複数VMによる並列分散システムのための動的な性能推定法の提案

君山 博之[1,a)]　丸山 充[2,b)]　小島　一成[2,c)]　藤井 竜也[3,d)]

概要：必要なときに必要な数のクラウド上の仮想計算機 (VM) を使って並列分散処理を行うことのできるシステムは，映画制作におけるレンダリング処理やインシデント発生時のログ解析など，一時的な大容量計算が必要なユーザから要望が多いシステムである．そのような要望に応えるため，我々は複数の VM を動的に組み合わせて並列計算させることによって大容量計算を行うためのフレームワークを提案している．しかし，VM は計算機の CPU や I/O などのハードウェアリソースを複数の VM 間で共有使用しているため，得られる性能はリソースを共有している VM の負荷に依存し，そのことが所望の性能を得るのに必要な VM 数の正確な見積もりを難しくしている．特に，複数の VM を使って並列計算させているときに性能不足を補うつもりで VM を追加した場合であっても，既に稼働中の VM が使用しているハードウェアを，追加した VM が共有使用することによって事前の想定よりもシステム全体の性能が向上しない可能性がある．この問題を解決するため，我々は所望の性能を得ることのできる VM 数を動的に推定する手法について研究を行っている．必要な VM 数を推定するためには VM の性能を推定する必要があるが，複雑な VM の構造を考慮し，その性能を正確に記述できる有効なモデルがないことから，実測値から VM 性能を推定する方法を検討した．本論文では，前述のフレームワークを用いて同じハードウェア上に VM を追加し，同時に計算処理を実行した際にどのように性能が低下するかを実測した結果を示し，その実測結果を元にした VM 数と性能との関係を記述する数学的モデルを提案する．さらに，そのモデルを使った VM およびシステム全体の性能推定方法を提案し，我々が提案する推定結果を使った VM 増設法について説明する．

1. はじめに

昨今，計算機そのものではなく計算機上に構築した仮想計算機 (VM:Virtual Machine) を時間単位で貸し出すクラウドサービスを使用するユーザが増えている．平成 30 年度の総務省の調査によれば，日本の企業の半数以上がクラウドサービスを導入しており，未導入の企業の多くがクラウドサービスの導入を検討していることが報告されている [1]．クラウドサービスには，計算機を自前で導入することなく使いたいときに使いたい期間だけ計算機を使えるメリットがある．これによって計算機導入のための設備投資や継続的なリース料の支払いが不要になり，特に日常的に計算機を使用しないユーザにとっては，クラウドサービスは大きなメリットがあるサービスである．

一般に，クラウドサービスとしてユーザに提供されている VM は，1 つの計算機ハードウェア上に複数構築されており，その計算機ハードウェア (CPU，Memory，Network Interface，内部バスなど) を，複数の VM と共有しながら動作していることから，その性能を正確に見積もることは難しい．そのため，ベストエフォートで提供しても支障がないメールや Web サービスをクラウド上に構築して運用するのが，主な使い方となっている．前述したように，クラウドサービスは時間単位で利用できることにそのメリットがあると考えられる．例えば，映画制作における合成処理やレンダリング処理は，編集工程の中でも大容量計算が必要な工程であり，そのような処理に対してクラウドサービスとして提供されている VM を大量に使用し，並列計算により非常に短い時間で処理ができれば，低コストで作業時間の短縮を図ることができると考えられる．

そのような背景の下，我々は，ネットワーク上に分散している大量の VM を一時的に利用し，大容量計算を並列

[1] 東京電機大学
Tokyo Denki University, Adachi, Tokyo 120–8551, Japan
[2] 神奈川工科大学
Kanagawa Institute of Technology, Atsugi, Kanagawa 243–0292, Japan
[3] NTT 未来ねっと研究所
NTT Network Innovation Laboratories, Yokosuka, Kanagawa 239–0847, Japan
[a)] kimiyama@mail.dendai.ac.jp
[b)] maruyama@nw.kanagawa-it.ac.jp
[c)] kojima@ic.kanagawa-it.ac.jp
[d)] fujii.tatsuya@lab.ntt.co.jp

的にかつリアルタイムで実行可能なシステムとそのためのフレームワークを提案している．さらに，広域分散配置された複数の計算機上に，ハイビジョンの 4 倍の解像度を持つ圧縮されていない 4K 映像の合成処理のアプリケーションを，そのフレームワークを使って実装し，実験によってその有効性を確認している [2]．この提案フレームワークは，ネットワーク上に分散されている複数の VM を使って並列分散処理させ，同時に各 VM の計算時間や転送処理時間をモニタし，そのモニタ情報を使ってデッドラインまでに計算を完了できるように VM 数を動的に増減させることによってリアルタイム処理を可能にしている．このフレームワークを使ってコストを抑えて計算を完了させるためには，計算に使用する VM 数を最適化し，なるべく少ない数の VM により計算させることが望ましい．そのためには VM における計算時間やデータの転送処理時間を適切に見積もる手法を導入する必要がある．

しかし，前述したように，VM は計算機ハードウェアリソースを他の VM と共有利用している．計算処理だけではなく外部ネットワークを介した VM の通信にも CPU やメモリを使用するため，ハードウェアの共有状態が非常に複雑になり，VM 性能のモデル化を困難にしている．また，ハードウェアを共有している他の VM がどのような処理を行っているかを知る手段がないことも，VM 処理性能を正確に把握することを難しくしている一因となっている．そのため，システム全体の処理性能を上げようとして VM を追加したときに，追加される VM の処理性能やシステム全体性能を正確に推定することは困難である．加えて，VM を追加したときに，先に構築した VM と同じハードウェア上に新しい VM が構築されてしまう可能性があり，その場合，既存の VM の処理性能を低下させることとなり，システム全体の性能が予想よりも向上しない可能性があることが，さらに問題を複雑にしている (文献 [3][4])．

