情報処理学会論文誌 Vol. 49 No. 6 2070–2080 (June 2008)

投機的仮想マシンスタンバイ制御によるプロビジョニング高速化

町田文雄[†1] 川戸正裕[†1] 前野義晴[†1]

データセンタや企業システムでは，低コストかつ低消費電力でディペンダブルなシステムを実現することが課題となっている．コストや電力を削減するため，複数のアプリケーションシステムでサーバを共有し，必要なときにサーバの構成を変更するプロビジョニング技術が利用できる．しかし，複雑な構成を持つシステムのプロビジョニングには時間がかかるため，必要なときに即座に構成を変えることは難しく，ディペンダビリティの低下が懸念される．本論文では，プロビジョニング処理を高速化するため，ホットスタンバイ用の仮想マシン（VM）を用いて，投機的にプロビジョニングを事前実行する手法を提案する．プロビジョニングを事前実行するために，ロジスティック回帰分析によるプロビジョニング予測モデルを用いる．実稼働する企業システムの負荷変動データを用いてシミュレーション評価を行った結果，20 分先読み予測モデルを用いて 50%の確率で 10 分以上の高速化の効果が期待できることを確認した．

High-speed Server Provisioning by Speculative Virtual Machine Standby Control

Fumio Machida,[†1] Masahiro Kawato[†1] and Yoshiharu Maeno[†1]

Data centers and enterprise systems have to be dependable with low costs and low power consumptions. For the reduction of costs and power consumptions, server provisioning is a useful technique to reconfigure the configuration of shared server on demand. However, in the complex systems, the process of the server provisioning takes long time and impedes prompt solutions to system problems. The long time provisioning process decreases the system dependability. In this paper, we propose a technique to shorten the provisioning processing time by speculative provisioning execution on the virtual machine as hot standby. In order to start the provisioning execution in advance, we provide a prediction function for the provisioning requirement based on the logistic regression function with using the system performance metrics. From the evaluation using the actual performance data of enterprise systems, for 50% of the occurrences of server provisioning, the provisioning time is shorten over 10 minutes by using the 20-minutes look-ahead prediction model.

1. はじめに

データセンタや企業システムでは高度な信頼性や可用性が要求されるとともに，コストや消費電力の削減が求められている．システムダウンやサービス品質の低下を防ぐため，企業やデータセンタの多くは予備サーバを用いたシステムの冗長化を行っている．予備サーバを用いることで，システム過負荷時や障害発生時に，サーバの追加やリプレースによって信頼性や可用性を維持することができる．しかし，過剰に用意された予備サーバはコストや電力を無駄に消費するため，システムの利用効率を低下させる原因となる．データセンタにおけるサーバの利用率は 20%程度[9] ともいわれており，予備のサーバは十分に活用されていない．サーバの利用効率を改善し，より少ないサーバで安定したサービスを提供し続けることが課題である．

サーバの利用効率改善のためには，複数のアプリケーションシステムでサーバを共有し，必要に応じてサーバの構成を動的に変更するサーバのプロビジョニング技術が利用される．サーバを共有することで利用効率を改善し，無駄なサーバを削減できる．NEC SigmaSystemCenter[10]，IBM Provisioning Manager[11] などのプロビジョニングツールはサーバの構成変更時に必要となる処理を GUI から簡単に自動実行する機能を備えている．

ところが，サーバのプロビジョニングには時間がかかるため，システムの過負荷状態や障害に対して迅速に対応することは難しい．OS やアプリケーションのセットアップだけでなく，ネットワーク機器の設定変更や監視サーバへの登録処理も含まれる場合があるため，複雑なシステムではプロビジョニングの完了までに数十分から数時間を要する．必要なときに即座にサーバを利用することができなければ，システムの信頼性や可用性を維持できない．

本論文では，サーバのプロビジョニングを高速化することで，共有サーバ環境におけるシステムディペンダビリティの向上を目指す．サーバのプロビジョニングはファイバチャネルによる Storage Area Network（FC-SAN）を導入してリモートブート構成を採用することでも高速化できるが，既存のサーバ環境を FC-SAN 構成に変更するコストは大きく，導入障壁は高い．本論文では，既存のサーバ環境を維持してサーバのプロビジョニングを高速化

†1 日本電気株式会社サービスプラットフォーム研究所
Service Platforms Research Laboratories, NEC Corporation

するため，スタンバイ用の仮想マシン（VM）を用いてプロビジョニングを事前実行する手法を提案する．既存のサーバ環境は VM 上で再現可能である．提案手法は，許容範囲を超えた負荷の変動や障害発生によるサーバのプロビジョニング要求を事前に予測し，投機的にプロビジョニングを事前実行する．予測手法として，システムの負荷変動データを用いたロジスティック回帰分析による手法を提案し，性能評価を行った．

本論文の構成を以下に示す．2 章では，サーバのプロビジョニング高速化に関連する既存技術について述べる．3 章では，VM を用いたプロビジョニング高速化手法を提案し，4 章でプロビジョニング要求の予測手法を示す．5 章で実稼働する企業システムの負荷変動データを用いたシミュレーション評価結果を示し，6 章で結論を述べる．

2. サーバプロビジョニング

サーバのプロビジョニング処理過程を示し，プロビジョニング処理を高速化する手法として利用可能な既存技術を示す．

2.1 プロビジョニング処理

サーバのプロビジョニングの具体的な処理過程を**図 1** に示す．ここではロードバランサを用いた負荷分散構成を持つ Web アプリケーションシステムを想定しており，アプリケーション（AP）サーバをプロビジョニングする例を示している．OS，AP のセットアップ，ネットワークの構成変更，管理サーバ，データベース（DB）サーバへの登録を行い，最後にロードバランサ（LB）の負荷分散対象に加えることで，サービスを開始可能な状態となり，プロビジョニングが完了する．

