情報処理学会論文誌 Vol.56 No.2 503–515 (Feb. 2015)

仮想化機構を利用する大規模サービス基盤のためのストレージ設定高速化機構

中島 淳[1,a)] 名倉 正剛[1,b)] 柴山 司[1,c)] 坂下 幸徳[1,d)]

受付日 2014年5月14日, 採録日 2014年11月10日

概要：サーバ仮想化機構の普及により，多数のサーバを利用した大規模サービスの提供が容易になっている．サーバ仮想化機構を利用して大規模サービス基盤を構築する場合，大量の仮想マシン（VM）を迅速に提供できないと，サービス開始が遅延する．大規模ネットワークサービス提供のためのサービス基盤を構築する場合を想定し VM 提供に要する時間を分析したところ，VM に利用するストレージの設定に多くの時間を要する場合があった．そこで本研究では，複数の VM を提供する際のストレージ設定を一括処理することで，ボリューム作成やボリューム割当てに要する処理を削減し，大量の VM の提供に要する時間を短縮するための手法を提案する．評価実験を行った結果，提案手法によって複数の VM を一度に提供する際の設定処理に要する時間を，大幅に短縮できることを確認した．

キーワード：大規模サービス基盤構築，ストレージ設定高速化，バルク処理

Storage Provisioning Acceleration for Large-scale Service Platform Using Server Virtualization Mechanisms

Jun Nakajima[1,a)] Masataka Nagura[1,b)] Tsukasa Shibayama[1,c)]
Yukinori Sakashita[1,d)]

Received: May 14, 2014, Accepted: November 10, 2014

Abstract: Recently, the server virtualization mechanism becomes widely used. As the result, the providing of the large-scale services with many servers becomes easier. When providing the large-scale service by using the mechanism, start of the service will be delay unless a lot of VMs are provided quickly. From the results of the analysis on the time spent providing VMs under the assumption of configuring the service platform for providing large-scale network service, we found that the configuration of storage which is used by VMs often take a long time. In this work, we propose a method for reducing the time spent providing a lot of VMs. By collectively dealing with the storage setting required to provide multiple VMs, our method reduces the time of the volume provisioning and volume assignment processing. We confirmed in the evaluation that our method reduces dramatically the time spent in configuring when providing multiple VMs.

Keywords: provisioning for large-scale service platform, fast storage setting, bulk processing

1. はじめに

近年，ネットワークサービスの利用形態は，ネットワークに接続された多数のサーバや携帯端末が有機的に連携することにより，1 つのサービスを形成するように変化している．サービスの規模も年々拡大し，それを運用するために必要なサーバ台数も増加している [1]．

一般的に物理ハードウェア資源を大量に用意することは困難である．そこで，限られたハードウェア資源で大規模サービスを運用するために，サーバ仮想化機構が利用されることが多くなってきている．サーバ仮想化機構を利用す

[1] 株式会社日立製作所横浜研究所
Yokohama Research Laboratory, Hitachi, Ltd., Yokohama, Kanagawa 244–0817, Japan
[a)] jun.nakajima.vu@hitachi.com
[b)] masataka.nagura.qj@hitachi.com
[c)] tsukasa.shibayama.qx@hitachi.com
[d)] yukinori.sakashita.hk@hitachi.com

ることで，少数の物理サーバ上に多数の仮想マシン（VM）を用意し，VM 上に大規模サービスに必要なサーバを用意することが可能である．このように，ネットワークサービスの利用形態だけでなく，運用形態も変化している．

サーバ仮想化機構を利用して構築されたサーバは，ハードウェアリソースに対してホスト OS やハイパーバイザを介してアクセスする．ストレージ装置に対しては，ホスト OS やハイパーバイザの提供するファイルシステム上に用意した仮想ストレージボリューム用のディスクイメージファイルを介してアクセスする．このため，仮想化機構を利用しないサーバに比べ，一般的にパフォーマンスが低下する．この状況を解決するため，仮想化機構やそれを利用するクラウド基盤ソフトウェアでは，ストレージ命令として標準的な命令セットを利用することで，ストレージに直接アクセスする方式が登場している [2], [3]．これらを利用し，ストレージ領域に対してファイル単位ではなく，ボリューム単位でアクセスすることで，パフォーマンスの低下を防ぐことができる．なお，このような方法を用意していない仮想化機構（Microsoft Hyper-V や，Xen など）でも，ファイルアクセスの衝突によるパフォーマンスの低下を防ぐために，1 つのボリュームに 1 つのディスクイメージファイルのみを配置する運用が行われることも多い．

これらの方法により，ストレージアクセス性能が向上するが，構築段階では個々の VM が利用する仮想ストレージボリュームの個数分だけ，ボリュームをストレージに用意する必要がある．大量の VM を利用する大規模サービスを構築する場合，大量のボリュームを用意する必要があり，個々のボリューム構築のための命令を発行するため，ストレージ設定時間が長時間化する．ストレージ装置を提供するベンダによっては，大量のボリュームを一括設定するためのバルク処理命令を用意し，独自の管理ソフトウェアを利用することで，短時間にストレージを設定できるようにしているものもある．バルク処理命令を利用することにより，1 つのコマンドに複数の命令セットに対するパラメータを記述し，一括してストレージ装置に命令を実行させることが可能である．ストレージ装置での処理は個々の実装に依存するが，前処理，後処理に含まれる処理を一度に実施したり，実際の装置に対する物理的処理にともなうロック取得などの処理を効率化したりすることで，ストレージ装置に対する物理的な処理に起因するボトルネックを改善し，高速化している．

しかし，それらの処理命令はストレージ管理標準（SMI-S: Storage Management Initiative - Specification）[4] のような統一規格によって標準化されていないため，仮想化システムの管理に利用できない．仮想化機構やそれを管理するための仮想化管理ソフトウェア（VMware に対する VMware vCenter，Hyper-V に対する Microsoft SystemCenter など）は，特定ベンダのストレージ命令に依存せずに，標準化されたストレージ命令セットしか利用しない．このため現状ではベンダ独自のバルク処理命令を利用できない．仮想化管理ソフトウェアから利用するストレージ装置を限定すれば，ベンダ独自の命令セットに対応できる．しかし，仮想化機構から利用するストレージ装置の種類を，特定ベンダの特定のストレージ装置に限定することは現実的ではない．さらにバルク処理命令は，ストレージベンダにとってストレージ装置の他社製品に対する主要な差別化要因である応答性能の改善を目的としており，各ベンダとも進歩が著しい．このため，仮に標準化規格も含めバルク処理に対応したとしても，差別化にともないより効率的な処理を実現するために命令セットをさらに追加・変更することは容易に起こりうる．そのような場合に，仮想化管理ソフトウェアからは従来からの応答性能の低い命令セットしか利用できなくなる．このように，仮想化管理ソフトウェアから実施されるストレージ設定を高速化することは容易ではなく，結果として VM 提供処理が長時間化する．

本研究では，まず仮想化管理ソフトウェアから VM 提供を実施する場合に要する処理時間を分析した．その結果，前述のように VM 提供処理に長い時間を要し，ストレージの設定処理がボトルネックとなっていることを確認した．そこで我々は，仮想化管理ソフトウェアからストレージ管理を実施する際に，ストレージ設定を高速化させる機構を提案する．提案手法を適用することにより，ストレージ装置ベンダ独自のバルク処理命令に対応できない仮想化管理ソフトウェアを改造することなくバルク処理命令を活用できるようになった．そして評価実験の結果，複数 VM を一度に作成する際の設定処理時間を，大幅に短縮できることを確認した．

以降，まず 2 章で仮想化管理ソフトウェアを利用した VM 提供に要する処理時間を分析し，大規模 IT システム構築の問題点を述べる．そして 3 章で提案方式について述べ，4 章で評価実験を行う．5 章で考察を行い，6 章で関連研究をあげ，7 章で結論を述べる．

2. 大規模サービス基盤構築時の VM 提供手順と構築時間に関する課題

2.1 大規模サービス基盤構築時の VM 提供手順

我々は，北米のパートナ企業が 5 つの大規模クラウドデータセンタを対象に実施した運用管理形態についてのヒアリング結果を利用し，企業向け大規模サービスでの VM 提供手順を調査した．ヒアリング対象のデータセンタでは，1,000〜10,000 程度の VM により企業向けサービスを提供していた．それらのサービスは，人事・給与システムや，勤務管理システムなどの社内向け業務システムや，販売促進サイトを運営する Web サイトのような社外向け業務システムであった．サーバ規模は数十台から数百台と差があったが，共通して VM 上に構築したサーバのディスク

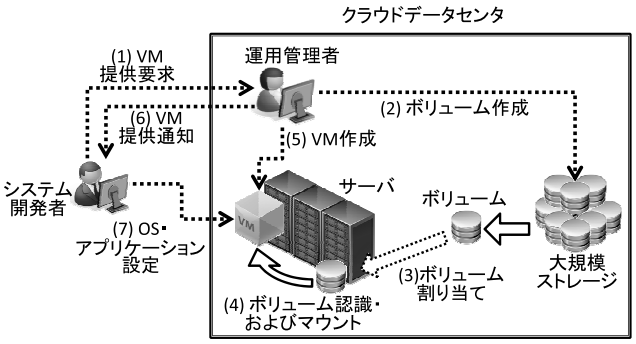


図 1 仮想化機構を利用したサービス基盤構築手順
Fig. 1 Overview of service platform provisioning with virtualization mechanism.

