログラムから共有されるライブラリを更新するような場合，ライブラリの古い版に依存するプログラムが存在すると，ライブラリの更新

† NEC サービスプラットフォーム研究所
NEC Service Platforms Research Laboratories

によって動作しなくなる可能性があるためである．

また，ハードウェア交換等の理由で，プログラムの実行環境を別ハードウェアに移行するためには，手動で機器内のデータをバックアップし，新しいハードウェアにコピーする等，大きな管理の手間がかかっていた．

以上のような問題への対応法の 1 つとして，シンクライアントや Web アプリケーションがあげられる．これらでは，サーバ側でプログラムを実行し，クライアント側ではその表示のみを行うプログラム，あるいは Web ブラウザを実行する．これにより，クライアント側の管理を不要にできるが，これらではプログラム実行時に常時ネットワークに接続していなければならないという問題がある．さらに，ネットワーク性能の限界により，実用的な速度で利用可能なアプリケーションが限られてしまう，大量の端末を管理するためには，プログラムを実行するサーバも大量に必要になるため，スケールしないという問題があった．

本研究では，このような問題を解決するため，プログラムの配布およびデータバックアップを自動化することで，管理を容易にするシステムを提案する．

本システムでは，Web アプリケーション等と同様，URL を指定するだけで，機器側の状態を意識せずプログラムを利用することが可能となる．この際，ローカルにあるファイルを利用するため，更新時以外はネットワークに接続されている必要はなく，サーバに負荷をかけない．これにより，サーバ側での処理コストが低減され，大規模，多数の機器の管理が可能となる．さらに，ファイル管理を「差分」を用いて行うことにより，大規模の機器管理を効率良く行うことができる．

本システムは，主にユビキタス機器の管理を目的としたものであるが，その技術は一般的なプログラム実行方式としてとらえることができる．本システムの提案する方式が，将来のより一般的なプログラム実行方式の端緒となる可能性がある．

本論文の構成は以下のとおりである．2 章で，本システムの設計と試作実装について述べる．3 章でその評価について説明する．4 章で関連研究について述べた後，5 章でまとめを行う．

2. プログラム配布およびデータバックアップシステム

本章と 3 章では，プログラム配布およびデータバックアップを行うための，サーバ側，および機器側におけるソフトウェア，ストレージを含めた全構成を「システム」と呼び，議論を行う．

2.1 システムの要件

本システムでは，ハードディスクレコーダ等の情報家電，携帯電話，コンビニエンスストアの POS 端末，Kiosk 端末等のユビキタス機器群を管理の対象とする．

これらの機器は近年高性能化が進んでいる．数百 MHz のプロセッサ，数十 MB のメモリ，および数百 MB〜数 GB の二次記憶を持ち，OS として Linux や Windows XP Embedded を採用する等，従来のパーソナルコンピュータに相当する能力を持つものが一般的になりつつある．この傾向は今後も続くものと考えられる．

もちろん，これにあてはまらないユビキタス機器も存在する．たとえば，携帯電話の下位機種では，高機能化より低価格化を求め，プロセッサ性能やメモリ量はより小さく，OS としてリアルタイム OS を採用するもの等がある．

本システムでは，ユビキタス機器の中でも，より高性能化が進み十分な処理能力を持つもの，特に Linux 等の UNIX 系 OS を，特別な制限なく動作可能な処理能力を持つ機器を対象とする．

以下に，本システムの要件を示す：

- 版管理機能

機器側でプログラムを実行する際には，自動的にプログラムが更新され，つねに最新版が実行できるようにしたい．また，古い版のライブラリに依存するプログラムと新しい版のライブラリに依存するプログラムを共存させたい．また，これら複数の版を効率良く保存しておけるようにしたい．

この要件が必要となる場合として，たとえばコンビニエンスストアの Kiosk 端末がある．端末のメニューから複数のサービスが起動でき，それぞれのサービスが，端末側で動作するプログラムとサーバ側で動作するプログラムとで構成されているとする．この場合，サービスの更新時には，サーバ側のプログラムだけでなく，端末側のプログラムも同時に更新する必要がある場合がある．

ここで，端末側の複数のプログラムがライブラリを共用するような場合を考える（たとえば，通信ライブラリ，GUI 用ライブラリ，音声合成用ライブラリ等）．更新した端末側のプログラムが，新しい版のライブラリに依存するよう記述されていた場合，単純に共用されているライブラリを更新してしまうと，古い版のライブラリに依存して記述されていた別のプログラムが動作しなくなってしまう．

ここで，プログラムごとに個別にライブラリをダウンロードすることも考えられるが，その場合は，

ライブラリの動作に必要なファイルを個別にダウンロードすることになり，特に画像や音声データ等のファイルサイズが大きい場合は非効率になりうる．

- 配布管理機能

多数の機器に対して新しい版を配布できるよう，スケーラビリティに配慮した構成である必要がある．また，プロキシサーバ経由や他の機器を経由したダウンロード，サーバにおける版更新時のプッシュ配信等，柔軟な配布を可能にしたい．

たとえば，コンビニエンスストアは各チェーンごとに全国に約 1 万店規模で存在する．情報家電や携帯電話の場合は，機種にもよるがこれ以上の数が存在すると考えられる．したがって，これら多数の機器に対応する必要がある．

また，配信経路として，プロキシの利用やプッシュ配信はいずれの機器の場合も有用であり，実現することが望ましい．

- 機器状態管理機能

機器を交換する際や，別機器でプログラム実行の続きを行う場合に備え，機器の状態をサーバにバックアップできる必要がある．また，これは効率良く行えなければならない．

たとえば，携帯電話の内部にある，電話番号帳や受信済みメール，写真等のユーザ情報を故障に備えてバックアップすることを考える．この際，携帯網を使うことを想定すると，できるだけ通信量を減らした形でバックアップする機能が必要になる．