これまで VM 上で動作するアプリケーション性能を予測する目的で，以下の研究が行われている．R. C. Chiang らは文献 [5][6] において，Support Vector や統計量を使った機械学習ベースの VM 性能予測による性能評価方法を提案している．また，S. Kundu らは，文献 [7] において，Artificial Neural Network を使って性能計測データを機械学習させることによって VM の性能を予測する方法を提案し，T. Wood らは文献 [8] において回帰モデルを使った性能予測法を提案している．これらの文献で提案されている手法は，リソースの競合状況を考慮せずに VM の性能を統計的な手法により予測する手法であり，これら手法を使って追加した VM が同じハードウェアを共有使用した場合の性能低下を予測することは難しい．また，Novakovic らは，文献 [3] において，リソース競合により性能が低下した VM を発見し，物理的に異なる計算機ハードウェア上に VM ごと移設する DeepDive システムを提案しているが，計算機を指定できない一般のクラウドサービスを使用する場合には適用が困難である．上記のように，VM 間のリソース競合による性能劣化まで考慮にいれた性能評価法については，ほとんど研究されておらず，試行錯誤によって最適なシステム構成を模索しているのが現状である．

そこで，競合による VM の性能低下が起こることを前提とし，システム全体の性能を動的に評価する手法を確立することを目標に，我々は，同じハードウェア上に構築した他の VM の負荷によって，VM ごとの処理性能がどのように変動するかを実測し，競合による VM の性能低下についての定量的な評価を行うとともに性能低下についてのモデル化，定式化を行った．そして，そのモデルを使った各 VM の性能推定方法，および，システム全体の性能の推定方法，その推定値を使った VM の増設方法を提案し，その考察を行った．

本論文では，第 2 章において我々が提案している複数 VM を使った大容量計算システムとそれを実現するフレームワークの概要について説明し，第 3 章において 1 台の計算機上で同時に動作させる VM 数を変えた場合の合成処理時間変動の実測結果を示す．第 4 章では VM の性能推定方法およびその推定値を使った VM の増設方法の提案とその提案方法に対する考察を示す．そして，最後の章において本論文のまとめを行い，今後の課題について示す．

2. 複数 VM を使った大容量計算フレームワークの概要

図 1 に，我々が文献 [2] において提案している複数 VM を使った大容量計算システムの概略構成図を示す．このシステムは，計算したい元データを格納する複数の Source node，Load balancer，Management node，VM 上で計算を実行する複数の Processing node，計算結果を集約する Destination node から構成される．このシステムでは，Source node から Load balancer に対して計算させたい元データを送信し，Management node が選択した Processing node に対して，Load balancer がその元データを転送する．データを受信した各 Processing node は計算を行い，その結果を Destination node へ転送し，Destination node は受信した計算結果を 1 つにマージする処理を行う．

さらに，この大容量計算システムを容易に構築できるように，我々は，図 2 に示す各ノードのためのソフトウェアフレームワークも提案している．このフレームワークの詳細は以下の通りである．まず，Load balancer には OpenFlow Switch (OFS) を用い，Management node 管理下の空いている VM と Source node との間に文献 [9] で提案している仮想的なオーバーレイネットワークを一時的に設定し，そのネットワークを介して VM へのデータ転送を実現する．Management node には，Open Source Software のデータベース管理システム (DBMS) である Redis[10] を導入し，

DBMS に各 VM の負荷情報を蓄積できるようにする．その Management node では，各 VM の負荷情報を Fluentd[12] を使って収集し，その負荷情報にもとづき Ryu[11] を経由して OFS を制御する．Source node，Processing node，Destination node には，HTTP/TCP/IP プロトコルにより通信を行うための独自開発した共通通信ソフトウェアモジュール (Communication module) を導入する．さらに，これらのノードには，各ノードに割り当てられた処理を行うためのソフトウェアモジュール (Processing module) を導入できるようにしている．アプリケーション全てを作り替えずに様々な処理を実現できるように Processing module には Shared object 形式を採用している．

この大容量計算システムは，クラウド上の複数の VM を物理的な位置や性能に関係なく動的に組み合わせることによって，リアルタイム大容量計算を可能にするシステムである，独立して並列計算可能な計算のみを対象としているが，映像フレーム単位で実行可能な合成処理や，状態遷移を伴わないシステムをシミュレートするモンテカルロシミュレーション，大規模なログからのキーワード検索など，幅広い応用が可能なシステムである．

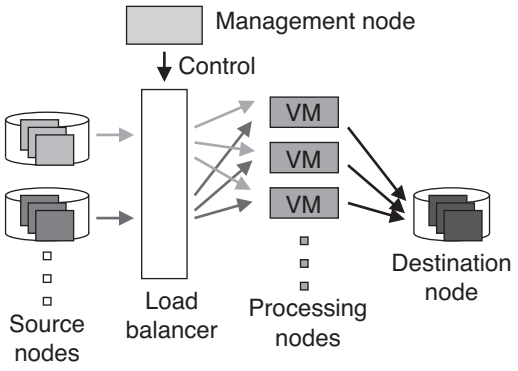

図 1 複数の VM を使った大容量計算システムの概略構成図

3. VM 処理性能計測実験

前述したように，ユーザから判らないように VM は計算機ハードウェアを他の VM と共有しており，その処理性能を理論的に推定するのは難しい．また，計算機ハードウェアを共有することが，VM の計算性能にどのような変化を及ぼすのかを実験的に評価する試みもほとんど行われてこなかった．そこで我々は，文献 [2] において実証実験に用いた非圧縮 4K 映像合成処理アプリケーションを使用し，同時に動作させる VM を追加しながら，各 VM での処理時間がどのように変化するかを実測した．この章では，この実験の概要および実測結果について説明する．

3.1 実験概要

この実験に用いたシステムの構成を図 3 に，評価に用いた計算機のスペックを表 1 に示す．VM を実行するための Hypervisor には Kernel-based Virtual Machine (KVM)[13] を用いた．この図に示すように，1 台の物理計算機上に複数の VM を起動し，同じ物理計算機上で動作している Open vSwitch[14] を使って Source node からの通信を各 VM に振り分けるようにした．この実験において，Processing node に使用した VM には仮想 CPU 数を 2，メモリを 2GB を割り当て，仮想ネットワークインタフェースとして制御用とデータ転送用の合計 2 つのインタフェースを設けた．データ用のネットワークインターフェースのみを上記の Open vSwitch と接続し，Source node との通信に使用した．この VM 上では，OS として Ubuntu server 14.04 LTS を動作させ，その上で非圧縮 4K 映像合成処理を行うアプリケーションを実行させた．