イメージを，大規模ストレージで一括管理していた．そして，このように大規模ストレージを利用してデータを一元管理することで，可用性の確保やバックアップの容易化と，装置リソースの共用によるコスト削減を図っていた．

システム開発者がデータセンタの運用管理者に VM の提供を要求してから，運用管理者が VM を提供し，サービスに必要な業務システムをシステム開発者が提供された VM に構築するまでの手順を，次に示す（**図 1**）．なお説明の都合上，本論文の以降の説明では，ホスト OS を利用せずハイパーバイザを利用する仮想化機構の場合を想定するが，ホスト OS を利用する仮想化機構の場合も同様である．

手順 1（VM 提供要求）：システム開発者が運用管理者に VM の提供を要求する．

手順 2（ボリューム作成）：運用管理者がデータ格納領域（ボリューム）を作成し，フォーマットする．

手順 3（ボリューム割当て）：ストレージ設定を変更し，VM を作成する対象のサーバの仮想化 OS（ハイパーバイザ）に，ボリュームを割り当てる．

手順 4（ボリューム認識・マウント）：作成したボリュームをハイパーバイザに認識させる．前述のようにストレージ領域に対してボリューム単位でアクセスできない仮想化機構の場合は，認識したボリュームをマウントし，その際にファイルシステムを作成する．

手順 5（VM 作成）：認識したボリュームを利用し，VM を作成する．ボリューム単位でアクセスできない仮想化機構の場合は，手順 4 で作成したファイルシステム上にディスクイメージファイルを作成し，そのファイルを利用して VM を作成する．

手順 6（VM 提供通知）：作成した VM を，運用管理者からシステム開発者に提供する．

手順 7（OS・アプリケーション設定）：システム開発者が，VM 上に OS やアプリケーションをインストールし，システムを構築する．

2.2 大規模サービス基盤構築時の課題

業務システムを構築する場合，システム開発者はまずシステム構成を決定し，必要なサーバ台数を見積もる．そして，予算上問題ないことを確認したうえ，データセンタの運用管理者に VM 提供を依頼する．このため，前述の調査結果のような数百台規模のサーバを利用する大規模サービス基盤を構築する場合は，運用管理者へ多数の VM 提供の依頼が一括して行われることが多い．

このような状況で VM 提供に時間を要すると，サービスインが遅延することになる．このため運用管理者には，システム開発者からの VM 提供の要求からできるだけ短時間で VM を提供することが求められる．自動プロビジョニングを実施する製品群を利用することで，VM 提供手順を自動化させオペレーションに起因するロスタイムの発生を防ぐことができる [5], [6]．しかし本質的には，2.1 節の手順 2 から手順 5 の処理自体を迅速に実施する必要がある．

そこで，予備実験としてシステム規模を変化させて，VM 提供の各手順に要する時間を調べた．

【予備実験：システム規模と提供時間の関係の測定】

まずストレージに接続されたサーバ上に仮想化機構として Microsoft Hyper-V をインストールした．実際のデータセンタ運用管理では，運用管理者はオペレーションルームで管理用コンソールを備えた PC を利用し，サーバやストレージを設定する．このため，次にネットワーク接続された管理用 PC を用意した．管理用 PC には，Hyper-V の管理ソフトウェアである Microsoft SystemCenter をインストールした．

そして管理用 PC から，管理コンソールを利用してボリューム作成（2.1 節の手順 2）やボリューム割当て（手順 3）を実施した．なお，この際前述のように，既存の管理コンソールからはバルク処理命令を利用せず，標準的なストレージ命令セットのみを利用した．次に同様に同じ管理用 PC から管理コンソールを利用して，割り当てられたボリュームをハイパーバイザに認識させた．Hyper-V はボリューム単位ではストレージ領域にアクセスできないため，ハイパーバイザ上にマウントし，ファイルシステムを作成した（手順 4）．そしてそのファイルシステム上にディスクイメージファイルを作成し，それを利用して VM を作成した（手順 5）．これらの手順に要する時間を測定した．複数の VM を対象に測定する場合は，これらの手順を繰り返した．なお 1 章や 2.1 節に記載したように，一般的な企業向けサービスでは，パフォーマンス低下を防ぐため，各 VM からボリューム単位でストレージにアクセスする．ここでは測定環境に Hyper-V を利用したため，複数の VM を作成する際に，VM ごとにボリュームを確保した．そして，それぞれのボリューム上に作成対象の VM が利用する 1 つのディスクイメージファイルを作成した．VM 数を 1 台から 100 台まで

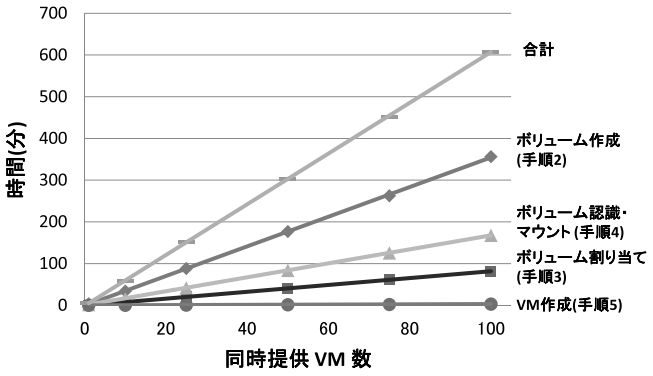


図 2 VM 数と提供に要する時間の関係
Fig. 2 Provisioning time by number of VMs.

変化させて測定し，各手順の合計値を算出した．

サーバには CPU が Intel Xeon X3363 2.83 GHz でメモリを 8.0 GB 搭載した PC を，管理用 PC には CPU が Intel Core2 Duo 2.93 GHz でメモリを 4 GB 搭載した PC を利用した．ストレージ装置には，SAS ディスク（15,000 rpm，2.5 インチディスク）を搭載しており，7 GB のキャッシュを備えたものを利用し，一般的にデータセンタで利用される RAID5 構成を設定した．さらに，サーバ用 OS には Windows 2008 R2 Datacenter を，管理用 PC の OS には Windows 7 Professional を用いた．そして，VM 1 台あたりに 250 GB のボリュームを作成し割り当てた．サーバとストレージの間は 2 Gbps のファイバチャネルによって接続し，それらの機器と管理用 PC との間は 100 Mbps のイーサネットによって接続した．

測定結果を，**図 2** に示す．

図 2 より，VM 提供に要する時間は，すべての手順で VM 数に比例して増加することが分かる．100 VM から構成されるシステムを構築する際には VM の提供に 600 分もの時間を要した．多数のサーバから構成される大規模 IT システムを構築する際に，運用管理者がシステム開発者に多数の VM を迅速に提供することが困難であるといえる．

各処理に要する時間を比較すると，まずストレージ設定操作（手順 2 のボリューム作成と手順 3 のボリューム割当て）に多くの時間を要する．また，それらほどではないが，ハイパーバイザ上での設定操作の一部（手順 4 のボリューム認識・マウント）にも時間を要している．一方でハイパーバイザ上での VM 作成（手順 5）については，VM 数により増加するが，それほど多くの時間を要していない．