OS や AP のセットアップは，インストール処理を含む場合と含まない場合がある．インストール済みのディスクイメージを参照することができれば，インストールは必要ない．

VLAN やファイヤウォールをシステムの構成要素として含む場合は，ネットワーク機器の設定変更処理も行う．同様に，SAN や NAS を使って構成されるシステムでは，ストレージの構成変更が必要となる．

さらに，他のサーバ群と連携してサービスを提供するシステムでは，稼働中のサーバに対して，新規サーバの追加を知らせるための処理が必要である．たとえば，DB サーバを利用するシステムでは，DB サーバへの登録処理が必要となる場合がある．

2.2 プロビジョニングステップの高速化

プロビジョニング処理を高速化するため，プロビジョニングを構成する各処理を高速化する手法を考える．

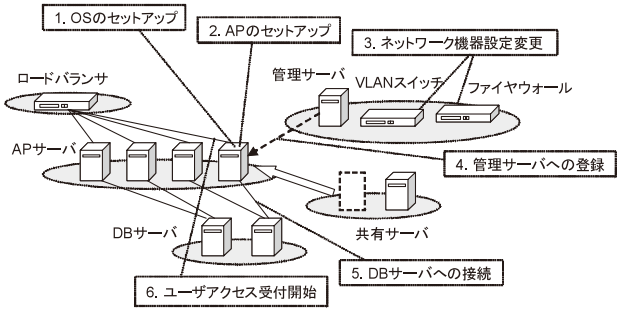

図 1　プロビジョニング処理過程
Fig. 1　Provisioning execution process.

OS やアプリケーションのインストールや設定作業を高速化する手法として，アプリケーション構成済みの OS イメージをリモートから配布（デプロイ）して起動する方法がある．プロビジョニングツールの多くはこれらのデプロイ管理機能を持っている．

OS の起動を高速化する手法としては，Mac OS X の launchd [12] や Linux の InitNG [13] などが知られている．これらの機能は，デバイスの初期化やサービスの起動処理を並列実行することで OS の起動にかかる時間を短縮する．また，Windows XP ではプリフェッチ機能により，起動時のディスクアクセスを削減して起動高速化を実現している．

また，アプリケーションサーバの起動は，起動時に読み込むサービスを最小限に絞ることで高速化できることが知られている．たとえば，軽量 J2EE サーバである GlassFish [14] では Lazy Initialization 機能として実装されている．

OS やアプリケーションサーバの起動高速化技術は，プロビジョニング処理の高速化に効果があるが，汎用的な解決手段ではない．企業システムやデータセンタは多様なシステム構成を持つため，適用可能な解決手段は限定される．また，稼働中の OS やアプリケーションサーバに変更を加えることは，ユーザの観点から望ましくない．

2.3 スタンバイによる高速化

あらかじめ必要となるアプリケーションのサーバが明らかである場合は，共有サーバを事前に構成変更してホットスタンバイさせることで，プロビジョニング要求に対して迅速な対応が可能となる．しかし，複数のアプリケーションを持つシステムでは，事前に必要となるアプリケーションのサーバは明らかではない．アプリケーションごとにホットスタンバイ用のサーバを用意すると，サーバ共有による利用効率の改善が達成されない．

そこで，アプリケーションシステムを構築する際の中間状態でスタンバイさせたサーバを共有することで，複数のアプリケーションシステムに対するプロビジョニング時間を平均的に短縮する手法が提案されている[8]．たとえば，異なるアプリケーションで利用する OS が共通である場合，OS をセットアップした状態でスタンバイしておくことで，いずれのアプリケーションのプロビジョニングが要求された際にも，OS のセットアップ処理が不要となり，プロビジョニング時間を短縮できる．ただし，この手法は未使用の共有サーバを迅速にプロビジョニングする手法であり，すでに共有サーバが別のシステムとして利用されている場合には適用できない．

2.4　VM を利用したプロビジョニング

VM を利用することで，プロビジョニングに利用する OS イメージの管理がより容易になる．一般的に，物理サーバ上で作成した OS のイメージを，異なるハードウェア構成を持つ物理サーバ上で再現することは困難である．しかし，VM のイメージ管理機能を用いることで，ハードウェアの構成も含めたシステム環境を VM イメージとして保存して維持できるため，異なるハードウェア構成を持つ物理サーバへのプロビジョニングが容易になる．また，稼働中の VM をサスペンドして保存し，必要となった際にレジュームして実行を継続することも可能である．

しかし，プロビジョニング対象となるサーバの種類が多い場合に，FC-SAN などの共有ディスク環境を持たないシステムでは，すべてのサーバの VM イメージを保持することは難しい．プールサーバ上のディスク領域が限られている状況では，必要となった際に改めてサーバを構築するか，リモートから VM のイメージを配布する必要がある．本論文で提案する手法は，FC-SAN などのシステムにも応用可能だが，特に共有ディスク環境を持たず，プールサーバ上のディスク領域が限られているシステムを対象とする．