以上の機能を，低い開発コストで利用できるようにする必要がある．

このため，対象とする言語を Java 等に限定せず，既存のプログラムを再利用できるようにしたい．さらに，単純なインタフェースで機能を利用できるようにする必要がある．

性能に関する具体的な要件については，3 章で議論する．

また，動作環境としては，組み込み系 OS として普及が加速している Linux（あるいは同等の UNIX 系環境）を，対象として考える．

2.2 設計方針

以上の要件をすべて満たすことは，一見困難に思えるかもしれないが，以下の設計方針に従うことで，実現可能であることを示す．

2.2.1 UnionFS を用いた差分管理

版管理機能において，古い版と新しい版を効率良く保存する必要がある．また，配布管理機能においても，スケーラビリティに配慮する必要あるため，更新時の通信負荷を減らす必要がある．

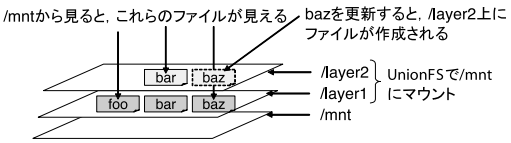

図 1 UnionFS
Fig. 1 UnionFS.

このため，新しい版は古い版およびその差分から構成することとし，そのために UnionFS [1),2)] と呼ばれるファイルシステムを利用することとした．

UnionFS とは，Linux 等の OS で動作するファイルシステムで，複数のディレクトリを重ね，1 つのディレクトリとして見ることができるものである．この様子を**図 1** に示す．

図に示すように，UnionFS では，あるマウントポイント（ここでは/mnt）に複数の他のディレクトリ（ここでは/layer1 と/layer2）をマウントし，これら 2 つのディレクトリに存在するファイルが/mnt に存在するように見せることができる．ここで，この 2 つのディレクトリには上下関係があり，同じファイルが両ディレクトリに存在するときは，上位のディレクトリが優先される．ここでは bar というファイルが両ディレクトリに存在するが，/mnt から見えるのは，より上位にマウントされた/layer2 ディレクトリのものである．さらに，このディレクトリ上でファイルの更新を行うと，下位のディレクトリに存在するファイルであっても，上位のディレクトリにファイルをコピーするようにできる．すなわち，書き込みを行う際でも，下位のディレクトリを読み込み専用とすることができる．

この機能を用いれば，古い版から新しい版に更新する際は，旧版との差分となるファイル群のみを取得し，上位ディレクトリとしてマウントすればよい．これにより，古い版を保存しながら，新しい版を構成できる．この際，余分に必要となる記憶領域は差分となるファイル群のみとなり，効率良く実現できる．

さらに，機器状態管理機能において，機器側のファイルをバックアップする際にもこの機能は利用できる．すなわち，バックアップ後に空のディレクトリを上位にマウントすることで，以後更新されたファイルはそのディレクトリ中に保存されるようになる．したがって，次回のバックアップ時は最上位のディレクトリの内容をバックアップすれば，前回との差分をバックアップできることになる．

2.2.2 プログラムごとの実行環境の分離

次に，プログラムの更新が他のプログラムに影響し

ないようにするため，プログラムごとにその「実行環境」を分離する必要がある．

そのため，UnionFS を用い，プログラム実行に必要なライブラリ等を，あるマウントポイントにすべてマウントし，プログラムはそのディレクトリに chroot して実行するようにした☆．

chroot により，そのプログラムにとってのルートディレクトリは当該マウントポイントとなるため，他のプログラムが利用するディレクトリとは分離される．これにより，他のプログラムの実行環境との分離を実現する．

ここで，プログラムを実行するマウントポイントや，そこにマウントされるディレクトリ群の情報等を，今後「実行環境」という概念として扱う．これについては後述する．

2.2.3 URL によるパッケージと実行環境の指定

次に，機器側でプログラムを実行する際にプログラムを自動更新するという要件を満たすことを考える．また，単純なインタフェースを提供するという要件も同時に満たすため，以下のように整理した．

まず，プログラムを構成するファイル群を「パッケージ」と呼び，その一意な ID を URL とした．たとえば，“http://pkgserver.name.co.jp/pkg_name” をパッケージの ID とし，この URL を指定することで，パッケージをダウンロード可能とする．ここで，同じ内容のパッケージが別の URL に存在しても，それは別のパッケージとして扱われる．

さらに，前述した「実行環境」に対しても，一意な ID として URL を与える．たとえば，“http://envserver.name.co.jp/env1” 等とする．ユーザはプログラム実行の前に，この URL に対応する「実行環境」の作成をシステムに依頼する．この際，その実行環境で利用するパッケージ群を指定する．

ここで，URL に指定されたサーバは，要求された実行環境に関する情報の登録を行う．後で機器が持つファイルをバックアップする際には，このサーバに対して行うことになる．

プログラムを実行する際には，実行環境の URL，および実行するプログラムのパス（たとえば “/bin/prog-name”）を指定する．システムは，その実行環境で利用するパッケージについて，更新を確認する．更新されていた場合はダウンロードし，パッケージを当該実行環境にインストールする．

☆ 本論文の試作では chroot を用いているが，他の技術の利用（jail, OpenVZ, VServer 等）も考えられる．

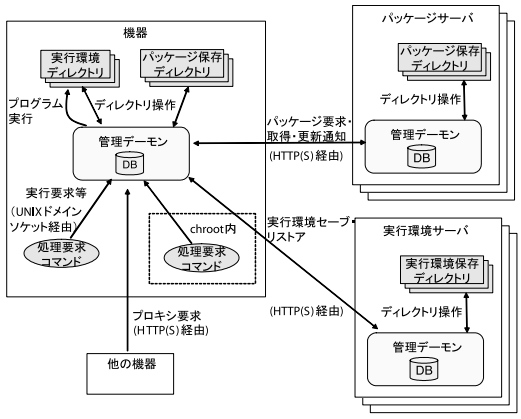

図 2 システムの構成図
Fig. 2 System structure.