予備実験では，実験環境準備の都合上，作成したボリュームをすべて同一のハイパーバイザで仮想ディスクボリュームとしてマウントしたうえで，そのボリュームに VM を作成した．100 VM を提供するために，100 ボリュームを単一のハイパーバイザへマウントした．手順 4 については，このマウント操作がシーケンシャルに実施されたため，処理時間が線形に増加した．しかし現在普及している CPU コア数や，最大同時実行内部命令数が数個であること考えると，通常は同一ハイパーバイザに作成されるのはたかだか数 VM 程度である．したがって 100VM は数十個のハイパーバイザ上に作成され，このようにボリュームマウント操作に長時間を要することは考えにくい．図 2 の合計からこのような実験環境の影響で増加した手順 4 や手順 5 の時間を取り除くと，100 VM から構成されるシステム構築時の VM の提供に，約 420 分（約 7 時間）の時間を要した．

一方で 2.1 節で述べたように，大規模サービスを提供するデータセンタでは，大規模ストレージ上に VM 用ボリュームを作成していた．このため，ストレージの装置台数はそれほど多くならない．したがって，少数のストレージに多数のストレージ設定が集中し，予備実験の結果に示されるようにストレージボリューム作成とボリューム割当ての処理が長時間化する．大規模サービスの構築時間を短縮するには，この操作に要する時間を短縮する必要がある．

3. 大規模サービス基盤のためのストレージ設定の高速化機構

3.1 概要

2.2 節で述べたように，サーバ仮想化機構を利用し大規模 IT システムを構築する際には，ストレージ設定が長時間化する．そこで本研究ではストレージ設定を高速化することで，大規模サービス基盤を利用した IT システム構築を高速化する手法を提案する．

ストレージ装置では，ディスクに対する読み出しや書き込み時間に対して，シーク時間や回転待ち時間のオーバヘッドが大きい．このため，一般的にバルクアクセスでアクセス処理を高速化できる．実際にこのようなストレージ装置の特性を利用して，バルクアクセスを利用してデータベースマネージャによる DB アクセスを高速化する研究も存在する [8], [9]．ストレージ管理操作を実施する場合にも同様の特性を示すため，ストレージ装置によっては，設定処理に対するバルク処理命令セットを備えている場合がある．しかし，1 章で述べたようにそれらはベンダ独自に実装されており，一括設定によりストレージ管理を高速化するためには，各ストレージベンダにより提供される専用の管理クライアントソフトウェアを利用する必要がある．SMI-S [4] のような統一規格によって標準化されていないため，仮想化機構を管理するための標準的な仮想化管理ソフトウェアからは利用できない．このため 2.2 節で実施した予備実験では，大量のストレージ設定命令の実行により，大規模サービス基盤の構築が長時間化していた．

そこで我々は，仮想化管理ソフトウェアから送信されるストレージ設定リクエストをキューイングし，一括して設定することにより，ストレージ設定を高速化する手法を提案する．この手法を利用することで，バルク処理命令に対

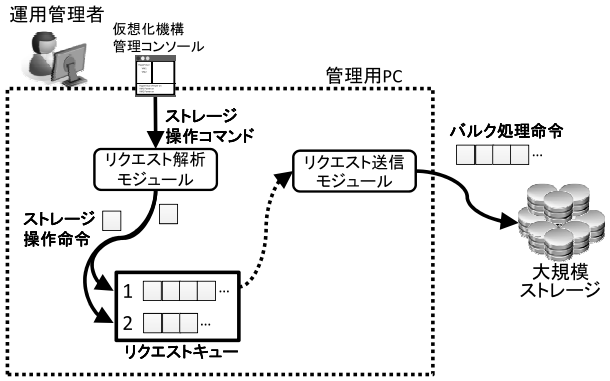


図 3 提案手法の概要
Fig. 3 Overview of our method.

表 1 リクエスト解析モジュールが受け取る SMI-S 標準命令一覧
Table 1 List of SMI-S standardized methods accepted by the request analysis module.

#	名称	概要
1	CreateOrModifyStoragePool	ストレージプール*1作成
2	DeleteStoragePool	ストレージプール削除
3	CreateOrModifyElementFromStoragePool	ストレージプールからのボリューム作成
-	CreateOrModifyElementFromElements	ディスク領域からのボリューム作成（未実装）*2
4	ReturnToStoragePool	ボリューム削除
5	ExposePaths	ボリューム割当て
6	HidePaths	ボリューム割当て解除

応できない仮想化管理ソフトウェアを改造することなくストレージ装置のバルク処理命令を活用できるようになる．

提案手法の構成とデータの流れを，**図 3** に示す．提案手法では管理用 PC にリクエスト解析モジュールとリクエスト送信モジュールを用意する．従来ではサーバ仮想化機構の管理コンソールから，運用管理者が指示したストレージ設定命令が直接ストレージ装置へ送信されていた．提案手法では，管理コンソールから発行された設定リクエストを，ストレージ装置の代わりに提案手法で提供されるモジュール群が受け取り，バルク処理命令に変換してストレージ装置に送信する．まずリクエスト解析モジュールが，前述の SMI-S 標準で定められたリクエスト形式の設定リクエストを受け取る．提案手法では，この標準仕様に定義済みとして記載の命令セット 22 個のうちの設定用の全 7 命令から，機能が重複し代替可能な 1 命令を除いた 6 命令を受け取るように実装した（**表 1**）．

リクエスト解析モジュールはバルク処理として一括実行できる命令ごとにリクエストキューに分類して蓄積する．バルク処理可能なリクエストの種類はストレージ装置によって異なる．提案手法ではあらかじめ 3.2 節に後述の分類ルールにより指定された方法に従い，バルク処理可能な命令であれば，リクエストを分類する．リクエスト送信モジュールは，一定時間経過ごとにリクエストキューに蓄積された操作命令を取り出し，ストレージ装置の備える一括設定のバルク処理命令形式に変換し，ストレージ装置に対して命令を送信する．ストレージはこのバルク処理命令を実施し，リクエスト送信モジュールに実行結果を一括して返す．リクエスト送信モジュールは，受け取った実行結果を，入力として受け取った個々のストレージ設定命令に対応した応答メッセージとして返却する．これにより，ストレージ設定処理が大量に実施される場合でも，ストレージ装置独自に用意されたバルク処理命令を活用することで処理を高速化でき，サーバ仮想化機構を利用した大規模サービス基盤の構築時間を短縮する．この際に，仮想化管理ソフトウェアからのリクエスト送信先を提案手法のリクエスト解析モジュールへ変更するだけで，仮想化管理ソフトウェアに対するソフトウェア変更なく提案手法を利用できるようにした．

提案手法を利用して実施する，ストレージのボリューム作成からボリューム割当てまでの処理の流れの一例を，**図 4** に示す*3．運用管理者が管理コンソールを利用してボリューム作成（表 1 #3 の CreateOrModifyElementFromStoragePool 命令）をストレージに対して実施し，作成したボリュームに対してサーバへの割当て（表 1 #5 の ExposePaths 命令）を実施している．以降ではまずストレージ装置の振舞いに合わせて記述する分類ルールについて 3.2 節に記載する．そしてこの処理の流れのうち，リクエスト解析モジュールによるリクエストの分類処理について 3.3 節に，リクエスト送信モジュールによるリクエスト送信処理を 3.4 節に述べる．

3.2 分類ルール

分類ルールは，対象のストレージ装置の振舞いと運用ポリシに合わせて，あらかじめ記述される．ルールには，ストレージ装置のコマンド群に対して，互いに同時実行できないコマンドセットのリスト，一括実行できるコマンドリストと一括実行に関連する属性，各コマンドを実行する際の実行前状態と実行後状態を記述する．

分類ルールの一例を次に示す．

```
1 group exclusive { // 同じ装置に対して同時実行できないコマンドのグループ
2   model Storage_A; // ストレージモデル名
3   serial 1_00; // ファーム番号
4   commandset // 同時実行できないコマンドセット
5   {
6     CreateOrModifyElementFromStoragePool;
```

*1 複数台の物理ディスクを 1 つの領域として認識させるために，ストレージ機能により定義される仮想的なディスク領域．
*2 #1 によって物理ディスクからストレージプールを作成し，#3 でストレージプールからボリュームを作成すれば代替可能のため，実装せず．
*3 この図では，簡単化のために，単一の管理コンソールから同時に 2 つのリクエストが発生する際の流れを記述している．大規模サービス基盤構築の際には，さらに多数の運用管理者から，同時に多数のリクエストが送信される場合もある．