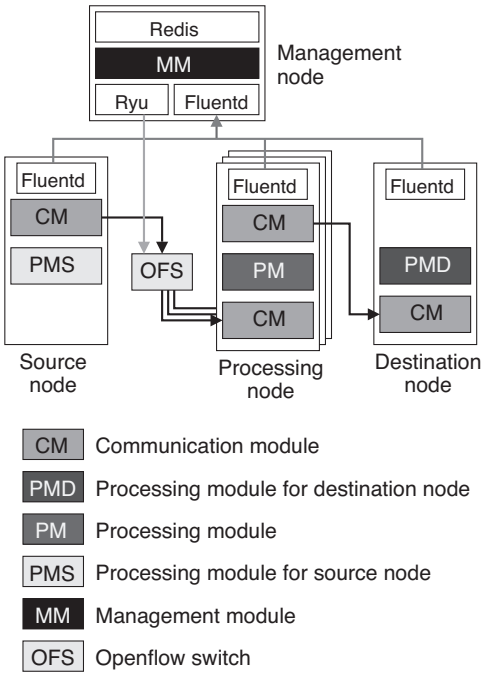

図 2 提案したソフトウエアフレームワーク

処理時間の測定は，同時に動作させる VM 数を 1 台から 20 台まで増やしながら実施した．この合成処理アプリケーションでは，(1)2 台の Source node から，非圧縮 4K 映像の前景映像と背景映像を 1 フレーム (以下，前景画像，背景画像とする) ずつ，全ての VM に対して同時に送信 (Source node からの転送処理) し，(2) 各 VM で合成処理を実行 (合成処理)，(3) 合成処理が終了した VM から順に Destination node に対して合成処理結果の送信 (Destination node への転送処理)，の順に処理が実行される．この処理時間測定実験では，(1)〜(3) の処理時間を以下の手順で計測，記録した．(1) の Source node からの転送処理の処理時間測定は 2 台の Source node で実施した．まず，Source node から画像データの転送を開始した時間を記録し，さらに画像データの転送が完了した通知を Processing node から受け取っ

表 1 評価に用いた計算機のスペック

CPU	Intel(R) Xeon(R) E5-2650 2.0 GHz (8 core) x 2 (HyperThreading ON)
Memory	128 GByte
OS	Ubuntu (Linux) server 14.04 LTS

た時間を記録し，その差分を転送処理時間とした．(2) の合成処理時間の測定では，合成処理が開始してから終了するまでの時間を VM で計測し，記録した．(3) の Destination node への転送処理時間測定では，合成処理を行った VM で Destination node への転送を開始した時間を記録し，さらに画像データの転送が完了した通知を Destination node から受け取った時間を記録し，その差分を転送処理時間とした．

使用した非圧縮 4K 映像の 1 フレームの画素数は 3,840 × 2,160 であり，1 画素には RGB データが各 24 ビットずつ含まれている．また，前景画像には上記の RGB データの他に，マスク情報として α チャンネルデータが 1 画素につき 24 ビット含まれており，前景画像，背景画像のデータサイズは，それぞれ 33,177,854 バイト，24,900,744 バイトである．Destination node へ送られる合成処理結果の画像ファイルは背景画像と同じフォーマットで，そのデータサイズは 24,900,744 バイトである．

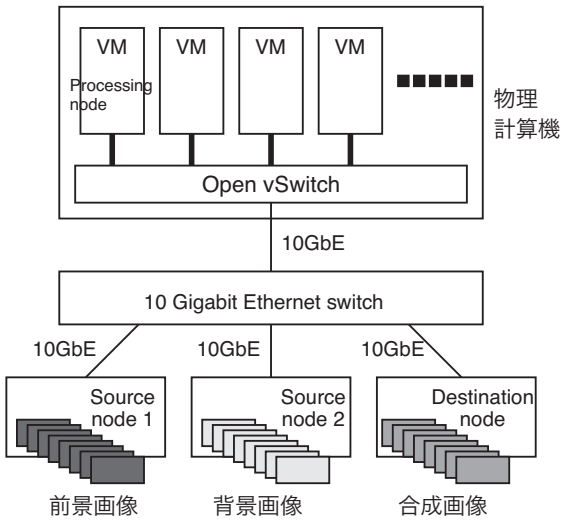

図 3 実験システム

3.2 処理時間計測結果

前述したように，上記の実験システムを使って (1)〜(3) の処理時間計測を行った，処理時間の計測は，全ての VM からの Destination node への合成処理結果の転送が完了するまで待った後，それぞれの VM 数に対して 10 回繰り返し実行した．処理毎の計測結果を以下の節に示す．

3.2.1 Source node からの転送処理時間

図 4 に，同時に合成処理させる VM 数を変化させた場合の，Source node から Processing node への平均転送処理時間の変化を示す．この図の横軸は VM 数，縦軸は転送処理時間である．図中の■は前景画像，▲は背景画像の転送処理時間を表している．図 4 および，後述する図 6，図 8 に示す処理時間の値は 10 回の平均値である．前景画像の方が転送処理により多くの時間を要しているのは，前述したように前景画像には RGB データの他に α チャンネルデータが含まれており，背景画像よりも 1.33 倍データ量が多いためである．この図から VM 数が 7 の場合を除いて，VM が増えるにしたがって徐々に転送処理時間が増えていくことが確認できた．

図 5 に，VM 数を 4，8，12，16，および 20 とした場合の前景画像の転送処理時間分布を示す．横軸が転送処理時間であり，縦軸が観測された割合である．この図のように VM 数が 4 や 8 の場合は，通信路上で，パケットが「衝突する」／「衝突しない」という単純なモデルで表すことができると考えられるため，ポアソン分布に類似する分布になったと考えられる．しかし，VM 数が大きくなるにしたがって，Open vSwitch や各 VM の通信処理を動作させるのに必要な CPU リソースの割り当て待ちによっていくつかの VM の転送処理が待たされるため，複雑な分布になったと考えられる．このように観測された処理時間が複雑な分布を持つことから，1 回の観測によって VM 状態を決めることは難しく，複数回の観測により VM の状態を推測する必要があると考えられる．