3.　VM スタンバイシステムによるアプローチ

既存のサーバ環境を変えることなく，共有サーバの利用効率を改善するプロビジョニング高速化手法として，VM を用いたスタンバイ手法を提案する．

3.1　VM スタンバイシステム

提案手法はプロビジョニングの事前実行のために VM を利用する．Xen[1] や VMware ESX Server[15] などの仮想マシンモニタ（VMM）は物理サーバ上に複数の VM を生成する機能を持つ．VMM を利用することで，稼働中のサーバと並列してプロビジョニングを事前実行するための VM を動作させることができる．プロビジョニングを事前に実行し，スタンバイしておく VM の実行環境（領域）を持つシステムを VM スタンバイシステムと呼ぶことにする．

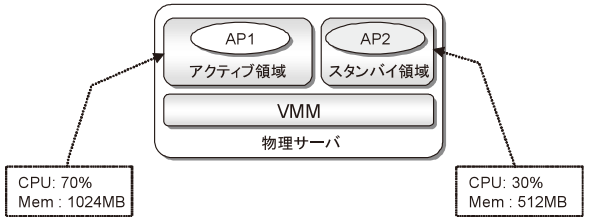

図 2　VM スタンバイシステムの構成
Fig. 2　Configuration of a VM standby system.

図 2 に VM スタンバイシステムの構成例を示す．VM スタンバイシステムは VMM によって仕切られたアクティブ領域とスタンバイ領域からなる．アクティブ領域では，運用中のアプリケーションを VM 上で稼動させ，アプリケーションを実行するために十分な資源（CPU，メモリ，I/O 帯域）を割り当てる．一方，スタンバイ領域では，近い将来利用される可能性のあるアプリケーションを VM 上で稼動させ，必要最低限な資源を割り当てる．たとえば，70%の CPU 資源をアクティブ領域に割り当て，残り 30%の CPU 資源をスタンバイ領域に割り当てる．

既存のサーバ環境は VM 上で再現可能なため，アプリケーションシステムを構成する OS やアプリケーション，ライブラリに変更を加える必要はない．また，スタンバイのために専用の物理サーバを用意する必要がないため，共有サーバの台数を削減して利用効率を改善できる．

3.2　Active/Standby スイッチ

VM スタンバイシステム上では，Active/Standby スイッチ（A/S スイッチ）機能によりプロビジョニングを実行する（**図 3** 参照）．A/S スイッチ機能とは，アクティブ領域とスタンバイ領域の資源割当て量をアプリケーションを稼働させたまま動的に入れ替える機能である．A/S スイッチにより，スタンバイ領域で稼動していたアプリケーションは運用のための十分な資源を割り当てられ，ロードバランサの負荷分散対象に加えられることによってサービス開始状態となる．

スタンバイ領域を活用することで，プロビジョニング処理ステップのいくつかを事前に実行してスタンバイ状態を作ることができるため，ホットスタンバイを用いた場合と同様なプ

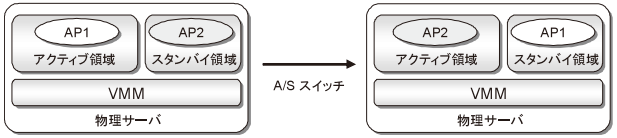

図 3　A/S スイッチ機能
Fig. 3　A/S switch function.

ロビジョニング高速化効果が得られる．

ただし，VM スタンバイシステムを使ってプロビジョニング時間を短縮するためには，プロビジョニングが要求されるアプリケーションを事前に予測し，スタンバイ領域に準備する必要がある．プロビジョニング要求の予測が的中すれば A/S スイッチによるプロビジョニングで高速化の効果が得られるが，的中しなかった場合は手戻りが発生し，プロビジョニング時間が逆に増加してしまう可能性がある．そこで，プロビジョニング要求を予測する手法を次に述べる．

4. プロビジョニング要求の予測

VM スタンバイシステムを用いてプロビジョニングを高速化するために，プロビジョニング要求を予測する手法を提案する．

4.1 プロビジョニング要求

プロビジョニングは静的に要求される場合と動的に要求される場合がある．管理者が事前にスケジュールを設定してプロビジョニングする場合を静的なプロビジョニング要求と呼ぶ．一方，障害発生時や，CPU 利用率などの負荷指標が指定したしきい値を超えた際に自動的にプロビジョニングが要求される場合を動的なプロビジョニング要求と呼ぶ．ここでは，動的なプロビジョニング要求を予測する手法を扱う．

4.2 基本アプローチ

動的なプロビジョニング要求を予測するため，性能情報を継続的に監視して，プロビジョニング要求が発生する前の前兆を検出する．動的なプロビジョニングは高負荷状態や障害時など，システムの異常状態に対処することを目的として要求される．システム状態の変化は性能情報の変化として観測可能なため，性能情報の変化を解析することで，異常な状態に至る前兆を検出できると考えられる．性能情報の相関関係に基づいてシステムの異常状態を検出しようとする手法も提案されている[6]．

前兆を検出する最も単純な方法として，プロビジョニング要求を判断する負荷指標（たとえば CPU 利用率）にプロビジョニング要求予測のための別のしきい値を設定する方法がある．この手法を 2 段階しきい値法と呼ぶことにする．負荷が単調増加してプロビジョニングのしきい値に達する場合は，2 段階しきい値法で予測ができる．

別の方法として，しきい値を設定した負荷指標の性能変動を時系列データとして解析し，しきい値を超える時点を予測する方法がある．時系列解析のモデルとしては AR，MA，ARMA，ARIMA モデルなどが知られており，システム負荷指標の予測にも応用されている[2]．

2 段階しきい値法や時系列解析を用いた方法は，対象となる負荷指標の過去のデータのみに着目してしきい値を超える時点の予測を行うため，適用可能なシステムが限定される．実際には，しきい値を設定した指標とは別の指標に前兆が見られる場合や，他のアプリケーションの負荷変動との相関に前兆が見られる可能性がある．また，障害に起因したプロビジョニング要求は予測できない．