以上のようにすることで，プログラムの自動更新および，単純なインタフェースによって実現するという要件を満たすことができる．

2.3 設計と実装

以上の設計方針に基づき，システムの詳細な設計および実装を行った．以下に，その詳細を示す．

2.3.1 システムの構成

まず，システムの構成図を**図 2** に示す．

本システムは，管理対象の機器と，パッケージサーバ，実行環境サーバから構成される．異なるパッケージを用いる可能性があるため，パッケージサーバは複数存在しうる．また，実行環境サーバに関しても同様である．

機器側では，プログラムの実行や，バックアップといった機能を利用する際,「処理要求コマンド」を利用する．この「処理要求コマンド」は,「管理デーモン」と通信を行い，要求を管理デーモンに伝える．実際の処理は管理デーモンが行う☆☆．

管理デーモンは，要求に応じて，実行環境に関するディレクトリとパッケージを保存するディレクトリの操作を行う（その具体的内容は，2.3.2 項，2.3.3.2 項で述べる）．また，要求が「ある実行環境でのプログラムの実行」であった場合は，管理デーモンは，chroot によって対象の実行環境ディレクトリをルートディレクトリとするプロセスを fork する．このプロセスが

☆☆ このような形で実現した理由は，以下のとおりである.「別の実行環境でのプログラムの実行」といった機能をある実行環境内で動作中のプログラムから利用する場合，その「別の実行環境」を構成するためのディレクトリには，現在実行中の実行環境からはアクセスできない．これは実行環境が chroot されたディレクトリとして構成されているためである．したがって，要求を chroot されたディレクトリ外で実行している管理デーモンに伝え，実行を行ってもらう．

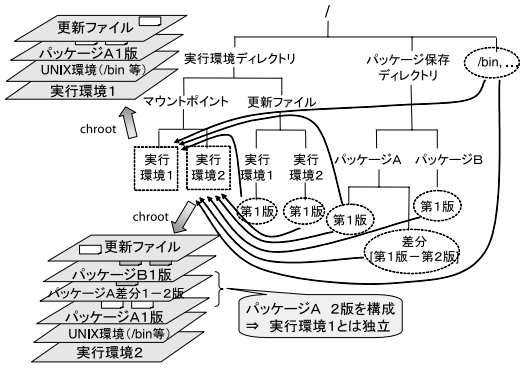

図 3 ディレクトリ構成例
Fig. 3 An example of directory structure.

要求されたプログラムを実行する．

パッケージを要求する，あるいはその更新通知を受け取る際，および実行環境をセーブ，リストアする際には，機器側の管理デーモンが，パッケージサーバ，あるいは実行環境サーバに対して HTTP(S) プロトコルで通信を行う．パッケージサーバ，実行環境サーバ内でも，機器側と同様にパッケージ保存ディレクトリ，および実行環境保存ディレクトリが管理されている．

2.3.2 ディレクトリの構成

それぞれの動作について説明する前に，UnionFS を用いたディレクトリの管理構造について説明する．

ディレクトリの構成例を**図 3** に示す．この例では，実行環境 1 ではパッケージ A の第 1 版を利用し，実効環境 2 ではパッケージ A の第 2 版とパッケージ B の第 1 版を利用している．

まず，実行環境 1 の方を用いて説明する．実行環境は，機器側の管理デーモンが，「/実行環境ディレクトリ/マウントポイント/実行環境 1」という空ディレクトリに，UnionFS を用いて必要なディレクトリをマウントすることで構成される．

マウントするディレクトリは以下のようになる．機器側の管理デーモンは，まず/bin，/lib，/etc 等の UNIX 環境を構成するディレクトリをマウントする．現在の実装では/bin 等をそのまま利用しているが，別のディレクトリにファイルを用意してもよい．

機器側の管理デーモンは，次に，利用するパッケージ A をマウントする．パッケージ A は，「/パッケージ保存ディレクトリ/パッケージ A」に保存されている．その下のディレクトリには，最初の版として「第 1 版」，および第 1 版と第 2 版の差分を含むディレクトリが存在する．ここでは第 1 版を利用するため，このディレクトリをマウントする．

機器側の管理デーモンは，最後に，プログラムのインストール・実行により更新されるファイルを保存するディレクトリとして，「/実行環境ディレクトリ/更新ファイル/実行環境 1/ 第 1 版」をマウントする．これにより，すべてのファイル更新はこのディレクトリに保存されるため，バックアップ時にはこのディレクトリのみバックアップすればよい．バックアップ後は新しいディレクトリを作成し，さらにその上に重ねることで，前回のバックアップ後に更新したファイルだけ，バックアップすることが可能になる．

次に，実行環境 2 の方を説明する．こちらはパッケージ A の第 2 版およびパッケージ B の第 1 版を利用している．これは，たとえばパッケージ B がパッケージ A の第 2 版が提供するライブラリを必要とする場合に相当する．

この場合，機器側の管理デーモンが，パッケージ A の第 1 版のディレクトリをマウントした後に，第 1 版と第 2 版の差分を構成するディレクトリをマウントすることで，パッケージ A の第 2 版を実行環境 1 とは独立に構成できる．