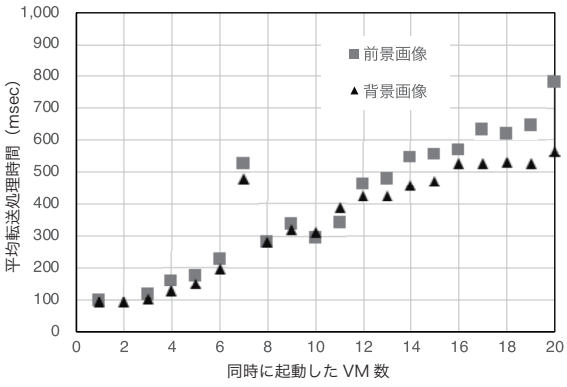

図 4 平均転送処理時間と同時に起動した VM 数の関係

3.2.2 合成処理時間

図 6 に，同時に合成処理を行う VM 数を変化させた場合の平均合成処理時間の変化を示す．この図の横軸は VM 数，縦軸は平均合成処理時間である．グラフ中の縦棒の上端は観測された合成処理時間の最大値，下端は最小値である．VM 数の増加に伴って合成処理時間が増加しているかどうかを確認するために，最小二乗法を用いて VM 数の一次関数へのフィッティングを行った．その結果，この一次関数の傾きの値は 0.675 となった．VM 数 1 のときの平均合成処理時間が 169msec であったことを考慮すれば，この

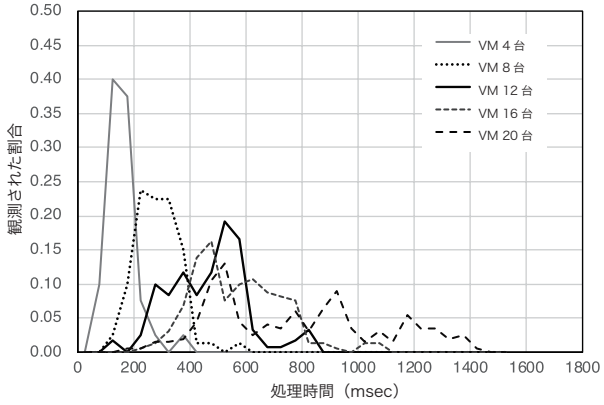

図 5 VM 数を変えた場合の転送処理時間分布

結果は，VM 数が 1 増える毎に平均合成処理時間が 0.4%増加することを意味しており，同時に動作させる VM 数の増加に伴い，わずかではあるが平均合成処理時間が増加していくことが確認できた．

図 7 に，VM 数を 4，8，12，16，および 20 とした場合の合成処理時間分布を示す．この図の横軸は転送処理時間であり，縦軸は観測された割合である．この図から，合成処理時間分布に関しては，VM 数が増えても変化がなかったことが確認できた．この理由として，ハードウェアリソース不足によりリソースが割り振られなかった VM は，合成処理の前の段階である Source node からの転送処理の前でリソース待ちとなり，他の VM の処理が完了しリソースが解放されるまで転送処理を待たされたためと考えられる．このことから，VM 数が増加すると Source node からの転送処理時間が増加する一方で，次の合成処理では待ち時間が生じないため分布にあまり変化が見られなかったと考えられる．

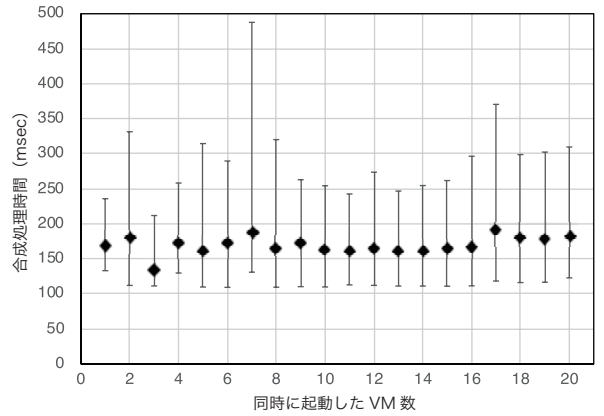

図 6 平均合成処理時間と同時に起動した VM 数の関係

3.2.3 Destination node への転送処理時間

図 8 に，同時に合成処理を行う VM 数を変化させた場合の Processing node から Destination node への転送処理時間の変化を示す．この図の横軸は VM 数，縦軸は平均転送処理時間である．図中の縦棒の上端は観測された平均転

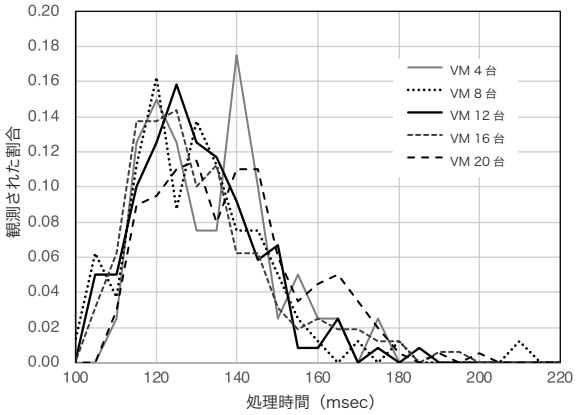

図 7 VM 数を変えた場合の合成処理時間分布

送処理時間の最大値，下端は最小値である．この図から判るように，この転送処理時間は VM 数が増えると徐々に増加していくが，VM 数が 8 台付近からは増加せずに，ほぼ一定となった．3.2.2 節で記述したように，先にリソースが割り当てられた VM は Destination node への転送処理まで完了してしまうため，後からリソースが割り当てられた VM が Destination node への転送処理を実行する段階では，ハードウェアの競合利用がほとんど起きないと考えられる．その結果，大部分の VM はリソース競合がなく，ある程度の転送速度を維持しながら Destination node への転送が可能になるため，このような処理時間変化になったものと考えられる．

図 9 に，VM 数を 4，8，12，16，および 20 としたの場合の Destination node への転送処理時間分布を示す．横軸が転送処理時間であり，縦軸が観測された割合である．この図からも，Destination node への転送処理時間分布は，VM 数が増えてもあまり変わっていないことが確認できた．合成処理と同じように，この処理を実行する段階ではリソース競合がある程度解消されているため，VM 数が増えても転送処理時間には大きな変化が起こらないものと考えられる．