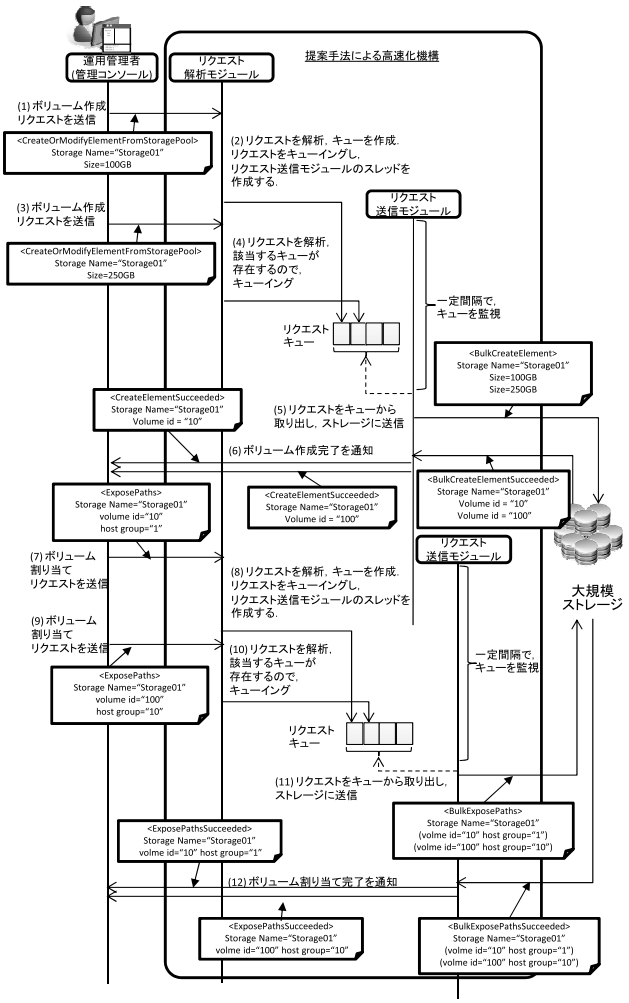


図 4 提案手法の処理の流れ

Fig. 4 Execution flow of proposed method.

```
7       ReturnToStoragePool;
8     };
9     commandset // 同時実行できないコマンドセット
10    {
11      ExposePaths;
12      HidePaths;
13    };
14  }
15
16  group bulk_support { // 一括実行可能なグループ
17    model Storage_A; // ストレージモデル名
18    serial 1_00; // ファーム番号
19    max_commands 100; // 実行可能なコマンド数
20    waittime 20; // 実行処理待ち時間
21    commandlist // 実行可能なコマンドを記述
22    {
23      CreateOrModifyElementFromStoragePool;
24      ExposePaths;
25      ReturnToStoragePool;
26      HidePaths;
27    };
28  }
29
30  command CreateOrModifyElementFromStoragePool {
31    target pool; // コマンド実行対象オブジェクト
32    target volume // コマンド実行対象オブジェクト
33    {
34      precondition none; // 実行前状態
35      postcondition created; // 実行後状態
36    };
37  }
38
39  command ExposePaths {
40    target volume // コマンド実行対象オブジェクト
41    {
42      precondition created; // 実行前状態
43      postcondition allocated; // 実行後状態
44    };
45  }
46
47  // 以降, ReturnToStoragePool, HidePaths コマンドに
        ついても同様に記述, ここでは省略
```

この例では，同じ装置に対して同時実行できないコマンドのグループ（exclusive：1 行目～14 行目）と，一括実行可能なコマンドのリストを表すグループ（bulk_support：16 行目～28 行目）と，各コマンド実行の実行前，実行後状態を記述している．複数コマンドの同時実行や，単一のコマンドのバルク実行などの仕様はストレージ装置やファームウェアのバージョンにより異なる．このため，exclusive グループや bulk_support グループには，対象のストレージモデル名とファームウェアのバージョンを記述している．

exclusive グループには，互いに同時実行できないコマンドの組合せを commandset 内に記述する．この例では，ストレージプールからのボリューム作成，削除処理である CreateOrModifyElementFromStoragePool コマンドと ReturnToStoragePool コマンド，およびボリューム割当て，解除処理である ExposePaths コマンドと HidePaths コマンドを，それぞれ互いに同時実行できないコマンドのセットとして定義している．ストレージ装置では同じリソースに対して，このように相反する設定操作を同時実行できない場合が多い．commandset 内に記述することで，いずれかのコマンドが実行されている際に，他のコマンドを実施できないようにしている．

bulk_support グループには，ストレージ装置にバルク処理命令が用意されているコマンドを，一括実行可能なコマンドとして commandlist 内に列挙する．この例では，対象のストレージ装置に CreateOrModifyElementFromStoragePool コマンド，ReturnToStoragePool コマンド，ExposePaths コマンド，HidePaths コマンドについてのバルク処理命令が用意されていることを記述している．さらに，バルク実行可能なコマンド数（max_commands）と，リクエストキューから命令を取り出してストレージ装置に対して命令を実行する間隔（waittime）を記述している．

これらは，commandlist 内に列挙されたそれぞれのコマンドに対して，最大 10 個のコマンドをバルク実行可能であり，最初のコマンドの受信後 20 秒間に受信したコマンドを，1 つのバルク処理に変換することを表している．

そして各コマンド実行について，対象オブジェクトと，オブジェクトに対して状態変化をともなう場合は，実行前状態，実行後状態を記述している．たとえば，30 行目～37 行目では，ストレージプールからのボリューム作成処理（CreateOrModifyElementFromStoragePool コマンド）が，ストレージプール（pool）とストレージボリューム（volume）を対象に実行され，ストレージボリュームについては，ボリュームが存在しない状態（none）から，作成済みの状態（created）に変化することを示している．さらに 39 行目～45 行目では，ボリューム割当て処理（ExposePaths コマンド）を示しており，このコマンドの実行には，ストレージボリュームが作成済み（created）である必要があり，実行後に割当て済み（allocated）に変化することを示している．したがってボリューム割当て処理は必ずストレージプールからのボリューム作成処理の後に実行される必要があることが示されている．このように，同一オブジェクトに対する命令セットで処理の順序関係がある場合に，その関係を記述できる．

3.3 リクエスト分類処理

リクエスト解析モジュールは，運用管理者が管理コンソールから入力したストレージ装置への設定リクエストを受信し，分類ルールに従って一括実行できる設定リクエストごとにリクエストキューに分類する．

3.2 節で例示した分類ルールに従うと，2.2 節の予備実験で時間を要したボリューム作成とボリューム割当ての処理について，bulk_support グループに記載されているため，バルク処理を実現できる．図 4 では，それらのコマンドを受け取った場合に，分類ルールに従ってリクエスト種類ごとに分類する場合の処理の流れを記載している．なおリクエスト種類が異なると，前処理，後処理に含まれる処理も異なることが多いため，ストレージ装置の実装ではリクエスト種類ごとにバルク処理命令を用意しているものが多い．そのためこの例では，操作対象のストレージ装置ごとに，リクエストの種類によって分類している．その結果，運用管理者が管理コンソールから送信した Storage01 に対するボリューム作成リクエスト (1) と (3)，および Storage01 が作成したボリュームに対する割当てリクエスト (7) と (9) をバルク処理可能なリクエストと見なし，リクエストキューに分類する．なお，ストレージ装置の識別名とリクエストの種類の一致するリクエストキューが存在しない場合は，新たにリクエストキューを作成しそこに分類し，そのキューを監視するためのリクエスト送信モジュールのスレッドを生成する．この例では，(1) と (7) は，それぞれのリクエスト種類ごとの最初に到着したリクエストであり，(2) と (8) の手順でリクエストを解析した結果キューを作成する．そして，(3) と (9) で受信したメッセージについては，(4) と (10) で対応するキューに分類する．

なお分類ルールに一括実行できる命令として記述されていない命令の場合は，リクエストキューに分類せず，従来と同様にリクエスト送信モジュールに直接送信する．

3.4 リクエスト送信処理

リクエスト送信モジュールのスレッドは，割り当てられたキューから分類ルールに記述されたバルク処理待ち時間ごとにリクエストを取り出し，集約してストレージにバルク処理として送信する．