そこで，本研究では，しきい値を設定した負荷指標だけでなく，他の負荷指標や他のアプリケーションの負荷情報を用いて，プロビジョニング要求を予測するモデルを用意した．モデルとしてロジスティック回帰モデルを用い，モデルの学習のために時系列データ群から抽出した相関ルールを用いる．

4.3 時系列データ群から相関ルールの抽出

予測モデルを作成するために，過去の負荷変動の時系列データを解析してプロビジョニング要求に対する相関ルールを作成する．時系列データ群からの相関ルール抽出は Das ら[3]の手法などが知られている．ここでは，負荷変動の現在の状態 S と時間 t 経過後のプロビジョニング要求 r の相関を知りたい．抽出すべき相関ルールを以下のように記述する．

$$S \xrightarrow{t} r \tag{1}$$

このルールは S の状態が観測された時点から t 後に r のプロビジョニングが要求されたという事実を表している．S の表現形式として，しきい値を設定した負荷指標だけでなく，システム中の様々な負荷指標の現在値，統計値を要素に持つベクトルを利用することができる．r は，S の観測から t 後に発生しているプロビジョニング要求を調べて決める．負荷指標にプロビジョニングのしきい値が設定されている場合は，t 後の負荷指標の値を参照して判定する．特定のアプリケーションのプロビジョニングが要求される場合は r にアプリケーション名を格納し，いずれのアプリケーションのプロビジョニングも要求されない場合は “none” を格納する．

4.4　ロジスティック回帰分析による予測モデル

複数の負荷指標値から，プロビジョニング要求が発生する確率を予測するためにロジスティック回帰分析を用いる．ロジスティック回帰分析はデータマイニング手法の 1 つであり，複数の説明変数を用いて 2 値の目的変数や比率などを求める際に利用される[4]．アプリケーション i のプロビジョニングが要求される確率 q_i は，相関ルールの観測された状態 S のベクトル表現 $v = (x_1, x_2, \ldots, x_n)$ を用いて以下のようにモデル化される．

$$q_i(v) = \frac{\exp\left(\beta_{i0} + \sum_{j=1}^{n} \beta_{ij} \cdot x_j\right)}{1 + \exp\left(\beta_{i0} + \sum_{j=1}^{n} \beta_{ij} \cdot x_j\right)} \tag{2}$$

ここで $\beta_i = (\beta_{i0}, \beta_{i1}, \ldots, \beta_{in})$ はロジスティック回帰式の回帰係数で，最尤法によって推定する．特定のアプリケーションのプロビジョニングが要求される場合を 1 として相関ルールから回帰係数を求める．

また，プロビジョニングが要求されない確率についても同様な手法を用いて予測モデルを作成することができる．抽出された相関ルールにおいて，プロビジョニング要求 r が “none” である場合を 1 として回帰係数を求める．生成されたモデルで予測される，プロビジョニングが要求されない確率を $q_\emptyset$ と表す．

4.5　スタンバイ状態の決定

各アプリケーションに対して予測されたプロビジョニング要求確率に基づき，期待プロビジョニング時間を最短にするスタンバイ状態を決定する．

最適なスタンバイ状態は，各アプリケーションに対するプロビジョニング要求確率の相関状態に依存して決まる．1 つのアプリケーションのプロビジョニング要求確率が突出して高い場合は，そのアプリケーションをスタンバイ状態として選択すべきである．しかし，同時に別のアプリケーションのプロビジョニング要求確率も高い場合は，何もない状態で待機した方が良い．予測が外れた場合に手戻りが発生し，プロビジョニング時間が増加してしまう危険性があるためである．

そこで，予測が外れた場合のリスクも考慮するため，期待プロビジョニング時間を最短にするスタンバイ状態を決定する．期待プロビジョニング時間はプロビジョニング要求確率とスタンバイ状態からプロビジョニング完了までにかかる時間との積和で計算する．まず，スタンバイ状態 j を仮定し，その状態から各アプリケーション i へのプロビジョニング時間 c_{ji} を求める．スタンバイ状態からプロビジョニング完了までにかかる時間は事前に測定しておく．次に，アプリケーション i に対するプロビジョニング要求確率 p_i と，算出したプロビジョニング時間 c_{ji} を乗じ，総和を取ることで，スタンバイ状態 j における期待プロビジョニング時間 E_j が求まる．プロビジョニングの対象となるすべてのアプリケーションの集合を A とし，E_j は以下のように表せる．

$$\mathrm{E}_j = \sum_{i \in \mathrm{A}} p_i \cdot c_{ji} \tag{3}$$

$$p_i = \frac{q_i}{q_\emptyset + \sum_{k \in \mathrm{A}} q_k} \tag{4}$$

ここでプロビジョニング要求確率 p_i は，予測モデルによって求めた要求確率 q_k と $q_\emptyset$ を用い，$\sum p_i = 1$ となるように標準化している．最後に期待プロビジョニング時間が最小となる状態をスタンバイ状態として選択する．

スタンバイ状態の決定は，いずれかのプロビジョニング要求の発生する確率 $1 - q_\emptyset$ がしきい値 d を超えた場合に制限する．スタンバイ状態を 1 度決定するとスタンバイ状態を構築する処理が開始されるため，その後しばらくスタンバイ状態の変更ができない．不要なスタンバイ状態の構築処理を避けるため，プロビジョニング要求の発生する確率にしきい値を設定することで，スタンバイ状態の決定頻度を制限する．以上より，スタンバイ状態 s は以下のように決まる．ただし $\emptyset$ は空の状態とする．

$$s = \begin{cases} j \,|\, \min \mathrm{E}_j & 1 - q_\emptyset \geq d \\ \emptyset & \text{otherwise} \end{cases} \tag{5}$$

