「マルチメディア，分散，協調とモバイル
(DICOMO2019)シンポジウム」 令和元年7月

複数クラウドを用いた大容量リアルタイム並列分散処理システムとその性能評価

君山 博之[1,†1] 丸山 充[2] 小島 一成[2] 藤井 竜也[3]

概要：計算機を所有することなく，利用したいときに利用したいだけ使うことのできるクラウドサービスが，一般企業を含む多くのユーザに活用されている．そのクラウドのメリットを最大限活かすために，我々は，複数のクラウド上の複数の VM を適応的に組み合わせることによって，完了時間にクリティカルな大容量計算を実行するためのフレームワークを提案している．そのフレームワークを実現可能かどうかを確認する目的で，フレームワークにもとづいた映像合成処理アプリケーションソフトウェアの実装を行った．汎用 PC サーバおよび学術研究ネットワークを用いて，幕張と石川の 2 箇所に構築した模擬クラウド環境を使って実証実験を行い，144 の VM を並列計算させることによって秒 44.1 映像フレームの合成処理ができることを実証した．さらに，このフレームワークを使ったシステムのスケーラビリティを確認するとともに，システム構築前の性能評価を可能にする手法の確立のため，シミュレーションプログラムを作成し，複数のクラウドを使った映像合成システムについて，その性能シミュレーション評価を行った．その結果，VM 数に応じてリニアに性能向上が図れることが確認された．一方で，シミュレーションによって導かれた性能絶対値については，実測値よりも約 50%高い値となり，今回，シミュレーションで考慮に入れられなかった TCP のフロー制御や VM 選択のオーバーヘッドなどを考慮に入れる必要があることが判った．

A large-scale real-time distributed parallel processing platform using multiple clouds and its performance evaluation

HIROYUKI KIMIYAMA[1,†1] MITSURU MARUYAMA[2] KAZUYA KOJIMA[2] FUJII TATSUYA[3]

1. はじめに

計算機を自ら所有することなく，利用したいときに利用したいだけ使うことのできるクラウドサービスが，一般企業を含む多くのユーザに活用されている．このサービスのメリットは，設備投資なく，使用料のみで保守コストを気にせずに計算機を利用できるところにある．そのため，個人ユーザだけでなく多くの企業がクラウドサービスを導入し，これまで自前で用意していた WWW サーバや，社員向けの業務用サーバ（例えばメールサーバ）の置き換えとしてこのサービスを活用している．しかし，このサービスの本質は “利用したいときに利用したいだけ利用できる” ところにあり，例えば，映像制作におけるレンダリング処理や事故や災害時のデータ解析・シミュレーションなど，定常的には発生しないが突発的に大量の演算を必要とする計算ニーズを充足するのに最適なサービスだと考えられる．

しかし，クラウドサービスの多くが計算機そのものではなく計算機上にソフトウェアにより構築された仮想計算機（VM）をユーザに貸し出すため，ユーザが借りた計算機は，“本物” の計算機よりも性能が劣る場合が多い．また，複数の VM が，1 台の計算機ハードウェアを共用することから，同じハードウェアを共用している他の VM の負荷によって，VM 自身の性能が変わる可能性がある．このことから，大容量計算を行った場合に何台の VM を組み合わせれば，所望の時間で計算が完了するかどうか予測できないという問題がある．

そこで，我々は，クラウド上の VM を用いて，終了時間にクリティカルで，かつ，独立して並列処理可能な大容量計算を容易に実行できる環境の実現を目指して，性能モニ

1 東京電機大学
2 神奈川工科大学
3 NTT 未来ねっと研究所
†1 現在，大同大学

タリングをしながら，複数のクラウド上の複数の VM を適応的に組み合わせることによって大容量計算を可能にするフレームワークを提案している [1]．このフレームワークでは，時々刻々と生成されるリアルタイムデータを処理すること，使用する VM の台数を適応的に変更可能にすることを考慮し，計算元のデータの一部をクラウドに転送し，計算をクラウドで実行し，その結果を結果を集約するサーバに送信する処理を繰り返す方式を採用している．

一般的にクラウド上で大容量計算を実行する場合には計算元データをネットワークを介してクラウドに転送する必要があるが，そのネットワークの容量がボトルネックになる可能性があること，また，同一クラウド内で VM を大量に使用すると VM 間でのリソース競合を起こし，性能が劣化する可能性があることから，このフレームワークでは複数のクラウドを利用できるようにしている．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特定データの抽出など，このフレームワークが適用可能な独立した並列計算によって解決できる問題は多く，幅広い応用が可能なシステムであると考えられる．

我々は，この提案フレームワークを使って，実際に非圧縮 4K 映像の合成処理を行うアプリケーションを実装し，144 台の VM を使った実証実験を通じて 1 秒あたり 44 フレームの速度で合成処理できることを実証し，本システムの有効性を確認した [1]．しかしながら，この実験結果は，VM 単体の性能評価実験結果から予想した性能を下回る結果であった．そこで，さらなる性能向上を目指し，我々は，このシステムのボトルネック箇所の調査を行った．その結果，単一物理サーバ上で複数 VM が同時に処理を実行した際に，VM 間でハードウェアリソース，特にネットワークリソースの取り合いが発生し，VM 単体で実行した場合よりも性能が劣化することを確認するとともに，競合する VM 数に対して VM 単体性能のどの処理がどのように劣化するのかを明らかにした [2]．