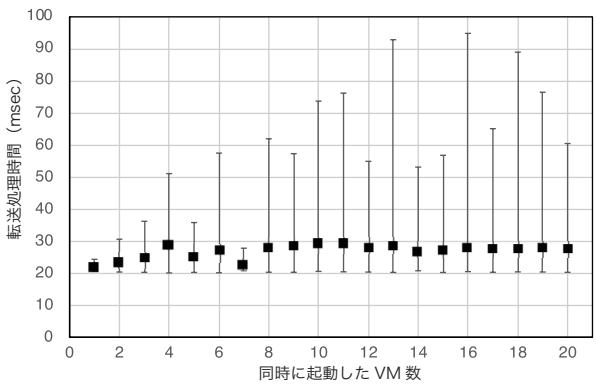

図 8 Destination node への平均転送処理時間と同時に起動した VM 数の関係

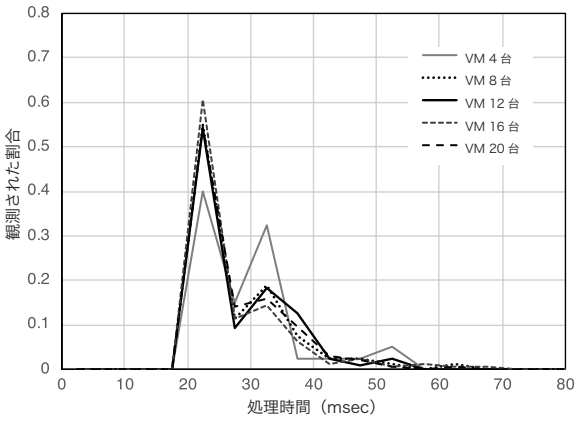

図 9 VM 数を変えた場合の Destination node への転送処理時間分布

4. 合成アプリケーションの VM 追加処理への適用

4.1 動的な VM 数決定方式の提案

文献 [2] において，我々が提案しているフレームワークでは，必要な処理性能を得ることのできる最適な VM 数での運用を可能にするために，VM の追加を動的に行う機能が実装されている．しかし，前章の実験結果が示すとおり，VM が他の VM と共通のハードウェアを使用することによって，VM の処理時間は共有しない場合と比べて長くなることが確認された．このことは，VM の追加により所望の性能が得られるかどうかを事前に評価するのが難しいことを意味している．なぜならば，既に利用中の VM と，これから追加しようとする VM とがどのハードウェアリソースを共有するのかが事前に判らないためである．それは，例えば，ネットワークリソースが性能のボトルネックになっているときに，既に稼働中の VM と同じデータセンターに構築した VM を 1 台追加しても，既に稼働中の VM の通信速度が低下することによって，所望の性能を得られる可能性が保証されないことを意味している．そこで我々は，システム全体の処理性能を所望のレベルまで到達させることを目的に，計測された処理時間データにもとづいて，VM を増設した際のシステム全体の性能を推定し，増設する VM 数を決定する方法について検討を行った．

まず，我々が提案する VM 性能評価データを用いた増設 VM 数の決定手順を以下に示す．

手順 1：VM を実システムに追加したとき，追加した VM を含め各 VM の処理時間がどのように増加したかを記録する．

手順 2：各 VM における処理時間の増加の度合いから，追加した VM と，どの VM がネットワークあるいは CPU リソース (または両方のリソース) を共有しているのかを推定する．

手順 3：手順 2 の結果にもとづいて，新たに VM を追加した場合に各 VM の処理時間と新規に追加する VM の処理時間を，前章の実験結果から推定する．さらに，この推定値から VM 追加後のシステム全体の処理性能を推定し，所望の性能が得られると見込まれる場合は，VM を実システムに追加する．

手順 4：所望の性能が得られると見込まれない場合は，所望の性能に達するまで，VM 数を増やしながらシステム全体の処理性能評価を行う．その結果，所望の性能に達すれば，そのときの VM 数と等しい数の VM を実システムに追加する．

手順 5：VM 追加後に実システム全体性能を評価し，所望の性能が得られなかった場合は，このときの各 VM の処理時間の記録を元に，再度，手順 1 から手順 4 を繰り返す．

以下に上記手順の詳細について説明する．手順 1 において VM を実システムに追加した際に，現在稼働中のどの VM がどのくらい影響を受けるのか，また，追加した VM の性能がどの程度になるのかを推定するための処理時間データを取得する．そして，次に手順 2 において，手順 1 で取得したデータから，新たに追加した VM が稼働中の VM とネットワーク，CPU 等どのハードウェアをどの VM と共有しているかを推定する．手順 3 において，新たに追加する VM も同じようにハードウェアを共有すると仮定し，新たに VM を追加した場合の各 VM の新しい処理時間を推定する．この場合，新しい VM とハードウェアを共有しないと推定された VM は処理性能は追加後も変化がないと見なし，共有すると推定された VM は共有による性能低下分を考慮した処理性能を推定値として使用する．各 VM の処理性能推定値をもとにシステム全体の性能を推定し，所望の性能に達するかどうかを判定する．ここで所望の性能に達することが確認できれば，実システムに VM を追加する．手順 3 において所望の性能に至らない場合に手順 4 を実行する．この手順では，VM をさらに追加したと仮定し，所望の性能に達するまで手順 3 の手法にもとづいて各 VM の処理性能の推定を行い，それらの値から全体処理性能を推定する．所望の性能に達した時点で，そのときの追加 VM 数と等しい数の VM を実システムに追加する．手順 2〜4 において推定されたシステム全体性能と，実際にシステムに VM を追加した後の性能が等しくなるのは，手順 1 で追加した VM と，手順 3 または 4 で追加した VM が同じハードウェアを共有するという仮定が成立する場合である．そこで，上記の仮定が満たされなかった場合に備えて，手順 5 として，VM 追加後のシステム全体の性能を再度測定評価し，不足している場合は手順 2〜4 を再度実行するという手順を加えている．

上記の手順を使ってシステム全体の性能を容易に推定可能にするため，我々は VM の増減が各 VM の処理時間に対してどのような影響を及ぼすかを定式化した．前章の実

測データを用いて，VM 数によって各 VM の処理時間がどのように変わっていくのかを考察し，Source node からの転送処理速度，合成処理速度，Destination node への転送処理速度について，それぞれ VM 数を変数とする定式化を行った．その式を適用することによって，VM を増やした際のシステム全体性能を予測することが可能となる．以下の節で，これらの定式化の方法について詳細に説明する．