5. 評　　価

提案手法を評価するため，企業で実稼働する社内システムの負荷変動データを用いてシミュレーション評価を行った．まずシミュレーション対象のシステム構成を説明する．次に過去の負荷変動データを用いたプロビジョニング予測モデルの作成について述べる．最後に予測モデルの評価結果と既存の予測手法との比較結果を示す．

5.1　評 価 対 象

社内ユーザ向け Web アプリケーションシステムである営業管理システム（app1），工数管理システム（app2），予算管理システム（app3）を対象とした．3 システムは Web アプリケーションのプレゼンテーションとロジックを実装するフロントエンドと，データを格納

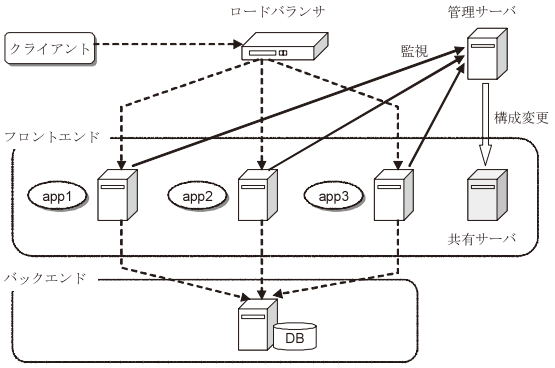

図 4　シミュレーション対象のシステム構成
Fig. 4　Target system configuration for simulation.

して管理するバックエンドで構成される．フロントエンドは複数のサーバで負荷分散をする構成がとられている．

現在 3 つのシステムは独立して稼動しているが，3 システムでフロントエンドのサーバを共有する構成をとることで，サーバ資源の利用効率向上が期待できる．そこで，**図 4** に示す最小構成のシステムをシミュレーション評価の対象とした．

各システムのフロントエンドとしてサーバが 1 台ずつあり，共有サーバとして VM スタンバイシステムが 1 台ある．また，共有サーバの構成を管理するための管理サーバがある．管理サーバは各システムの負荷状態を監視し，負荷がしきい値を超えて過負荷状態となったシステムに対して共有サーバをプロビジョニングして割り当てる．

5.2　予測モデルの作成

2006 年 7 月時の各システムの 1 カ月間の負荷変動データをサンプルとして用い，過負荷状態時に要求されるプロビジョニングの予測モデルを作成する．負荷変動データは CPU 利用率やディスク使用率を 10 分間隔でサンプリングしたデータで，4,464 レコードからなる時系列データである．過負荷状態は CPU 利用率が 30%を超えた状態とした（負荷変動データを採取したシステムでは計算機資源が十分に割り当てられており，ピーク時でも CPU 利用率は 50%程度であった）．各システムで過負荷状態として検出されたレコードの数，および全レコード数に対する割合は**表 1** のとおりであった．

はじめに，過負荷状態をプロビジョニング要求として，プロビジョニング要求と 20 分前の負荷状態を相関ルールとして抽出した．負荷状態を表す性能特徴ベクトル v は，各アプリケーションシステムの CPU 利用率（usr/sys/wio），ディスク使用率（device busy），クライアント接続数（Apache のセッション数）を用いて以下のように表現した．ベクトルの要素の定義は**表 2** に示すとおりである．

$$v = (x_1, x_2, \ldots, x_{16}) \tag{6}$$

また，抽出された相関ルールの例を**図 5** に示す．

次に，抽出した相関ルールを学習データとして，3 システムに対するプロビジョニング要求の予測モデルを作成した．予測モデルは 4.4 節に示したロジスティック回帰モデルを用いた．ただし，回帰係数の導出にはデータマイニングツールの 1 つである WEKA [5),16)] を用い，ロジスティックモデル木[7)] と呼ばれる手法を採用した．ロジスティックモデル木は決定木とロジスティック回帰分析を組み合わせた手法で，予測に不要と考えられる説明変数を枝刈りできる．

生成された 20 分先読み予測モデル（Model-20）の一覧を**表 3** に示す．予測モデルは表

表 1　過負荷状態のレコード数
Table 1　The number of records in overload.

システム	レコード数	全レコードに対する割合
app1	19	0.0043
app2	20	0.0045
app3	15	0.0034
合計	54	0.0121

表 2　性能特徴ベクトルの要素
Table 2　Elements of a feature vector.