このようにすることで，複数の実行環境間でパッケージを構成するファイルを共有しながら，複数の版を独立に利用することが可能になる．

2.3.3 システムの動作

次に，システムの動作を順を追って説明する．

2.3.3.1 実行環境の作成

まず，ユーザ（必ずしもエンドユーザではなく，通常はプログラム）は，実行環境サーバに「実行環境」の作成を依頼する．これには，以下のようなコマンドを実行する：

```
prov create-env -name http://envsrv.name/env1
-package http://pkgsrv.name/pkg1
```

ユーザは “prov” というコマンドを経由して，管理デーモンに要求を行う☆．ここでは，“create-env” というサブコマンドで実行環境作成依頼を示している．オプションとして “-name” に実行環境名，“-package” にパッケージ名（複数可）を渡す．実行環境名，パッケージ名はそれぞれ URL 形式である．実行環境名のサーバ部分に localhost を用いた場合は，実行環境サーバを利用しない．

この要求を受け取った管理デーモンは，実行環境サーバ名が localhost でなければ，指定されたサーバに対し，実行環境作成要求を送る．

要求された実行環境名が利用可能な場合，実行環境サーバは要求内容をデータベースに保存し，成功を

☆ コマンド名は「ソフトウェアプロビジョニング」からとった．

返す．

機器側の管理デーモンは，利用パッケージ等の実行環境に関するメタデータをデータベースに保存する．

2.3.3.2 プログラムの実行

次に，プログラムの実行を説明する．プログラムを実行環境内で実行するには，以下のようにする：

```
prov execute -name http://envsrv.name/env1
"/path/to/command options..."
```

本項では，実行環境のリストアが不要な場合について説明する．

この要求を受け取ると，管理デーモンは，まず対象とする実行環境で利用するパッケージ群をデータベースから検索する．そして，そのパッケージ群がシステムにすでにダウンロードされているかどうかを確認する．

また，すでにダウンロードされている場合は，利用パッケージ群の最新版番号を当該 URL からサーバに問い合わせ，自機器の版が最新かどうかを判断する．

もしダウンロードされていなかった場合は，すべてのファイルを含む最新版のパッケージをダウンロードする．自機器が持つ版よりも新しい版のパッケージがサーバに存在する場合は，その差分をダウンロードする．これらは，tar + gzip で 1 ファイルにまとめて送られる．

要求を受けたパッケージサーバの側では，要求に応じたファイルを作成して返送する．ここで，パッケージサーバ側でも，図 3 に示したパッケージ保存ディレクトリと同様の構成でパッケージが管理されている．要求に応じたファイルを返送するためには，テンポラリのマウントポイントに，必要なディレクトリ（最新版（第 m 版）なら第 1 版のディレクトリおよび，第 1 版と第 2 版の差分 ···，第 m − 1 版と第 m 版の差分，第 n 版と第 m 版との差分であれば，同様に第 n 版から第 m 版までの差分のすべて）をマウントし，その内容を tar + gzip でまとめる．

このまとめられたファイル群は，第 n 版と第 m 版の差分（初回のダウンロードの場合は，第 m 版全体）を構成する．したがって，その間の版を構成するファイルは送られない．これは効率化のためである☆．

このように，パッケージサーバは容易に要求されたファイルを作成できる．

ファイルを受け取った機器側では，受け取ったファイルをパッケージ保存ディレクトリに展開し，データベースに現在保持する版の情報として記録する．

ここで，ダウンロードしたパッケージが持つメタ情報として，そのパッケージが依存する他のパッケージ名がある（これは版ごとに変わる可能性がある）．新しい版をダウンロードした際は，依存するパッケージを確認し，再帰的に同様の処理を行ってパッケージをダウンロードする．上記メタ情報には，依存するパッケージの版も指定することができる（第 n 版から第 m 版まで，つねに最新版を優先，等）．

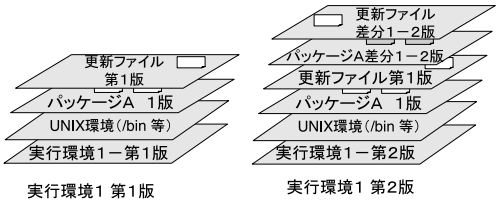

図 4 パッケージ更新時のディレクトリマウント例
Fig. 4 An example of directory mount when a package is updated.

次に，実行環境の構成を行う．

まず，機器側の管理デーモンは，実行環境ディレクトリに，マウントポイント，および更新ファイルを保存するディレクトリを新たに作成する．そして，UNIX 環境から順番に，必要なパッケージ群等を当該マウントポイントにマウントし，最後に新たに作成した更新ファイルを保存するディレクトリをマウントする．

対象とする実行環境がすでに存在していた場合でも，パッケージを新しい版に更新した場合は，新規作成時と同様に，新しいマウントポイントと更新ファイルを保存するディレクトリを別に作成する．

この様子の例を**図 4** に示す．

このような形でマウントすることで，以前の版に影響を与えることなく，新しい版を作成することができる．

さらに，パッケージによっては機器の構成に応じた設定等を行う必要があるかもしれないため，機器側の管理デーモンは，パッケージに添付されるインストールスクリプトを実行する．

以上で，新しい版を構成することができた．最後に機器側の管理デーモンは，要求されたコマンドを実行するプロセスを fork する．このプロセスは，対象の

☆ このため，別の実行環境が，後から当該パッケージの（送られなかった）途中の版を要求した場合，その版を再度ダウンロードする必要がある．また，後述するプロキシ機能を利用する場合にも同様に，要求された版よりより新しい版を持っていても，必要な版差分が構成できなくなる場合がある．
また，UnionFS ではファイルの削除を，“.” で始まるプレフィックスを持つファイルを上位の層のディレクトリ上に作成することで実現している．このような形で差分を送ると，“.” で始まるファイルを送ることができないので，更新によって削除されたファイルを伝えることができない．この部分の改善は今後の課題である．