リクエスト集約の概要を，**図 5** に示す．この図は，ボリューム割当てのリクエストを集約する際の，元になる設定リクエストと，それに対応する集約後の設定リクエストを示している．この図の例では，“host group” で示されるボリューム割当て対象のサーバが異なる複数の設定リクエストを集約している．

そしてリクエスト送信モジュールは，集約したリクエストをストレージに送信する．その際に，ストレージ装置のバルク処理のための命令セットを利用する．リクエスト送信モジュールは，ストレージ装置からの応答を，管理コンソールから受信した個々の命令への応答メッセージに変換して返却する．図 4 の例では，(5) と (11) によってキューを監視して，リクエストを取り出し，バルク処理命令としてストレージに送信している．そして (6) と (12) によって，それぞれの処理の完了を通知している．

なお，この例では運用管理者の操作する管理コンソールにボリューム作成リクエストの完了通知が到着したのち，ボリューム割当てリクエストが送信されている．しかし，ストレージ装置によっては非同期でボリューム作成を実施

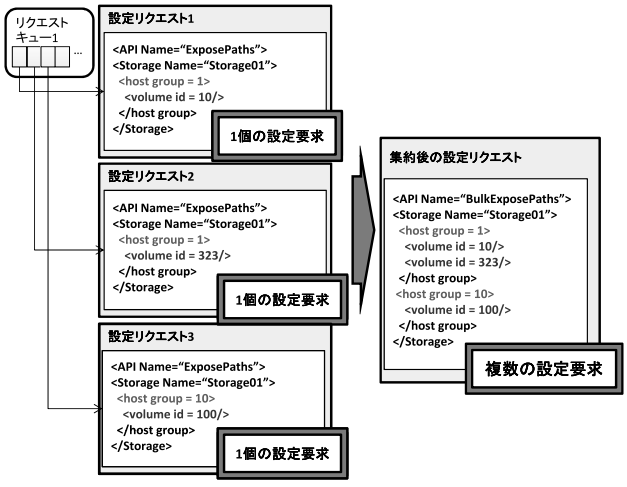


図 5 リクエスト集約の概要
Fig. 5 Request combining.

する場合もあり，その場合にはボリューム作成が実際に完了する前にストレージ装置が応答を返すため，実際には作成されていないボリュームに対してのボリューム割当てリクエストを受信する場合がある．そこで 3.2 節で例示した分類ルールでは，ボリューム割当て命令を実施するための実行前状態について定義している．この定義によって割当て対象のボリュームの状態が作成済み（created）になっていないボリュームを対象にした命令を除いた命令を，(11) でストレージ装置に送信する．

この処理の流れの終了後，運用管理者は割り当てられたボリュームを利用して，VM を作成する．提案手法によりストレージ設定処理に要する時間が短縮されるため，VM を迅速に提供できるようになる．

4. 評価実験

4.1 実験 1：VM 構築時間の短縮効果の測定

提案手法の効果を評価するため，2.2 節の予備実験と同一環境で，提案手法の処理時間を測定した．

まず，2.2 節での予備実験と同様に，Hyper-V を利用した管理用 PC から，管理コンソール経由でボリューム作成（2.1 節の手順 2）とボリューム割当て（手順 3）を実施した．そして同じ管理用 PC から SystemCenter のコンソールを利用し割り当てられたボリュームをマウントし（手順 4），そのボリューム上で VM 作成（手順 5）を実施した．この手順 2 と手順 3 において提案手法を利用し，設定リクエストをキューイングしたうえで設定を行った．

作成する VM 数を 1 台から 100 台まで変化させ，設定リクエストをリクエスト解析モジュールに VM 数分だけ連続的に発行した．そしてそれぞれの手順に要する時間を計測した．なお，分類ルールとして 3.2 節の例と同様に，ボリューム作成とボリューム割当てのリクエストを一括処理可能として設定した．そして，同時実行命令最大数（max_commands）には 100 を指定した．これらは，設定対象のストレージ装置の仕様を基に決定した．また，SystemCenter コンソールからのコマンド実行のタイムアウト時間は 5 分で，事前実験の結果から 1 つのボリューム作成処理に約 230 秒，ボリューム割当て処理に約 47 秒を要していた．これらの時間から，バルク処理待ち時間（waittime）にはコマンド実行にそれほど影響を与えない程度の時間として 20 秒を指定した．この時間の設定のためのルール作成指針については，5.3 節で後述する．そして作成する VM 数分のストレージ設定命令を，この間にすべて発行するようにした．

測定結果を，**図 6** に示す．提案手法は手順 2 と手順 3 の時間を短縮したが，手順 4 と手順 5 については短縮されないため，図 2 とほぼ同程度の時間になった．その結果，手順 2 と手順 3 に比較して，手順 4 に要する時間が著しく長くなり，同一グラフへの表現が困難であるため，図 6 には短縮した手順 2 と手順 3 のみを記載している．

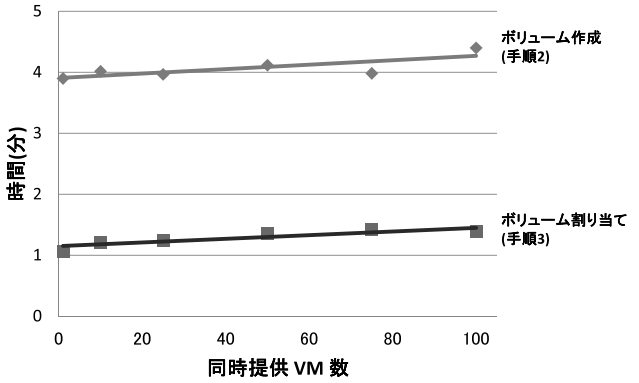


図 6 VM 数とストレージ設定に要する時間の関係（提案手法）
Fig. 6 Storage provisioning time by number of VMs (proposed method).

従来は 100 個のボリューム作成と割当てに，約 450 分（図 2 での手順 2 と手順 3 の合計値）を要していたのに対し，提案手法では，約 320 秒（図 6 での手順 2 と手順 3 の合計値）で実施できた．また，VM 数を 1 台から 100 台まで増加させても，これらの時間は合計で 50 秒程度しか増加しなかった．このように提案手法では，従来の管理コンソールからのストレージ設定リクエストを変更せず，その結果としてシステムへの大幅な変更を加えることなく，バルク処理によって設定時間を短縮できることが確認できた．

4.2 実験 2：ストレージ装置の用意するバルク処理の多重度による影響の測定

4.1 節に記載した実験では，同時実行命令最大数（max_commands）に 100 を指定した．これは，評価実験に利用したストレージ装置に 100 命令を同時に受け付けるバルク処理コマンドが用意されていたからである．そこで，ストレージ装置に用意されたバルク処理コマンドの処理できる多重度による影響を評価するため，実験 1 に対して max_commands の値を変化させて，影響を測定した．

4.1 節の実験 1 と同様に，ボリューム作成から VM 作成までの一連の手順を実施した．作成する VM 数は実験 1 とは異なり 100 台に固定し，設定リクエストをリクエスト解析モジュールに VM 数分だけ連続的に発行した．分類ルールには図 4 の例と同様に，ボリューム作成とボリューム割当てのリクエストを一括処理可能として設定した．なお，ストレージ装置の用意したバルク処理コマンドの多重度が，30 の場合，60 の場合，100 の場合を想定し，max_commands を，30，60，100 に変化させた．バルク処理待ち時間（waittime）には 20 秒を指定し，この間に 100 台分のストレージ設定命令を管理コンソールからすべて発行した．このような状況で，提案手法により短縮される手順 2 と手順 3 に要する時間を測定した．

それぞれの手順について，100 台分のリクエストの発行

表 2 バルク処理の多重度と応答時間の関係
Table 2 Relations between multiplicity of bulk execution and provisioning time.