メトリック		app1	app2	app3
CPU (%)	Usr	x_1	x_7	x_{12}
	Sys	x_2	x_8	x_{13}
	Wio	x_3	x_9	x_{14}
Disk (%)	Disk 1	x_4	x_{10}	x_{15}
	Disk 2	x_5	x_{11}	x_{16}
クライアント数		x_6	-	-

表 3　各アプリケーションに対する 20 分先読み予測モデル
Table 3　20-minutes look-ahead prediction model for provisioning requests of application systems.

i	$f_i(v)$
app1	$-5.01+0.31\cdot x_3+0.03\cdot x_5+0.03\cdot x_6+0.44\cdot x_9-0.06\cdot x_{12}+1.27\cdot x_{13}-0.13\cdot x_{14}-0.1\cdot x_{16}$
app2	$-12.88+0.07\cdot x_1-0.1\cdot x_3+0.34\cdot x_7-0.25\cdot x_9+1.43\cdot x_{11}+0.14\cdot x_{12}$
app3	$-12.87+0.2\cdot x_1-1.48\cdot x_2+0.09\cdot x_6-1.01\cdot x_9+0.52\cdot x_{12}$
none	$6.35-0.07\cdot x_1-0.27\cdot x_2-0.06\cdot x_3-0.02\cdot x_6-0.05\cdot x_7+0.13\cdot x_{10}-0.09\cdot x_{12}+0.52\cdot x_{14}$

S: 負荷状態(性能特徴ベクトル v) $\xrightarrow{t:\text{時間}}$ r: プロビジョニング要求
(14,3,3,15.2,10.9,18,3,1,1,5.6,3.7,2,1,1,5.9,3.7) $\xrightarrow{20\text{min}}$ none
(18,3,3,17.9,13.1,22,4,1,0,5.1,3.3,2,1,0,5.4,3.4) $\xrightarrow{20\text{min}}$ none
(11,2,2,12.4,8.6,18,33,2,0,5.4,3.6,3,1,0,5.8,3.7) $\xrightarrow{20\text{min}}$ app2
(16,3,4,18.3,13.9,14,31,1,0,5.4,3.6,3,1,1,6,3.8) $\xrightarrow{20\text{min}}$ app2
(12,2,2,11,7.5,21,33,2,0,5.2,3.3,2,1,1,5.4,3.4) $\xrightarrow{20\text{min}}$ app2

図 5　性能特徴ベクトルと相関ルール
Fig. 5　Performance feature vectors and correlation rules.

表 4　プロビジョニング時間 c_{ji}（分）
Table 4　Provisioning Time (minute).

	要求された状態 i		
スタンバイ状態 j	app1	app2	app3
app1	0	15	15
app2	15	0	15
app3	15	15	0
none	10	10	10

で示される各関数 $f_i(v)$ を用いて次のように表現される．

$$f_i(v)=\beta_{i0}+\sum_{j=1}^{n}\beta_{ij}\cdot x_j \tag{7}$$

$$q_i(v)=\frac{\exp(f_i(v))}{1+\exp(f_i(v))} \tag{8}$$

また，同様な手続きによって各アプリケーションに対する 40 分先読み（Model-40），60 分先読みを行う予測モデル（Model-60）も作成した．

5.3　予測機能の評価

作成した予測モデルによるプロビジョニング高速化の効果を評価するため，2006 年 8 月時の負荷変動データを用いてシミュレーション評価を行った．

評価用の負荷変動データと同じ負荷変動が図 4 に示す構成のシステムでも発生したものと仮定する．評価用の負荷変動データにおいては 20 レコードが過負荷状態として検出され，連続した過負荷状態で数えると合計で 8 回の過負荷状態が検出された．つまり，評価対象のシステムでは 8 回のプロビジョニング要求が発生する．

作成した予測モデルは 8 回のプロビジョニング要求を事前に間違いなく予測し，スタンバイ状態を準備できるかどうかで評価される．

スタンバイ状態 j から要求されたアプリケーションシステムの状態 i へのプロビジョニング時間 c_{ji}（分）は**表 4** に示すとおりに設定した．また，不必要なスタンバイ状態の構築処理を避けるため，プロビジョニング要求確率に対するしきい値 d は 0.4 に設定した．

Model-20，Model-40，Model-60 を用いて評価した結果を**表 5** (a) に示す．事前にスタンバイ状態の構築が行われなかった場合の結果は空欄としてある．スタンバイ状態が決定された場合は，先読み時間（分）と，予測の結果（hit or miss）を記述している．たとえば，“20, hit” は 20 分前にプロビジョニング要求の予測に成功したことを示している．

表 5　予測モデルの評価結果
Table 5　Evaluation results of the prediction model.

プロビジョニング要求				(a)			(b)		
日付	曜日	時間	対象	Model-20	Model-40	Model-60	Model-20’	Model-40’	Model-60’
2006/8/2	Wed	16:10-16:20	app2						
2006/8/3	Thu	10:20-11:20	app2	20, hit	10, hit	20, hit	20, hit	20, hit	10, miss
2006/8/7	Mon	11:40-12:10	app1				20, hit		
2006/8/8	Tue	11:20-11:40	app1	10, hit	10, miss		20, hit	10, hit	20, hit
2006/8/25	Fri	11:40-12:00	app1					10, hit	
2006/8/28	Mon	11:30-11:50	app1	10, hit			10, hit		30, hit
2006/8/31	Thu	7:30-7:50	app3						
2006/8/31	Thu	11:00-11:20	app1	10, hit				10, miss	20, hit

Model-20 では 8 回のプロビジョニング要求のうち 4 回のプロビジョニング要求を事前に予測できている．残りの 4 回についてもプロビジョニング要求の予測ミスではないため，予測を行わない場合と比較してプロビジョニング時間が悪化することはない．全体で 50 分のプロビジョニング高速化の効果が得られる．

これに対し，40 分先読み，60 分先読み予測モデルでは 1 回しか予測に成功していない．40 分先読み予測モデルは予測ミスが発生しているため，全体でプロビジョニング高速化の効果はほとんど得られない．

5.4　改良予測モデルでの評価

各アプリケーションの負荷変動は曜日や時間帯に依存して変化する傾向が見られる．たとえば，昼休み時間帯である 12:00〜13:00 には負荷が下がることや，月曜の午前中や金曜には負荷が高まる傾向が見られる．そこで，曜日や時間帯の情報を予測モデルに組み込むことで，予測性能が向上するかを検証した．