マウントポイントに chroot し，要求されたコマンドを実行する．

以上が，プログラム実行のフローである．以上見たとおり，ユーザは実行を行う実行環境および，実行したいプログラムのパスを指定するだけで，自動的にパッケージの初期ダウンロード，更新，インストールが行われ，プログラムが実行される．

さらに，機器側の管理デーモンが，パッケージを新たにダウンロードする際に，図 4 に示した構成をとったことで，新版をインストールした後でも，安全に旧版を実行することが可能になる．

また，オプションで，サーバへの最新版の問合せを抑制することもできる．これにより，ネットワーク接続が存在しない際等でもプログラムを実行できる．

2.3.3.3 パッケージの更新通知

パッケージがサーバ側で更新される際には，更新通知を受け取ることができる．更新通知を受け取るには，以下のようなコマンドを実行する：

```
prov subscribe -package http://pkgsrv.name/pkg1
-env http://envsrv.name/env1 -script
"/path/to/callback"
```

このコマンドは，「指定されたパッケージが更新された際，通知を受け取り，指定されたコールバックスクリプトを実行する」ということを意味する．実行するコールバックは，実行環境ごとに指定できる．

コールバックスクリプトの中では，パッケージのダウンロードや，新版のインストールを行うことが考えられる．ここで，ダウンロードやインストールは，“prov download”，“prov install” コマンドが用意されており，個別に実行可能である．これにより，サーバからプッシュで更新を配信することができる．

あるいは，「新しい版がインストールされました」という表示をユーザに行い，新しい版の実行をうながすことも考えられる．

このコマンドを受け取った管理デーモンは，該当するパッケージサーバに更新時の通知を依頼する．

パッケージサーバ側では，依頼のあった対象パッケージ，実行環境を記録しておき，パッケージ更新時にはその情報とともに通知を送る．通知は，機器側に対し，HTTP でアクセスすることで行われる．通知先の機器の IP アドレスは，対象とする実行環境の，実行環境サーバに問い合わせることで取得する（localhost の場合は要求時の IP アドレスを記録しておき，それを利用する）．これについては後述する．

2.3.3.4 プロキシ経由のダウンロード

プロキシを利用して，パッケージをダウンロードするには，prov execute コマンドに “-proxy proxy.name” オプションを付ける．

この指定が行われると，proxy.name に指定されたサーバをプロキシサーバとして，新版の存在の確認やダウンロードに用いる．実現には，HTTP レベルのプロキシを利用している．

ここで，機器間でのダウンロードを行うために，機器側にプロキシ機能を持たせる．すなわち，機器側の管理デーモンは，パッケージサーバの管理デーモンと同等の機能を持つ．

プロキシとして要求を受け取った機器は，パッケージサーバと同様，ローカルに持つパッケージ保存ディレクトリから，対応する版，版差分の tar + gzip ファイルを作成し，要求元に返す．存在しない場合はエラーを返す☆．

これにより，たとえば複数の機器どうしがネットワークに接続されているが，サーバへのネットワーク接続がない場合等に，それらの機器のうち最新の版を別の機器でも利用する，といったことが可能になる．

また，prov subscribe コマンドに -proxy を指定することも可能で，この場合は，プロキシ経由での更新通知となる．

2.3.3.5 機器側データのバックアップとリストア

prov execute コマンドに -auto-save on というオプションを指定すると，プログラム実行終了後に自動的に機器側のデータがバックアップされる．また，利用するパッケージや，マウント順等のメタデータも同時にバックアップする．

また，以前の説明では省略したが，prov execute コマンド実行時には，機器側の管理デーモンは，実行環境サーバに対し，現在の実行環境の版を要求する．もし，機器が持つ実行環境より高い版がサーバに存在すれば，自機器が持つ版との差分を取得することで，実行環境のリストアが行われる．

これはたとえば，機器 A である実行環境においてプログラムを実行した後，（同一種類の別機器である）機器 B で当該実行環境においてプログラムを実行しようとした際に起こる．このような場合でも，機器 A で更新したファイルが機器 B に正常にリストアされ，さらに機器 B で更新したファイルを再度機器 A にリストア可能である．すなわち，シンクライアントや Web アプリケーションのように，プログラム実行の続きを

☆ 要求されたパッケージについて，要求された版より新しい版が存在したとしても，要求された差分を構成できない場合がある（2.3.3.2 項参照）．この場合も存在しない場合と同様，エラーを返す．

別機器で行うことができる．

リストアの際，実行環境サーバでは，パッケージサーバの場合と同様，差分となるディレクトリを抽出する．さらに，データベースに保存されているメタデータとあわせて tar + gzip し，要求機器に送る．

複数の機器で同一の実行環境を同時に実行することによる矛盾を防ぐため，プログラムの実行時には，機器側の管理デーモンは，実行環境にロックをかけるよう，実行環境サーバに依頼する．すでにロックがかかっている場合は，その実行環境を使ってプログラムを実行することはできない．この際，ロックをかけた機器の IP アドレスを実行環境サーバに登録する（このデータを更新通知の際に利用する）．実行終了時に機器側の管理デーモンは，ロックを解除するよう，実行環境サーバに依頼する☆．

障害時に機器を入れ替える際等，ロックを解除できない場合のため，手動でロックを解除するコマンドも用意されている．また，頻繁な実行環境のバックアップを望まない場合に備え，手動でのロック・バックアップ・リストアコマンドも用意されている．