#	max_commands					
	30		60		100	
手順 2（ボリューム作成）	224 sec	(1～30)	239 sec	(1～60)	234 sec	(1～100)
	445 sec	(31～60)	455 sec	(61～100)		
	689 sec	(61～90)				
	927 sec	(91～100)				
手順 3（ボリューム割当て）	82 sec	(1～30)	82 sec	(1～60)	89 sec	(1～100)
	136 sec	(31～60)	141 sec	(61～100)		
	182 sec	(61～90)				
	234 sec	(91～100)				

を開始してから，応答を得るまでの時間を**表 2** に示す．なお，max_commands を 30 および 60 に設定した場合は，1 回のコマンド実行によりそれぞれ 30 台分，60 台分のストレージ設定命令しか処理できない．そのため管理コンソールから 100 台分のリクエストを一括して発行しても，100 台分のストレージ設定命令を 1 回のコマンド実行によって実行できない．表 2 では max_commands を 30 および 60 に設定した場合に，管理コンソールによるリクエスト発行から数回のコマンド実行が終了して応答を得られるまでの時間を，それぞれのコマンド実行回に分けて記載している．

ストレージ装置の用意するバルク処理の多重度が waittime で設定された待ち時間に発行されるリクエストよりも少ない場合，ストレージ装置に対して複数回のコマンド呼び出しを実施するため，設定完了するまでの時間にはコマンド実行回数分だけのコマンド実行時間を要する．このように提案手法は，ストレージ装置の用意するバルク処理の多重度に影響を受ける．

4.3 実験 3：VM 作成時のバルク処理の待ち時間の測定

提案手法では，個々のリクエストの実行の際に，バルク処理を実行するための待ち時間が生じる．最初にキューイングされたリクエスト（図 4 の場合における (2) や (8) のリクエスト）が待ち時間が最も長くなり，バルク処理実行の直前に発行されたリクエストが待ち時間が最短になる．4.1 節や 4.2 に記載した実験では，waittime で設定した時間内にすべてのリクエストが発行される状況で測定を実施した．しかし現実にはそのような状況ばかりではなく，発行される命令がキューの監視のタイミングで分けられ，別々のバルク処理命令で実行されることもある．その場合は，後の方のバルク処理命令に含まれているリクエストは，待ち時間がさらに長くなる．

そこで，提案手法を利用する際に，個々のリクエストごとに待ち時間がどの程度生じるかを測定した．その際に，まず実際の運用管理においてどのくらいの頻度でリクエストが発生することが現実的であるかを調査し，その結果に合わせてリクエストを発行した．

まずパートナの仮想サーバ運用管理ソフトウェアベンダが想定する VM 作成リクエストの発生パターンのうち，連続実施されるパターンを分析した．その結果，100 回の VM 連続作成時のリクエスト発生間隔は，最小 0.14 秒，最大 1.95 秒（平均 1.04 秒）であった，そこで，平均値の 1.04 秒ごとに VM 作成に必要なボリューム作成とボリューム割当てのリクエストを連続的に発生させた．また，パートナ企業のデータセンタへのヒアリングの結果，VM 作成を少なくとも一度に 10 個程度は一括実施することが多いというコメントを得た．そこで最大値の 1.95 秒間隔で到着した 10 個のリクエストを受け取る間，バルク処理を待ち続けられるように，waittime を 20 秒に設定した．

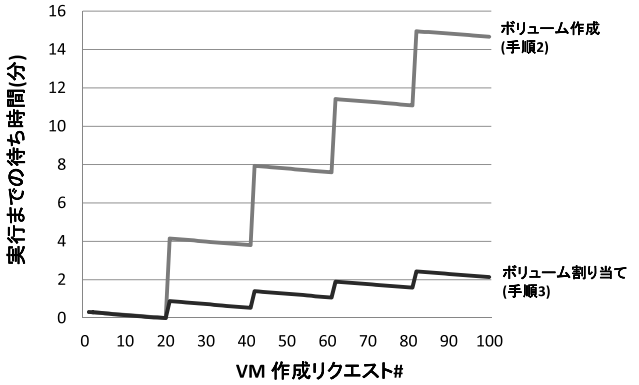


図 7 VM 作成リクエストごとの実行待ち時間
Fig. 7 Waiting time for execution by VM provisioning request.

VM 作成リクエストを 1.04 秒間隔で 100 回連続送信した際の，各リクエスト送信に対応するボリューム作成リクエスト，ボリューム割当てリクエストがキューイングされてから実行されるまでの待ち時間を，**図 7** に示す．それぞれのリクエストが waittime で表されるバルク処理待ち時間分だけキューイングされ，バルク処理待ち時間ごとに一括実行されている．1 つのバルク処理実行単位で見ると，キューイング開始直後（前のバルク処理命令実行直後）に受信したリクエストが一番待ち時間が長くなり，順に待ち時間が減っていく．また，バルク処理命令を連続実行させた場合，2 回目以降のバルク処理実行時は前回までのバルク処理実行の終了を待つ必要がある．バルク処理命令の連続実行回数が多ければ多いほど，実行までの待ち時間に，バルク処理実行終了を待つための時間が加算される．

5. 考察

5.1 VM 構築時間の短縮効果について

実験 1 より，提案手法では VM 数が増加しても，従来に比べ非常に短時間で VM 提供を完了できることが分かった．図 2 と図 6 からボリューム作成処理（手順 2），およびボリューム割当て処理（手順 3）を比較すると，VM 数の増加に対する処理時間の増加の幅が，提案手法ではそれぞれ約 1/1000 倍，約 3/1000 倍になっていた．ストレージ設定ではシーク時間や回転待ち時間のオーバヘッドが大き

く，バルク処理により個々の設定のつど実施することを防ぐことができる．このため，提案手法では処理時間の増加を抑えることができた．

なお，提案手法では手順 2 と手順 3 以外の時間を短縮しない．2.2 節で言及したように，大規模サービス基盤構築時に VM 提供を長時間化させる原因になる手順はこれら 2 つの手順であった．実験結果より，それらの処理を十分に高速化できることが分かった．

100 台の VM をすべて同一の大規模ストレージ上のボリュームに作成した場合，2.2 節の予備実験の結果では 100 VM で 7 時間を要していた．図 6 より，提案手法ではこの時間を 10 分以内に短縮でき，大規模サービス基盤の構築を迅速に実施できる．

5.2 提案手法が影響を受けるパラメータについて

提案手法は，ストレージ装置の用意するバルク処理命令を利用できない仮想化管理ソフトウェアからの命令を，バルク処理命令を利用できるように変換することで，VM 構築時間を短縮している．このため，まずストレージ装置の用意するバルク処理命令の処理能力に影響を受ける．

実験 2 では，max_commands に設定したバルク処理命令の多重度を変化させたときの，100 VM 構築に要する時間を測定している．バルク処理命令の多重度が大きい場合，バルク待ち時間以内に到着した多くの命令を 1 回のバルク処理命令で処理できる．バルク処理命令の多重度が小さい場合，バルク待ち時間以内に到着した命令を分割して実行する必要が生じる．そのため，表 2 に記述したように，複数回のバルク処理実行が発生し，単一のバルク処理命令の倍数分だけの処理時間を要する．

実験 3 では，間隔をおいて命令が連続実行された場合の，各リクエストに対する待ち時間を示している．バルク処理命令の待ち時間（waittime）にすべての命令が発行されないと，この実験結果のように複数回のバルク処理命令に分割される．さらにバルク処理命令の処理時間がバルク処理命令の待ち時間より長いと，バルク処理命令を複数回実行した場合に，2 回目以降のバルク処理命令では前回までのバルク処理の実行終了を待つ必要がある．

このように，提案手法は，バルク処理命令の多重度と，バルク処理命令 1 回あたりの処理時間と，バルク処理命令の待ち時間に影響を受ける．これらのパラメータの設定のための指針について，5.3 節で述べる．

5.3 分類ルール作成指針

運用管理者は分類ルールを記述することにより，提案手法によってバルク処理に変換する場合の動作を指定できる．3.2 節で記述した分類ルールのうちの大部分は，対象とするストレージの装置の仕様に基づいて決定される．本節では，それ以外の運用管理者知識に基づいて記述される部分について，その作成指針を示す．

まず，一括実行可能なコマンドを bulk_support グループに列挙する．bulk_support グループは，ストレージモデル名とファームウェアの番号ごとに作成する．バルク処理可能なコマンド数や実行処理待ち時間に，複数コマンド間で異なる値を指定する場合は，bulk_support グループを複数記述し，値の異なるコマンドどうしを別グループに記述する．