ロジスティック回帰分析では数値データ以外のパラメータを説明変数として扱うことが可能なため，性能特徴ベクトルを拡張することで，曜日や時間帯の情報を予測モデルに組み込むことができる．

2006 年 7 月のデータを学習用データとして用い，2006 年 8 月のデータを用いて同様な評価を行った．20 分先読み（Model-20’），40 分先読み（Model-40’），60 分先読み予測モデル（Model-60’）を使って評価した結果を表 5 (b) に示す．

いずれの予測モデルにおいても，予測の成功回数あるいは先読み時間で性能の改善が見られる．Model-20’ では全体で 70 分のプロビジョニング高速化の効果が得られる．Model-40’，Model-60’ は予測ミスが 1 回発生しているが，予測成功回数は増加している．

5.5　予測モデルの比較評価

2 段階しきい値法，時系列解析法を用いて CPU 利用率の負荷変動を予測し，プロビジョニング要求の予測を行った場合と比較評価した．

2 段階しきい値法では，プロビジョニングを決定するしきい値とは別に，スタンバイ状態の生成を開始するためのしきい値を設定する．評価用の負荷変動データに対し，CPU 利用率のしきい値を 29%〜25%に設定して評価した．

時系列解析法では，CPU 利用率の変動の履歴を自己回帰モデルにあてはめ，その後に続く CPU 利用率の値を推定し，将来のプロビジョニング要求を予測する．CPU の Load average の予測で実績のある[2] AR モデルを採用し，過去 60 分間の CPU 利用率の変動データから 20 分後の CPU 利用率を予測するモデルを使って評価した．評価結果を**表 6** に示す．プロビジョニング要求が発生する 60 分以上前にスタンバイ状態を決定した場合は先読み時間を “60+” と表記した．

2 段階しきい値法は，しきい値設定が低い値であるほど，先読み可能な時間は長くなり，

表 6　2 段階しきい値法と時系列解析法を用いたプロビジョニング予測の結果
Table 6　Prediction results by two level thresholds method and time series analysis method.

プロビジョニング要求				2 段階しきい値法					時系列解析法
日付	曜日	時間	対象	thresh-29	thresh-28	thresh-27	thresh-26	thresh-25	ar-20
2006/8/2	Wed	16:10-16:20	app2	60+, hit	60+, hit	60+, hit	60+, hit	60+, hit	
2006/8/3	Thu	10:20-11:20	app2						
2006/8/7	Mon	11:40-12:10	app1						
2006/8/8	Tue	11:20-11:40	app1			10, hit	10, hit	10, hit	
2006/8/25	Fri	11:40-12:00	app1						
2006/8/28	Mon	11:30-11:50	app1					10, hit	
2006/8/31	Thu	7:30-7:50	app3					10, hit	
2006/8/31	Thu	11:00-11:20	app1	10, hit	10, hit	10, hit	20, hit	30, hit	

予測の的中回数も増える．しきい値を 25%に設定した場合は，5 回のプロビジョニング要求の予測に成功している．ただし，スタンバイ状態の決定頻度が増加するため，無駄なプロビジョニング処理や予測ミスの回数も増加する．1 カ月間のプロビジョニングの要求頻度は表 1 より 0.0121 程度であるので，スタンバイ状態の決定頻度は 0.01〜0.02 程度であることが望ましい．**表 7** は各予測手法による予測成功回数とスタンバイ状態決定頻度を示している．2 段階しきい値法と比較し，提案手法（Model-20 および Model-20'）はスタンバイ状態決定頻度を抑え，効率良くプロビジョニング要求を予測している．

2 段階しきい値法は，提案手法で予測できなかった 8 月 2 日のプロビジョニング要求を予測できているが，提案手法で予測に成功している 8 月 3 日のプロビジョニング要求は予測できなかった．8 月 2 日と 3 日の app2 の CPU 利用率変化を見ると，8 月 2 日は定常的に負荷が高い状態にあったのに対し，8 月 3 日には突発的な負荷上昇が発生している（**図 6** 参照）．急激な負荷の上昇を予測する際には 2 段階しきい値法よりも提案手法の方が適していると判断できる．

一方，時系列解析法ではプロビジョニング要求の予測に成功することはなかった．自己回帰モデルでは直近の CPU 利用率の値の線形結合で将来の CPU の値を予測するため，急激に CPU 利用率が増加するような状況の予測には向かない．AR モデルのパラメータを変更した場合や，AR モデルの拡張である ARMA モデルや ARIMA モデルを利用した場合でも，入力とする値の範囲を大きく超える将来の値を予測することはできないため，プロビジョニング要求の予測に利用することは難しい．

表 7　スタンバイ状態決定頻度の比較
Table 7　Frequencies of standby state decision by different prediction methods.

予測手法	予測成功回数	スタンバイ状態決定頻度 (%)
Model-20	4	0.0157
Model-20'	4	0.0190
thresh-29	2	0.0085
thresh-28	2	0.0204
thresh-27	3	0.0340
thresh-26	3	0.0423
thresh-25	5	0.0517
ar-20	0	0.0007

提案手法は CPU 利用率以外の情報を予測モデルの入力として扱える点にも特長がある．5.4 節の評価結果より，曜日や時間帯の情報を用いて予測の性能を向上させることが可能で

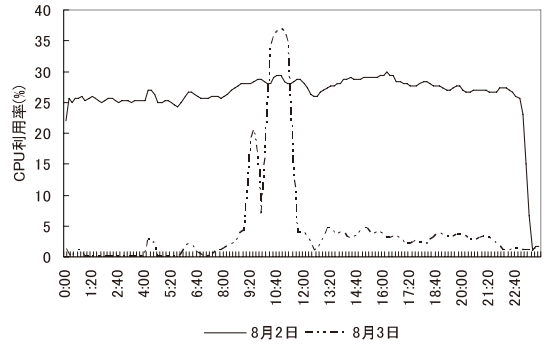

図 6　app2 の CPU 利用率変化
Fig. 6　CPU utilizations of app2.