3. 評　　価

試作実装を行った本システムについて，評価を行った．

評価の目的は，実応用において，実用的なレスポンス時間で運用可能かどうか，また，それを実用的なサーバ数で実現できるか，の見通しを得るためである．

本システムを用いた応用にはさまざまなものが考えられるが，たとえば携帯電話のように，ユーザとの対話的処理を行うことが想定される場合，レスポンスタイムは十分短い必要がある（具体的な値としては，Web ページを表示する際の「8 秒ルール」から類推すると，8 秒程度以内である必要があると考えられる）．対話的処理を行わない場合は，レスポンスタイムはこれより長くてよい．

また，レスポンスタイムは機器の数にも依存する．多数の機器が同時にサーバにアクセスすると，レスポンスタイムが悪化する可能性があるためである．レスポンスタイムを保ちながら多数の機器に対応するためには，複数のサーバを用意し，処理を振り分ける必要がある．

ここで，具体的な機器の数や，それらの機器に対して具体的にどの程度の数のサーバを用意することが必要か，またその数は許容できるものなのか，は応用に依存する．

たとえば，2.1 節で述べた，コンビニエンスストアにおける Kiosk 端末の場合，店舗数が 1 万とし，1 台あたり 1 分に 1 度の頻度でサービスが利用されると仮定すると，1 万台/60 秒 = 166 台/秒のリクエストが到達することになる．

また，たとえばハードディスクレコーダを管理する場合を考える．DVD あるいはハードディスクレコーダは現在 40%程度の普及率を示しており，世帯数から計算すると，2000 万台程度存在することになる．このうち，半分の機器が本システムを利用すると仮定すると，1000 万台の機器を対象とすることになる．これらの機器が，夜 6 時〜7 時の 1 時間に集中して起動する場合を想定すると，1000 万台/3600 秒 = 2777 台/秒のリクエストが到達する．

必要なサーバの台数は，1 秒間にサーバが処理できるリクエスト数から求めることができる．これについては評価結果の部分で議論する．

試作実装には，RedHat Linux 9 のカーネルを 2.6.17.1 にアップグレードしたものに UnionFS 1.2 をインストールしたものを OS として用いた．また，HTTP サーバとして Apache 2.2 および PHP 5.1.2，データベースとして MySQL 4.1.12 とコネクションプーリングのため SQLRelay 0.37.1 を用いた．機器側の管理デーモンおよびコマンドラインインタフェースは C で記述し，パッケージのダウンロード等，HTTP(S) でアクセスされる部分は PHP で記述した．

このような試作システムを用い，その性能評価として，試作システムに負荷を与えた状態での実行時間を測定した．

まず，試作システムに与える負荷として，JMeter [3] を用い，パッケージサーバに対して，複数のスレッドを用い，継続的なアクセスを行った．ここでは，`prov download` と同じ方法で，約 100 Kbyte のファイルのダウンロードを連続して行っている．

この負荷は，機器からパッケージサーバへのアクセスとして，パッケージの更新をシミュレートするためのものである．すなわち，パッケージがサーバで更新された後，スレッド数分の機器がプログラムの実行を同時に開始した状況をシミュレートしている．これは，（新規パッケージ配布時は同時アクセスが起こらないと仮定すると）運用上最もアクセスが集中する状況になる．

対象とするパッケージ，およびその版は 1 つである．したがって与えられた負荷では，同じファイルがダウ

☆ なんらかの形で機器側の IP アドレスが変化した場合，現在の実装では手動でロックを解除する必要がある．この問題を解決するためには，MAC アドレス等，変化しにくい情報をキーにロックを行う必要がある．

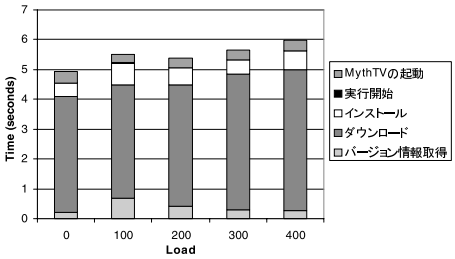

図 5 MythTV 第 1 版の実行
Fig. 5 Execution of MythTV version 1.

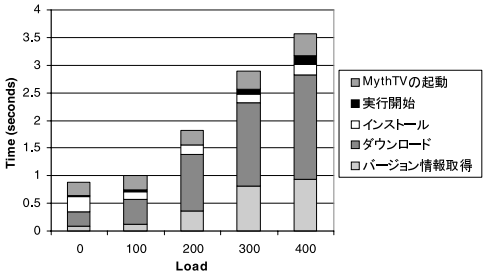

図 6 MythTV 第 2 版への更新・実行
Fig. 6 Update to MythTV version 2 and its execution.

ンロードされる．このため，パッケージがディスクからメモリにキャッシュされ，パッケージサーバ側の負荷が実際より小さく評価されている可能性がある．また，JMeter 側ではダウンロードしたファイルは保存していない．

同時にアクセスするスレッド数を 400 まで変化させて測定を行った．ここで，スレッド数が 500 以上では，リクエストに正常に返答できなくなることがあったため，測定は 400 までとした．

その状態で，本試作システムを用い，新しいプログラムのダウンロード・インストール・実行と，新版への更新を行い，その実行時間を測定した（実行環境のバックアップは行っていない）．対象プログラムとしては，実用的なプログラムである MythTV[4] というフリーのハードディスクレコーダソフトを用いた．また，第 2 版として，MythTV の見た目を変更するためのコンフィギュレーションファイルの配布を行うものとした．第 1 版のファイルサイズは約 11 MByte，第 1 版と第 2 版の差分のファイルサイズは 1 KByte 以下である．

測定環境としては，Athlon 2 GHz，512 MB の PC を用いた．パッケージサーバ，クライアント機器の OS としては上記の RedHat Linux を用いたが，JMeter の実行には Windows XP を用いた．機器間は 100 Mbps のイーサネットで接続されている．