4.2 Source node からの転送処理速度の定式化

前章の図 4 に示したように，転送処理時間は同時に動作する VM 数が増えることによって，大きく増加することが確認された．これは通信のためのリソース (ネットワーク帯域や CPU リソース) を各 VM 間で共有していることによるものと考えられる．そこで，A_1 を未定パラメータ，VM 数を X とした時の転送速度 (スループット) を $T_1(X)$ をとして，$T_1(X)$，X，および A_1 が，数式

$$T_1(X) = A_1/X \tag{1}$$

に従う，つまり，Source node からの転送処理においては，帯域リソース A_1 を全 VM で共有するモデルで表すことができると仮定した．この仮定を検証するために，図 4 から求めた VM1 台あたりの前景画像の平均転送速度のうち VM 数 3 から 20 までの値を使って，最小二乗法により A_1 を求めた．図 10 に，図 4 から求めた VM1 台あたりの平均転送速度と，データから求めた A_1 を代入した $T_1(X) = A_1/X$ の曲線を示す．この曲線からも確認できるように，この処理において通信のためのリソースを VM 間で共有しているという仮定は妥当であると考えられる．ただし，VM 数の少ない領域では，送信側 (Source node) または受信側 (Processing node) の処理性能によって通信速度が決まることから，この曲線からは外れてしまったと考えられる．

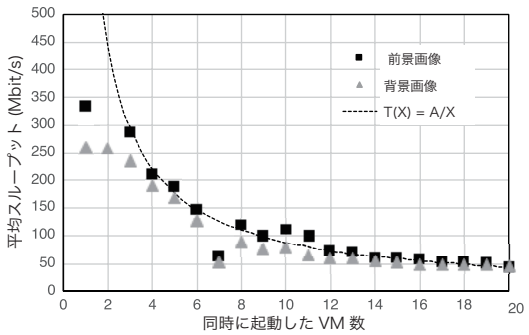

図 10 Processing node への転送における平均スループットと同時に起動した VM 数の関係

4.3 Processing node における合成処理速度の定式化

同時に処理を実行する VM 数が，その平均処理速度や平均処理時間に対して，どのような影響を及ぼすかを理論的にモデル化することは難しい．例えば，性能モデルとしてよく使われる待ち行列理論を用いてこれらを評価する場合，VM 上の OS やアプリケーションプロセス，Open vSwitch，Hypervisor である KVM，ホスト OS などが，1 台の計算機上の有限数 s 個の CPU コアを奪い合う待ち行列モデルを適用する方法がある．待ち行列モデルを使って，平均処理速度を求めるために必要となる平均待ち時間を正確に導出するためには，待ち時間を含まない正味の処理時間分布や処理リクエストの到着時間間隔分布が必要である．しかし，これらの分布は一般的な分布であることから，この系は $GI/G/s$ の待ち行列モデルで表され，その平均待ち時間を解析的に求めるのは極めて困難である．そこで，到着時間間隔分布と処理時間分布をポアソン分布と仮定してモデルを単純化し，$GI/G/s$ の待ち行列モデルの代わりに $M/M/s$ の待ち行列モデルの挙動について考えることとした．

観測される VM の処理時間は，この待ち行列モデルにおいてリソースの空きを待つ平均待ち時間と待ち時間を除いた正味の平均処理時間の和である平均滞在時間として計算できる．そして，今回必要とする平均合成処理速度は，この平均滞在時間の逆数によって導出できることから，ここでは平均滞在時間を定式化する．この待ち行列モデルの平均滞在時間 W はサービス密度 ρ によって，数式

$$W = \frac{1}{\mu}\frac{\rho^s}{(1-\rho)^2}\frac{s^{s-1}}{s!}P_0 + \frac{1}{\mu} \tag{2}$$

により表すことができる [15]．ここで，P_0 は系の中で誰も待っていない確率 (つまり，全ての VM が空いている確率)，μ は平均サービス率 (つまり，平均処理時間の逆数) であり，λ を平均到着率 (つまり，平均到着時間間隔の逆数) とすると $\rho = \lambda/(s\mu)$ の関係がある．

ρ が十分小さい場合，つまり，処理時間よりも到着時間間隔が十分に長い場合（言い換えれば，ネットワークがボトルネックになり CPU リソースに空きが多い場合）を考えると，(2) 式の前半部分の分母，すなわち，$s!(1-\rho)^2$ は ρ の値によらず一定となり，ρ^s に比例することになる．さらに，ρ^s は，ρ の値が 0 の近傍における Tayler 展開を考えた場合，ρ の次数の低い線形多項式 (一次あるいは二次) で近似することができると考えられる．

そこで，線形多項式により表せるかどうかを検証するために，VM 数を X としたときの合成処理時間の平均値 $Y_2(X)$ が，未知の変数 A_2，B_2 により，一次式

$$Y_2(X) = A_2 \times X + B_2 \tag{3}$$

で表されると仮定し，図 6 のデータを用いて最小二乗法により A_2 と B_2 を求めた．図 11 に，その結果を示す．この図から判るように，この合成処理アプリケーションの場合は，VM における合成処理時間は VM 数の一次関数によるモデルで表すことができると考えられる．そして，この処理時間の推定値の逆数を計算することによって，増設後の

処理速度を推定できると考えられる．

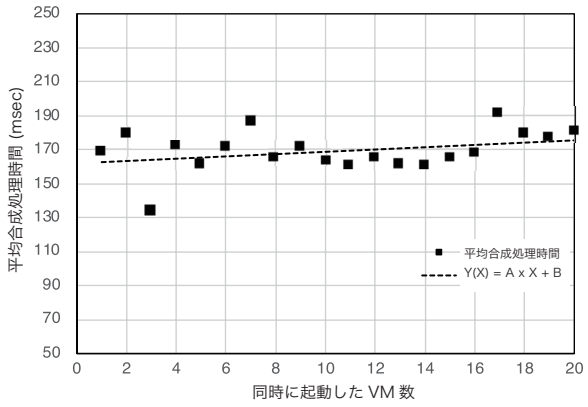

図 11 平均合成処理時間の線形近似

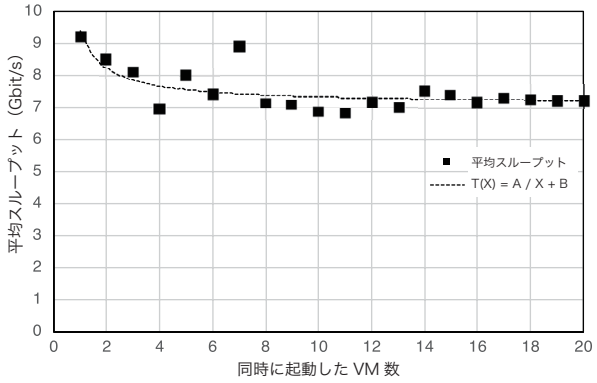

図 12 Destination node への転送における平均スループットと同時に起動した VM 数の関係

4.4 Destination node への転送処理速度の定式化

3.2.3 節で示したように，Source node からの転送処理におけるハードウェアリソースの競合によって後続の処理の競合が緩和される場合，図 8 のように，同時に稼働する VM 数が増えてもある程度のスループットは各 VM で確保することができるものと考えられる．そこで，Destination node への転送処理速度を表すモデルとして，4.2 節で用いた生成した VM の台数に反比例してスループットが減少するモデルに加えて，リソース競合の緩和によって確保できるスループット分を上乗せしたモデルを考える．つまり，X を VM 数，A_3 および B_3 を未定パラメータとしたときに，スループット $T_3(X)$ が数式