バルク処理可能なコマンド数（max_commands）とバルク実行処理待ち時間（waittime）は，実際のバルク処理実行に要する処理時間を基に決定する．このため，複数のコマンドでバルク処理実行に要する処理時間が大きく異なる場合は，これらの値を異なる値に設定すべきであり，それらのコマンドは別の bulk_support グループに定義する．

bulk_support グループ内に記述される max_commands は，実験 3 で記述したように，実際の運用管理現場において一度にどれくらいの VM 作成が同時に実施されるかによって決定される．実験 3 のようなヒアリングや，あるいは過去の運用管理ログを解析することによって決定する．waittime については，運用管理ログから VM 作成がどれくらいの間隔で実施されることが多いかを解析し，その結果により決定する．なお，バルク処理実行可能なコマンドがストレージ装置に用意されていても，運用ポリシによってはコマンド実行にバルク実行処理のための遅延を生じさせることが困難な場合もある．その場合は max_commands やコマンド実行間隔とは関係なく，waittime に許容できる遅延時間を設定する．waittime を小さくした場合，受信した命令数がそれほど多くない状態でもバルク実行処理コマンドを実行するため，遅延時間が短くなる．しかし，バルク実行処理コマンドの実行回数が増加する場合もある．その場合は，最初の方に受け取った命令については遅延時間を少なく実行できるが，後の方に実行されるバルク実行処理コマンドに含まれる命令の実行については，前のバルク実行処理コマンドの終了を待つ必要があるため，結果として遅延時間が大きくなる可能性がある．なお，遅延をまったく許容できないコマンドについては，bulk_support グループ内に記述しない．

また，exclusive グループには，同じ装置に対して同時実行できないコマンドセットを記述するが，運用管理ポリシによっては意図的に同時実行をしないコマンドを記述する場合もある．たとえば 3.2 節で記述したルールの一例では，異なるボリュームを対象にした場合にボリューム作成（CreateOrModifyElementFromStoragePool）とパス作成（ExposePaths）を同時に実行可能である．これは対象とするストレージ装置の仕様に基づいて決定している．しかし，実際にボリューム作成とパス作成を同時に実行するとストレージ装置内のログが混在してしまい，障害発生時の原因の切り分けが困難になるかもしれない．そのような

場合には，exclusive グループに同時実行できないコマンドセットとしてそれらのコマンドを記述することにより，同時実行させないように指定する．

個々のコマンド実行について，ストレージ装置側で実行される複数の処理の間に論理的な依存関係がある場合，状態遷移を定義してそれぞれのコマンド実行前後で遷移する状態を記述する．たとえば，ボリューム割当て処理はボリューム作成を実施した結果作成されたボリュームに対して実施するため，単純にバルク処理を実施することができない場合がある．3.2 節で記述したルールの一例では，コマンド実行対象のボリュームが，ボリューム割当て処理の実行前状態（created）に遷移していなければ，実行できないことを記述している．このように，個々のコマンド実行について，単純にバルク処理に変換するための定義だけではなく，運用管理シーケンスを考えたときに順序関係がある場合に，各コマンドに対して状態遷移を定義する．

このように，提案手法ではストレージ装置の振舞いと運用ポリシに合わせて，ルールを作成することができる．実験 1 での提案手法の処理は，3.2 節で記述したルールに従っているが，この実験環境では 20 秒以内に 100 リクエストを受け取る可能性があることを，運用ポリシとして記述している．その結果として，受け取ったすべてのリクエストを，1 つのバルク処理命令として実行している．また，実験 2 では，実験 1 でのルールに対して max_commands を変更している．このため，20 秒以内に多数の VM 作成リクエストが発行されても，max_commands が小さい場合には設定されたコマンド数以下のコマンドを含むバルク処理命令を，複数回発行している．

5.4 提案手法適用のメリットとデメリット

提案手法を適用することにより，個々のストレージ装置のバルク処理命令に対応できない仮想化管理ソフトウェアを改造することなくバルク処理命令を活用できる．実験 1 では，従来ではボリューム作成（CreateOrModifyElementFromStoragePool）とパス作成（ExposePaths）命令による呼び出しがそれぞれ 100 命令ずつの合計 200 命令が実施されていたが，リクエストキューに蓄積した命令をバルク処理命令に変換することにより，2 つのバルク処理命令の実行に変換している．実験 1 ではバルク実行待ち時間内に，仮想化管理ソフトウェアからのすべての命令を受け取る想定で実験を実施したため，このように効率良く命令実行を削減できた．また，実験 3 では実際のデータセンタでの利用をヒアリングした結果に基づき命令を発行したが，こちらについても，仮想化管理ソフトウェアからの 20 命令前後の命令を 1 つのバルク処理命令に変換していることを確認できた．個々のストレージ設定命令の実行時間が長い場合，バルク処理命令を活用できることにより，処理時間を大きく削減することが可能であり，大規模サービス実現のための仮想化環境を迅速に構築することができる．なお本論文で問題として取り上げているのは，2 章に記述したように VM 構築に要する時間の長時間化であるが，提案手法は，表 1 に記載したように SMI-S に定義されているすべての設定用命令セットを対象にしている．このため，大量の VM を同時に削除する場合に，ボリューム割当て解除やボリューム削除のための処理時間が長時間化する場合にも，提案手法を適用することにより高速化を実現できる．

一方で，個々の命令にある程度の遅延時間を発生させ，バルク処理命令により一括実行させるため，同時に複数の命令が発行されないと，命令発行の遅延だけが発生し，一括化の効果が得られない．バルク処理のための待ち時間（分類ルールの waittime）の時間を実際の命令実行時間に比較して適切な時間に設定しないと，この遅延時間が命令実行時間と比較して大きくなる可能性がある．そのため 5.3 節に示したように，実際の運用管理現場での過去の管理ログを参照に，適切な時間を設定する必要がある．また，バルク処理命令では内部的に一度に複数命令を実行するため，1 回の命令の実行時間が元の命令よりも大きくなる可能性がある．実際に実験 1 の結果からは 100 命令をバルク処理させることで，1 命令の実行よりもボリューム作成（CreateOrModifyElementFromStoragePool）とパス作成（ExposePaths）命令の合計で 50 秒ほど実行時間が大きくなっている．この実験で利用したストレージ装置はそもそも個々の命令の実行時間が大きかったため，この増加分が実行時間にそれほど影響しなかった．しかし，実際に利用するストレージ装置での個々の命令の実行時間と，それをバルク処理に変換した場合の実行時間の差異によっては，バルク処理に変換しても，個々の命令をそのまま呼び出した場合と処理時間がそれほど変わらなくなる場合もある．

提案手法が有効に機能するかどうかは，これらのメリットとデメリットのトレードオフになる．実際のデータセンタを対象に実施した事前調査では，1 つの VM を作成する場合でさえ，複数のボリュームを利用しているケースが大半であった．たとえば，システム領域とデータ領域を別個の領域に作成していたり，ログ用の領域を別個の領域に確保したりしており，1 つの VM の作成でも複数ボリュームに対するストレージ設定を実施していた．ヒアリングを実施した対象のデータセンタのように大量の VM を一括して作成するようなデータセンタではなくても，1 つの VM 作成に必要な設定命令の発行の場合に複数のボリュームを利用することが一般的である．運用管理ログを解析し，waittime や max_commands を 1 つの VM 作成に必要な値に設定すれば，そのような場合でも提案手法を有効に利用できる．

なお，バルク処理可能なリクエストはストレージ装置により異なるため，仮想化管理ソフトウェアが必要とする特定の命令については，ストレージ装置側でバルク処理命令

セットが用意されておらずに，提案手法による変換処理を実施できない場合もある．実際に今回評価で利用したストレージ装置とは別のストレージ装置では，ボリューム作成（CreateOrModifyElementFromStoragePool）についてはバルク処理を実装していなかったため，VM 作成時の運用管理手順のうちボリューム割当てに対してしか提案手法による変換処理を実施できなかった．バルク処理可能なコマンド数についても，同様にストレージ装置やファームウェアのバージョンにより異なっていた．ただし提案手法ではそれらを分類ルールとして記述可能である．表 1 に記載した命令のうちの一部についてしかバルク処理を実装していなかった場合や，その特性が異なっていた場合でも，バルク処理として変更される特性を分類ルールのパラメータとして記述することで，バルク処理可能なコマンドに対して提案手法による変換処理を実施できる．

6. 関連研究

ストレージ装置へのリクエストを一括処理する手法は，3.1 節で述べたように，データベースアクセス高速化のために一般的に利用されており，ストレージ装置に対してバルク処理を実施するための手法やインタフェースも提案されている [10], [11]．しかしこれらは，3.1 節で述べたストレージベンダ独自のバルク処理と同様に，それぞれの方式に従った API 呼び出しを実施する管理ソフトウェアの実装が必要になる．このため，本研究で扱った，仮想化管理ソフトウェアから直接ストレージ設定を実施する場合の高速化手法としては利用できない．