図 5 から図 8 に評価結果を示す．

図 5 は初期状態からの第 1 版実行時の実行時間を示す．MythTV が実際に起動するまでの時間を，サーバからのバージョン情報取得，プログラムのダウンロード，インストール，chroot 環境でのプログラム実行，MythTV が起動し，利用可能になるまでの時間に分けてグラフに表示している．

ここで，第 1 版の実行までには，負荷が小さい場合で 5 秒程度，大きい場合で 6 秒程度かかっている．

この値は本節の冒頭で，実用的なレスポンスタイムとしてあげた 8 秒よりも小さく，要件を満たしている．

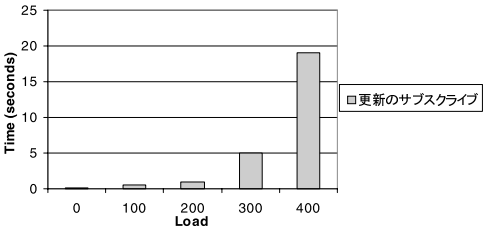

図 7 更新通知のサブスクライブ
Fig. 7 Subscription of update notification.

しかし，いずれの場合でも，約 11 MByte のファイルのダウンロードに 4 秒程度かかっており，実応用でネットワーク帯域がより小さいものになった場合は，この部分がより大きくなることも考えられる．

次に**図 6** に第 1 版から第 2 版へ更新時の実行時間を示す．この場合は，負荷が小さい場合で 1 秒程度，大きい場合で 3.5 秒程度かかった．いずれの場合も本節の冒頭で，実用的なレスポンスタイムとしてあげた 8 秒よりも小さく，要件を満たしている．

図 7 に更新通知依頼をサーバに行った際の実行時間を示す．この場合，負荷が 200 スレッドまでは十分小さい値だが，300，400 になると急増しており，400 では 19 秒程度までかかっている．これは，本処理がデータベースへの書き込みをともなう処理であるため，高負荷時の影響が大きかったためだと考えられる．本評価では，300 スレッド程度までなら，要件を満たすレスポンスタイムで利用可能である．

最後に**図 8** に更新通知を受け取って更新を行った際の実行時間を示す．この場合は，負荷がかかっている状態のいずれでも 1.2〜1.4 秒程度である．負荷のない場合は 0.4 秒程度である．この場合は，バージョン情報取得の時間がかからないため，図 6 の場合よりも短い時間でダウンロード・インストールができている．

以上のように，サーバ負荷が一定以下であれば，十分な応答速度で試作システムを利用できることが確認できた．

次に，多数の機器に対応する際に必要な，サーバ数

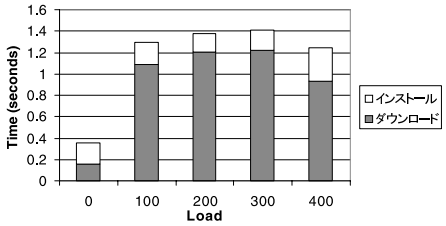

図 8 更新通知によるダウンロード・インストール
Fig. 8 Download and install by update notification.

について考察する．

スループットはスレッド数/完了時間となるため単純な評価はできないが，負荷を与えたスレッドにおいて 1 リクエストが 5 秒程度までで完了すると仮定すると（11 MByte のファイルのダウンロードを行った図 5 においても 5〜6 秒であったため，十分保守的な見積りだと考える），300 スレッドで動作した場合は 60 リクエスト/秒を 1 台で処理できることになる．

本章の冒頭で述べた，コンビニエンスストアにおける Kiosk 端末の場合，166 台/秒のリクエストが到達するため，これを処理するためには，3 台以上のサーバが必要になる．

また，ハードディスクレコーダの例の場合，2777 台/秒のリクエストが到達するため，47 台以上のサーバが必要になる．

これらのサーバ台数は，許容できる範囲かどうかは運用形態にも依存するが，おおむね現実的な範囲であると考える．

また，サーバ台数の追加には，Web サーバにおける負荷分散技術を応用し，ロードバランサや CDN 等を活用することができる．上記の台数によるリクエストの処理を，実際にロードバランサや CDN 等を利用することで達成できるかどうかを確認することは，今後の課題である．

4. 関 連 研 究

我々が提案するような，機器のソフトウェア管理および状態のバックアップを統合的に行うシステムは，我々の知る限り多くは存在しない．

古くからある類似したシステムとしては，Plan 9 [5] があげられる．Plan 9 では，すべてのファイルをサーバにある WORM（Write Once Read Many）型の大型記憶装置に保存し，ローカルにあるディスクはキャッシュとして扱う．すべてのファイル更新はファイルサーバに定期的にバックアップされる．また，プロセス単位でファイルシステムの「名前空間」を作ることができる．すなわち，/bin のようなディレクトリをプロセスごとに別のものにすることが可能である．これを用いれば，版管理の問題を解決することが可能であると考えられる．このように Plan 9 では我々のシステムが持つ機能の一部を実現しているが，プログラムの自動的な更新や，版管理等は直接は対応していない．また，この機能は特定のサーバと機器間に閉じており，パッケージサーバや実行環境サーバとして複数のものを仮定する我々のモデルとは異なる．

また，UnionFS を用いてサーバの管理を行った研究として，Blutopia [6] がある．このシステムでは，サーバ上にプログラムをインストールする際，UnionFS で作成した層にインストールし，このサーバを別の用途に用いる場合は，UnionFS で作成した層を取り除くことで，簡単にプログラムインストール前の状態に戻す．しかし，複数の実行環境間の共存や，版管理，システムのバックアップ等の機能は持たない．