$$T_3(X) = A_3/X + B_3 \tag{4}$$

によって表せるというモデルを考える．このモデルは，VM 数が多いときには B_3 の帯域が競合なく確保できるものとし，VM 数が少ないときは，同時に処理を実行中の VM によって帯域 A_3 が分割されるモデルである．

このモデルを検証するために，図 8 の値を使って，同時に使用した VM 数ごとに平均転送速度を求め，その値を使って最小二乗法により A_3 と B_3 を計算した．図 8 の値から計算した平均転送速度と，最小二乗法によって求めた A_3 と B_3 を使って計算した $T_3(X)$ の値を図 12 に示す．この図から，Destination node への転送処理速度はこの提案モデルによって記述することができると考えられる．

4.5 増設 VM 数計算への適用に対する考察

VM の追加処理を行う場合に，上記の (1) 式，(3) 式，(4) 式を使ってシステム全体の性能を推定する方法について説明する．まず，最初のシステムを構築する際には，大まかな処理性能を事前評価し，その評価結果によって VM 数を決定する．そして，必要数の VM を起動し，ソフトウェアをインストールして，処理を実行し始めるとともに，各 VM の処理時間の測定を行う．処理時間の測定は，Source node からの転送処理，合成処理，および Destination node への転送処理についてそれぞれ実施し，VM ごとに記録する．事前評価よりも，システム全体の性能が低かった場合は，VM を増設しながら稼働中の VM の処理時間がどのように変化するかを測定，記録する．

稼働中の VM のうち，増設により処理速度が低下した (つまり処理時間が増加した) VM に対して，処理時間から処理速度を求め，それらの値から上記の (1) 式，(3) 式，(4) 式の未定パラメータ A_1，A_2，B_2，A_3，B_3 の推定を行う．そして，推定した未定パラメータと上記の (1) 式と (4) 式を使い，増設時における Source node からの転送処理速度，Destination node への転送処理速度をそれぞれ推定し，それらの値から各転送処理時間の推定値を求める．そして，(3) 式から求めた合成処理時間の推定値を求め，上記の転送処理時間と合わせて VM ごとの処理時間の推定値を求める．VM 増設により変動しないことが観測された VM に対しては直前に観測された処理時間をそのまま用いる．これらの処理時間の推定値を使って，線形計画法により単位時間あたりの合成処理数が最大になるように，VM ごとに単位時間あたりに処理させる画像の枚数 (フレーム数) を計算する．この結果を用いて，システム全体の単位時間あたりに処理できるフレーム数を求め，それをシステム全体性能の推定値とする．

このように，VM 増設の影響を受ける VM とそうでない VM とを分離するのは，我々の提案している大容量計算システムは，複数のクラウドを使うことを前提としており，影響受けない VM も少なからず存在する．そのことから，影響を受けない VM を分けて直前の性能データを推定値として用いることとしている．このように求めたシステム全体性能の推定値にもとづいて，さらに増設が必要かどうかの判定を行い，所望の性能が得られるまで，繰り返し性能推定することによって，適切な数の VM の増設が可能になると考えられる．

4.6 他のアプリケーションへの適用に対する考察

今回処理性能測定を行ったアプリケーションは，処理を 3 分割したうちの最初の処理である VM へのデータ転送処理においてハードウェアリソースの争奪が行われ，Destination node への転送処理が終わるまで，最初にリソースを確保できなかった VM にはリソースが割り当てられなかった可能性がある．そのため，最初にリソースを確保できなかった VM については，Source node への転送処理で待たされた以外は競合がほとんど起きなかったと考えられる．そのため，我々が提案した方法は最初の処理で競合する場合には有効であるが，2 番目以降の処理で競合が起きる場合については考慮できていないと思われる．そこで，2 番目以降の処理で競合が起きた場合の VM 性能の推定方法について考察を行う．

まず，2 番目の処理において競合が起きるアプリケーションとして，例えば，少ないパラメータのみを送って新たな画像を生成する処理や，生成した画像の色変換やコンボリューショナルフィルタなど演算量の大きい画像フィルタリング処理，高い圧縮率での画像圧縮アプリケーションなどが想定される．VM 上の処理で競合する場合は CPU リソースを奪い合うことになるため，4.3 節と同様に (2) 式をベースに考えるのが妥当であると思われる．4.3 節での議論と異なるのは，演算量が大きくなると処理時間が長くなるためトラヒック密度である ρ が大きくなる，つまり ρ が 1 に近づくことである．そのため，ρ が 1 に近くなれば W の値は ρ^{s-2} に漸近すると考えられ，同時に処理を実行している VM 数を X とすると処理時間 $Y_4(X)$ は，未定パラメータ A_4 と B_4 を使って，数式

$$Y_4(X) = A_4 \times {B_4}^{X-2} \tag{5}$$

によって表すことができると考えられる．この式を使うことによって，新たに VM を追加した場合の処理時間の推定が可能になると考えられる．

次に，3 番目の処理である VM により処理された結果を転送する処理が競合する場合について考える．このアプリケーションの例としては，VM において比較的少ない処理量で，その処理結果として大量のデータが生み出されるようなアプリケーション，例えば，指定した色の画像を生成するようなアプリケーションが想定される．その場合は，Destination node のネットワークがボトルネックとなり，4.2 節の (1) 式が適用できると考えられ，その式を用いることによって，VM 追加時の処理時間を推定することが可能になると考えられる．