また，VM 構築全体に要する時間を短縮する手法として，Borges らは，モデル駆動で自動プロビジョニングを行う方法を述べており [12]，Chapman らは，クラウドプロビジョニング自動化のためのアーキテクチャを提案している [13]．これらの手法は，プロビジョニング自動化のための製品群 [5], [6] と同様に，オペレーションを減らすことで VM のプロビジョニングを高速化する．また，De らはリクエスト履歴を分析し，プロビジョニング処理を予測したうえで高速化する方法を提案しており [14]，Bjorkqvist らや Malawski らは，プロビジョニングのポリシを設定したり，プロビジョニングのためのタスクスケジューリングを行ったりすることで，プロビジョニングを高速化する方法を提案している [15], [16]．本研究の提案手法は，これらの自動化技術やスケジューリングによる高速化手法を実施する際のプロビジョニングの処理自体を高速化するものであり，これらの関連研究と同時に利用することで，プロビジョニング処理をより高速化できる．

7. まとめ

本研究では，大規模サービス基盤を構築する場合に，VM 提供の際のストレージ設定時間が長くなることを，予備実験の結果より明らかにした．そして複数の VM を提供する際のストレージ設定を高速化し，VM 提供にかかる時間を短縮するための手法を提案した．ストレージ装置が提供するベンダ独自の一括処理命令をサーバ仮想化機構からは利用できないため，提案手法ではストレージ装置の代わりにサーバ仮想化機構からのストレージ設定命令を受信し，キューイングして一括処理する．これにより，ボリューム作成やボリューム割当てに要するリクエスト送受信処理を削減した．この際に，仮想化管理ソフトウェアからの呼び出し先を変更するだけで利用できるようにした．そして評価実験により，多数の VM を一括して提供する場合に，従来に比べて設定処理に要する時間を，大幅に短縮できることを確認した．実験環境では 100 VM の構築に 7 時間を要していたが，提案手法ではこの時間を 10 分以内に短縮できた．これにより，大規模サービス基盤の構築にかかる時間を削減できる．

提案手法は，ストレージ装置に用意されたバルク処理命令に対応できない仮想化管理ソフトウェアに対して，それを改造することなくバルク処理命令を活用することを実現するための方式である．しかし，ストレージベンダ独自で用意した他社製品に対する差別化のための機能については，バルク処理命令に限らなくても同様の問題が起こりうる．提案手法に対してルール記述と，ルール解析処理部分を拡張すれば，その場合にも対応可能である．したがって，バルク処理にかかわらず，ストレージベンダ独自の命令を用意している場合で，なおかつその機能を仮想化管理ソフトウェアから利用させることが効果的な場合に，提案手法を効果的に活用できる．

参考文献

[1] Barroso, L.A. and Holzle, U.: *The Datacenter as a Computer – An Introduction to the Design of Warehouse-Scale Machines*, Morgan and Claypool Publishers (2009).

[2] VMware, Inc.: Virtual Volumes (VVOLs) Tech Preview (online), available from ⟨http://blogs.vmware.com/vsphere/2012/10/virtual-volumes-vvols-tech-preview-with-video.html⟩ (accessed 2014-08-20).

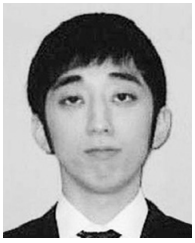
[3] OpenStack Foundation: Cinder (OpenStack Block Storage (“Cinder”) (online), available from ⟨https://wiki.openstack.org/wiki/Cinder⟩ (accessed 2014-08-20).

[4] Storage Networking Industry Association: Storage Management Technical Specification, Part 3 Block Devices Version 1.6.0, Revision 4 (online), available from ⟨http://www.snia.org/sites/default/files/SMI-Sv1.6r4-Block.book_.pdf⟩, pp.57–59 (Feb. 2012) (accessed 2014-08-20).

[5] VMware, Inc.: VMware White Paper: Automated Provisioning and Deployment (online), available from ⟨http://www.vmware.com/files/pdf/services/VMware-Auto-Provisioning-Whitepaper.pdf⟩ (accessed 2014-08-20).

[6] IBM Corporation: Tivoli Service Automation Manager (online), available from ⟨http://www-01.ibm.com/

software/tivoli/products/service-auto-mgr/⟩ (accessed 2014-08-20).
[7] Michael, A., Armando, F. Rean, G., et al.: A View of Cloud Computing, *Comm. ACM*, Vol.53 April, pp.50–58 (2010).
[8] Cooper, F.B., Ramakrishnan, R., Srivastava, U., et al.: PNUTS: Yahoo!'s hosted data serving platform, *Proc. VLDB Endowment*, Vol.1, No.2, pp.1277–1288 (2008).
[9] 猿渡俊介，高木潤一郎，川島英之ほか：センサデータベースマネージャにおける問合せ処理とデータ圧縮の同時最適化，情報処理学会論文誌，Vol.53, No.1, pp.320–335 (2012).
[10] McCullough, J.C., Dunagan, J., Wolman, A., et al.: Stout: An Adaptive Interface to Scalable Cloud Storage, *Proc. USENIX Annual Technical Conf.*, pp.47–60 (2010).
[11] Google, Inc.: Google Cloud Storage: Sending Batch Requests (online), available from ⟨https://developers.google.com/storage/docs/json_api/v1/how-tos/batch⟩ (accessed 2014-08-20).
[12] Borges, H.P., De Souza, J.N., Schulze, B., et al.: Automatic generation of platforms in cloud computing, *Proc. IEEE/IFIP Network Operations and Management Symposium* (*NOMS*), pp.1311–1318 (2012).
[13] Chapman, C., Emmerich, W., Marquez, F.G., et al.: Software architecture definition for on-demand cloud provisioning, *Proc. 19th ACM International Symposium on High Performance Distributed Computing* (*HPDC '10*), pp.61–72 (2010).
[14] De, P., Gupta, M., Soni, M., et al.: Caching techniques for rapid provisioning of virtual servers in cloud environment, *Proc. IEEE/IFIP Network Operations and Management Symposium* (*NOMS*), pp.562–565 (2012).
[15] Bjorkqvist, M., Chen, L.Y. and Binder, W.: Opportunistic Service Provisioning in the Cloud, *Proc. IEEE 5th International Conference on Cloud Computing* (*CLOUD 2012*), pp.237–244 (2012).
[16] Malawski, M., Juve, G., Deelman, E., et al.: Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds, *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC12*), pp.1–11 (2012).



中島 淳

2003 年慶應義塾大学理工学部卒業．2005 年同大学大学院理工学研究科修士課程修了．同年株式会社日立製作所システム開発研究所（現，横浜研究所）入所．現在同研究所研究員．IT システム運用管理に関する研究に従事．

名倉 正剛 （正会員）

1999 年慶應義塾大学理工学部卒業．2001 年同大学大学院理工学研究科修士課程修了．2006 年同大学院理工学研究科博士課程単位取得退学．博士（工学）．2001～2003 年日本電気株式会社．2006～2007 年慶應義塾大学大学院理工学研究科特別研究助手．2007～2009 年奈良先端科学技術大学院大学情報科学研究科特任助教．2009 年日立製作所システム開発研究所（現，横浜研究所）入所．現在同研究所研究員．ソフトウェア工学，IT システム運用管理に関する研究に従事．電子情報通信学会，日本ソフトウェア科学会各会員．

柴山 司 （正会員）

2002 年京都大学工学部電気電子工学科卒業．2004 年同大学大学院情報学研究科修士課程修了．同年株式会社日立製作所システム開発研究所（現，横浜研究所）入所．現在同研究所研究員．IT システム運用管理に関する研究に従事．SNIA 日本支部技術委員会会員．

坂下 幸徳 （正会員）

2003 年北陸先端科学技術大学院大学情報科学研究科情報システム学専攻修士課程修了．同年株式会社日立製作所システム開発研究所（現，横浜研究所）入所．同研究所主任研究員．2012 年北陸先端科学技術大学院大学情報科学研究科後期課程（社会人コース）入学．現在 Hitachi America, Ltd. 出向中．同社 R&D SNSL Lab Manager & Senior Researcher．IT システム運用管理の研究に従事．SNIA Technical Council Member，SNIA 日本支部技術委員会副委員長．