また，異なる構成を持つシステムとして，Virtual Appliance があげられる．これは，特定のプログラム実行に特化した仮想マシンのイメージを配布するものである．仮想マシンを利用することで，別のプログラムとの干渉を排除することができるが，OS を含んだ仮想マシンのイメージを配布する必要があるため，頻繁な更新には性能面から対応できない．

また，サーバ側のクラスタを管理するためのシステムはいくつか知られている（たとえば文献 7)）．これらは，生死管理や異常検出を行うような，より密な管理を目指しており，大規模なユビキタス機器群の管理には性能的に対応できない．

バックアップに層構造を持つファイルシステムを利用した研究としては，文献 8) 等がある．これらはバックアップに特化して層構造を持つファイルシステムを利用している．

プログラムの更新に関しては，Windows Update や yum，apt 等のツールが有名である．Windows Update は配布前に旧版との互換性に関して十分なテストを行うことで，版間の非互換性に関する問題を解決している．このため，一般・多種のアプリケーションの更新用途には向かない．yum や apt のようなツールの場合は，プログラムをインストールする際に依存関係を解決し，必要なプログラムのインストールも行う．しかし，やはり旧版との非互換性がある場合の問題は解決できていない．

Java には OSGi [9] と呼ばれるシステムがあり，プログラムの版管理を行うことができる．また，Web Start [10] と呼ばれる仕組みを用いることで，つねに最新版をダウンロード，実行することができる．また，

Microsoft .NET Framework にも同様の機構として ClickOnce[11] というものがある．これらは我々のシステムにおける版管理機能と同様のものを提供するが，実行対象が限定されるという問題点がある．また，機器の状態をバックアップする機能は持たない．

携帯電話等のファームウェアをネットワーク経由で更新する仕組みとして，Innopath[12] や Bitfone[13] 等がある．これらは，旧ファームウェアと新ファームウェアの差分のみを送ることでシステムを更新する仕組みを提供するが，ファームウェア全体を更新するのみで，実行環境という概念はない．また，一度更新すると，旧版は失われる．

5. おわりに

本研究では，プログラムの配布およびデータバックアップを自動化することで，機器の管理を容易にするシステムを提案し，その設計，試作，評価を行うことで，その効率の良い実装が可能であることを示した．

本システムは，主にユビキタス機器の管理を目的としたものであるが，その技術は一般的なプログラム実行方式としてとらえることができる．本システムの提案する方式が，将来のより一般的なプログラム実行方式の端緒となる可能性がある．

今回の試作では，機器・サーバ間に信頼関係があることを仮定してシステムのセキュリティを設計している．今後の課題として，より強固なセキュリティ機能を実現することで，信頼関係の低い機器・サーバ間でも本システムを適用できるようにすることが考えられる．

また，今回の試作では，版の履歴を 1 つにまとめ，UnionFS で扱うディレクトリ層の数を減らす最適化は行わなかった．今後の課題として，このような層の融合を行う最適化の実現もあげられる．

参考文献

1) Quigley, D., Sipek, J., Write, C.P. and Zadok, E.: Unionfs: User- and Community-Oriented Development of a Unification File System, *Proc. 2006 Ottawa Linux Symposium* (2006).
2) Wright, C.P., Dave, J., Gupta, P., Krishnan, H., Quigley, D.P., Zadok, E. and Zubair, M.N.: Versatility and Unix semantics in namespace unification, *ACM Trans. Storage*, Vol.2, No.1, pp.74–105 (2006).
3) Apache Software Foundation: Apache JMeter. http://jakarta.apache.org/jmeter/index.html
4) Richards, I.: MythTV. http://www.mythtv.org/
5) Pike, R., Presotto, D., Dorward, S., Flandrena, B., Thompson, K., Trickey, H. and Winterbottom, P.: Plan 9 from Bell Labs, *Computing Systems*, Vol.8, No.3, pp.221–254 (1995).
6) Oliveira, F., Patel, J., Hensbergen, E.V., Gheith, A. and Rajamony, R.: Blutopia: Cluster Life-cycle Management, Technical Report RC23784, IBM (2005).
7) NEC Corporation：統合システム運用管理（WebSAM）. http://www.nec.co.jp/middle/WebSAM/
8) Muniswamy-Reddy, K., Wright, C.P., Himmer, A. and Zadok, E.: A Versatile and User-Oriented versioning File System, *3rd USENIX Conference on File and Storage Technologies* (*FAST 2004*) (2004).
9) OSGi Alliance: Welcome to the OSGi Alliance. http://www.osgi.org/
10) Sun Microsystems Inc.: Java Web Start Technology. http://java.sun.com/products/javawebstart/
11) Noyes, B.: Deploy and Update Your Smart Client Projects Using a Central Server, *MSDN Magazine*, Vol.19, No.5 (2004).
12) Innopath: Integrated Mobile Device Management. http://www.innopath.com/
13) Bitfone, Inc.: Bitfone corporation. http://www.bitfone.com/

(平成 19 年 1 月 22 日受付)
(平成 19 年 5 月 7 日採録)

荒木 拓也（正会員）

1971 年生．1994 年東京大学工学部電気工学科卒業．1999 年同大学大学院情報工学専攻博士課程修了．博士（工学）．同年 NEC 入社．2003 年から 2004 年にかけて米 Argonne National Laboratory 客員研究員．現在，サービスプラットフォーム研究所勤務．プログラミング言語，並列処理，分散処理，グリッドコンピューティング，ユビキタスコンピューティングに興味を持つ．

小倉　章嗣（正会員）

1980 年生．2002 年東京工業大学理学部情報科学科卒業．2004 年同大学大学院情報理工学研究科修士課程修了．同年 NEC 入社．現在，サービスプラットフォーム研究所勤務．モバイルコンピューティング，グリッドコンピューティング，HPC 等に興味を持つ．