5. まとめと今後の課題

クラウド上の複数の VM を使った分散システムは，不定期に大容量計算を行う必要があるユーザにとって，大きなコストメリットをもたらすと考えられる．そのニーズを満たすために，我々は，ネットワーク上に分散している大量の VM を一時的に利用し，大容量計算を並列的にかつリアルタイムに行う動的並列分散処理システムとそのためのフレームワークを提案している．しかし，不特定の VM とハードウェアリソースを共有している VM の単体性能を正しく見積もるのは難しいため，システム全体の性能を正確に見積もるのは難しく，所望の性能を得るために必要な VM 数を正確に決定するのは困難である．さらに，性能不足により新たに VM を追加しようとすると，稼働中の VM とハードウェアリソース競合を起こし，既に稼働中の VM の性能まで低下させる可能性があるため，所望の性能を得るために追加が必要な VM 数を見積もるのも非常に困難である．そこで我々は，予め概算で構築したシステムをベースに，稼働中の VM 上で観測された処理時間データと試験的に実行した VM 増設時の処理時間データを用いて，増設によるシステム全体の処理時間の変動を推定し，その推定値をもとに増設する必要がある VM 数を正確に決定する方法について検討した．

本論文では，まず，我々が提案している動的並列分散処理フレームワークを使った非圧縮 4K 映像合成処理アプリケーションを例に，実ネットワークおよび実 PC サーバ上に複数の VM を構築し，同時に合成処理を実行する VM 数を変えながら，各処理時間がどのように変化するかを実測した結果を示した．この実験では，VM で実行される処理を，(1) 合成に必要な画像ファイルを Source node から Processing node（VM）への転送処理，(2)VM における合成処理，(3)VM から Destination node へ合成結果の転送処理に分割し，それぞれの処理に対して処理時間の計測を行った．これらの計測結果から，このアプリケーションにおいては (1) の転送処理においてリソース競合が多く発生することが確認された．

これらの計測結果にもとづき，(1) の処理に対しては有限のリソースを複数の VM で競合使用していると考え，VM への転送処理速度が反比例して減少していくモデル，(2) の処理に対しては処理時間が VM 数に対して比例する一次関数のモデル，(3) の処理に対しては VM の数 X に対して，処理速度が $A/X + B$ の関係で処理速度が減少していくモデルを提案し，処理ごとに提案モデルにもとづいて定式化を行った．さらに，これらの数式を用い，システム全体の性能を動的に推定する方法を提案するとともに，(2) および (3) の処理において競合が発生するアプリケーションに対して，どのように定式化すべきかについても考察を行った．今後，この提案方式を実装し，システム全体の性能を推定できるか，また，他のアプリケーションに対しても適用できるかを検証していく予定である．

今回の実験では，全ての VM の処理が完了してから，次の合成処理を行っているが，1 つの VM の処理が終わったら次の画像をその VM に対して送る場合でも，このモデ

ルが有効かどうかを合わせて評価していく必要があると考えている．さらに，様々なアプリケーションを使って性能を実測することによって 4.6 節の考察が正しいことを実証していく予定である．今回の実験では Hypervisor として KVM，OS として Ubuntu linux を使用したが，リソースの割り当て方法は Hypervisor や OS に依存することから，異なる仮想環境や OS 環境でも評価実験を行う予定である．今後，これらの評価を行い，複数の VM を動的に組み合わせることにより，様々な業務で利用可能な並列分散処理プラットフォームを完成させる予定である．

謝辞　本研究の一部は，2015 年度総務省委託研究 SCOPE「非均質計算機環境を使ったリアルタイム大容量データ処理アプリケーションプラットフォームの研究開発」(1501000004) の助成を受けて実施しました．

参考文献

[1] 総務省: 平成 30 年度版 情報通信白書”, (入手先 ⟨http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h30/pdf/index.html⟩) (2018.9.18).
[2] 君山博之，北村匡彦，小島一成，丸山充，藤井竜也: 大容量計算のための複数クラウドを使った動的並列分散処理フレームワークの提案，第 24 回マルチメディア通信と分散処理ワークショップ論文集，pp.126–133 (2016).
[3] Novakovic, D., Vasic, N., Novakovic, S., Kostic, D., and Bianchini, R.: DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments, Proc. 2013 USENIX Annual Technical Conference (USENIX ATC ’13), pp.219–230 (2013).
[4] Riskhan, B., and Muhammad, R.: Virtual Machine Performance Approaches in the Online Education System, Proc. International MultiConference of Engineers and Computer Scientists 2016 (IMECS 2016), Vol.1, pp.1–6 (2016).
[5] Chiang, R. C., Hwang, J., Huang, H. H., and Wood, T.: Matrix: Achieving Predictable Virtual Machine Performance in the Clouds, Proc. 11th International Conference on Autonomic Computing (ICAC ’14), pp,45–56 (2014).
[6] Chiang, R. C., and Huang, H. H.: TRACON: Interference-Aware Scheduling for Data-Intensive Application in Virtual Environments, Proc. 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’11), pp47:1-12 (2011).
[7] Kundu, S., Rangaswami, R., Dutta, K., and Zhao, M.: Application Performance Modeling in a Virtualized Environment, Proc. 2010 The Sixteenth International Symposium on High-Performance Computer Architecture (HPCA 16), pp.1–10 (2010).
[8] Wood, T., Cherkasova, L., Ozonat, K., and Shenoy, P.: Profiling and Modeling Resource Usage of Virtualized Applications, Proc. the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware ’08), pp.366–387 (2008).
[9] 北村匡彦，君山博之，澤邊知子，藤井竜也，小島一成，丸山充:SDN スイッチを使った動的分散処理方式の提案，信学技報，Vol.CQ2015-134, No.59，pp.147-151 (2016).
[10] Redis (入手先 ⟨http://redis.io/⟩) (2018.9.20).
[11] Ryu SDN Framework (入手先 ⟨https://osrg.github.io/ryu/⟩) (2018.9.20).
[12] Fluentd |Open Source Data Collector |Unified Logging Layer (入手先 ⟨https://www.fluentd.org/⟩) (2018.9.20).
[13] KVM (入手先 ⟨https://www.linux-kvm.org/page/Main_Page⟩) (2018.9.20)
[14] Open vSwitch (入手先 ⟨https://www.openvswitch.org/⟩) (2018.9.20)
[15] 吉岡良雄: 待ち行列と確率分布 -情報システム解析への応用-, 森北出版株式会社 (2004).