ある．2 段階しきい値法や時系列解析法では，過去の CPU 利用率の情報のみに基づいて予測を行うため，急激に CPU 利用率が増加する状況を事前に予測することは難しい．提案手法は CPU 利用率に大きな変化が見られない場合でも，アプリケーションのクライアント接続数やディスク利用率，曜日や時間帯などの情報から，CPU 利用率が急増する状況の前兆を検出できる可能性がある．

6. ま と め

ディペンダブルで高効率なデータセンタを実現するためには，サーバを共有するだけでなく，必要なときに迅速にサーバの構成を変えるプロビジョニング技術が必要である．本論文では VM を利用して共有サーバ上にスタンバイ領域を作成し，スタンバイ領域でサーバのプロビジョニングを投機的に実行することでプロビジョニングを高速化する手法を提案した．プロビジョニング要求が発生する前に適切なスタンバイ状態を準備するためには，プロビジョニング要求を事前に予測する必要がある．プロビジョニングはシステムの異常状態時に要求されるため，システムの状態を示す様々な負荷変動データを入力とし，ロジスティック回帰分析によって将来の異常状態を予測することで，プロビジョニング要求を予測する手法を提案した．企業における社内システムの負荷変動データを用いて予測手法を評価した結果，20 分先読み予測モデルによって 50%のプロビジョニング要求を予測できることが確かめられた．プロビジョニング要求が的中した場合は 10 分から 20 分のプロビジョニング高速化が実現でき，プロビジョニング要求に対して迅速な対応が可能となる．

参 考 文 献

1) Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P. and Neugebauer, R.: Xen and the Art of Virtualization, *19th ACM Symposium on Operating Systems Principles* (*SOSP19*) (2003).
2) Dinda, P. and O'Hallaron, D.: Host Load Prediction Using Linear Models, *Cluster Computing*, Vol.3, No.4 (2000).
3) Das, G., Lin, K., Mannila, H., Renganathan, G. and Smyth, P.: Rule discovery from time series, *Proc. 3rd International Conference of Knowledge Discovery and Data Mining* (*KDD*), pp.16–22 (1998).
4) Hosmer, D.W. and Lemeshow, S.: *Applied logistic regression*, *2nd ed.*, Wiley (2000).
5) Holmes, G., Donkin, A. and Witten, I.H.: Weka: A machine learning workbench, *Proc. 2nd Australia and New Zealand Conference on Intelligent Information Systems* (1994).
6) Jiang, G., Chen, H. and Yoshihira, K.: Discovering likely invariants of distributed transaction systems for autonomic system management, *Proc. 3rd IEEE International Conference on Autonomic Computing* (*ICAC2006*), pp.199–208 (2006).
7) Landwehr, N., Hall, M. and Frank, E.: Logistic model trees, *Proc. 14th European Conference on Machine Learning* (*ECML*), pp.241–252 (2003).
8) 小倉章嗣，荒木拓也：ユーティリティグリッドにおける計算機設定時間短縮のための状態管理手法，第 5 回情報科学技術フォーラム（FIT2006），L-023 (2006).
9) Data Center Automation in a New Light, Symantec article library (2007). http://www.symantec.com/enterprise/library/
10) WebSAM SigmaSystemCenter. http://www.ace.comp.nec.co.jp/SigmaSystemCenter/
11) IBM Provisioning Manager. http://www-306.ibm.com/software/tivoli/products/prov-mgr/
12) launchd. http://developer.apple.com/macosx/launchd.html
13) InitNG. http://www.initng.org/
14) Glassfish. https://glassfish.dev.java.net/
15) VMware. http://www.vmware.com/
16) WEKA. http://www.cs.waikato.ac.nz/ml/weka/

(平成 19 年 9 月 28 日受付)
(平成 20 年 3 月 4 日採録)

町田　文雄（正会員）

1977 年生．2001 年東京工業大学工学部情報工学科卒業．2003 年同大学院情報理工学研究科計算工学専攻修了．同年日本電気株式会社入社．現在，NEC サービスプラットフォーム研究所にて自律コンピューティング，大規模システム運用管理の研究に従事．2007 年 IARIA ICAS Best Paper Award 受賞．

川戸　正裕（正会員）

1975 年生．1998 年東京工業大学工学部電気電子工学科卒業．2000 年同大学院情報理工学研究科計算工学専攻修了．同年日本電気株式会社入社．以来，分散コンピューティングの研究に従事．電子情報通信学会会員．

前野　義晴

1967 年生．東京大学にて学士，修士（理学，物理学専攻）．筑波大学にて博士（システムズ・マネジメント，企業科学専攻）．日本電気株式会社勤務．専門は，ネットワーク・コンピューティング，システム・デザイン．インターネットや分散コンピューティングシステムに見られる複雑な挙動や非線形的な現象の起源を明らかにし，システムのデザインのための新しい方法論を切り拓くことを目指している．電子情報通信学会学術奨励賞等受賞．INSNA，APS，IEEE 各会員．