さらに，本システムのスケーラビリティの確認，および，システムを構築せずにシステム全体性能の評価を可能にすることを目的に，VM 単体の性能評価実験で得られた知見やデータを取り入れたシステム全体性能の事前評価シミュレーションツールを開発した．そして，複数のクラウド上の VM を複数使って，本フレームワークで分散処理システムを実現した場合のシステム全体性能を評価し，VM 数に応じて性能が拡張できることを確認した．

クラウドを利用した分散処理システムとしては Hadoop/MapReduce[3] が多く使われており，その改良に関する報告も数多くなされている．このシステムは，分散ファイルシステム環境と分散処理が一体となったシステムで，その環境に蓄積できれば効率良く短時間で分散処理ができるため，多くのいわゆるビッグデータを扱う企業が採用している．しかしながら，例えば映像制作の場合は，処理すべき映像データはカメラで撮影したデータであり，カメラ内のローカルストレージに入っている．そのため，Hadoop/MapReduce で処理する場合は，カメラ内の全ての撮影済み映像データもしくは編集済みのデータを，ネットワークを介して，一旦，その環境に全て転送してから処理をする必要があるため，我々の提案システムの方が処理時間の短縮が見込まれると考えられる．

本論文では，第 2 章では我々が提案してきている複数 VM を使った大容量計算フレームワークについて説明し，第 3 章ではそのフレームワークを使って実現したリアルタイム映像合成アプリケーションの予備評価実証実験とその結果，第 4 章では，VM 単体の性能評価結果にもとづいて開発したシステム性能を評価するためのシミュレーションプログラムの概要と，そのプログラムを使ったスケーラビリティ評価結果，および，プログラムの課題について示す．そして，最後の章にまとめと今後の課題を示す．

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 [1] 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，VM 上で動作させることのできる，Source node，Load balancer，Management node，Destination node から構成されている．計算したい元データを複数の Source node に格納し，その Source node から元データの一部を Load balancer を経由して，クラウド上に立ち上げた複数の Processing node の 1 台に転送する．そして，データを受信した Processing node は計算を実行し，計算結果を Destination node に転送する．Destination node は受信した計算結果を集約する．上記の処理を繰り返し実行し，複数の Processing node を使って並列的に計算させることによって，大容量計算を実現する．Management node は，Processing node での転送処理も含む処理時間全体をモニタし，モニタ結果にもとづいて Processing node 選択する．そして，選択した Processing node への転送を Load balancer に対して指示する．このシステムでは，Management node が，適応的に使用する Processing node 数を変更することによって，必要な計算量やデッドラインに応じた最適なシステムを動的に構成可能にしている．

図 1 に示した我々の提案システムでは，文献 [4] で提案している負荷分散メカニズムを採用している．このメカニズムを使うことによって，Source node は Load balancer 宛にデータを送信するだけで，Load balancer が Management node の指示により複数の Processing node に対して動的に負荷分散することが可能である．別な負荷分散方式として，Load balancer を使わずに Management node から，直接

Source node に対して送信先の Processing node を通知して負荷分散を行う方式があるが，その方法と比べて我々の提案システムは Source node を単純に実装することができる利点がある．ログを処理したり編集中の映像を処理することを考えると，Source node にはイントラネット内のログサーバやファイルサーバを適用することが想定され，OS やファイアウォールなどの設置環境によっては，Management node と通信が必要な専用のアプリを Source node に実装できない場合も想定される．そのような場合でも，我々の提案方式は，外部ネットワーク上の Load balancer へ HTTP を使って転送するだけのアプリケーションまたはスクリプトを Source node 上に実装するだけで実現が可能である．

さらに，この大容量計算システムを容易に実現できるように，我々は，図 2 に示すソフトウェア構成もあわせて提案している．その概要は以下の通りである．Source node，Processing node，Destination node には，データ送受信のための共通プログラムである Communication module（CM）を用意し，その上に各 node 固有の処理を Processing module（PM），Processing module for Destination node（PMD），および，Processing module for Communication node（PMC）として実装する．CM 同士の通信には，途中に NAT や Firewall があった場合でも通信できるように，TCP/IP を下位プロトコルに持つ HTTP を採用している．さらにその上位として REST API をベースにしたプロトコルを実装し，この CM によってプロトコル処理を実行する．また，Load balancer として，文献 [4] で提案しているように，OpenFlow Switch（OFS）を採用している．

OFS を使い，Management node 管理下の空いている VM と Source node との間に文献 [4] 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することによって，node の IP アドレスを変えることなく，そのオーバーレイネットワークを介して任意の Processing node にデータの転送を可能にしている．Management node，Source node，Processing node，Destination node には，Management node において負荷分散を行うための負荷情報や処理時間情報の収集と集約を行うソフトウェア（Load information collector）の導入も提案している．さらに，Management node には，収集した情報を蓄積するためのデータベース（DBMS），および，次に使用する Processing node を決定し，Load balancer に仮想的なオーバーレイネットワークを作成するための指示を行う Management module，および，Management module からの WebAPI による OFS の制御を可能にするための OpenFlow controller の導入もあわせて提案している．

3. 実装と予備評価実験

上記の提案フレームワークにしたがって並列計算アプリケーションが実現できること，および，設置場所の異なる

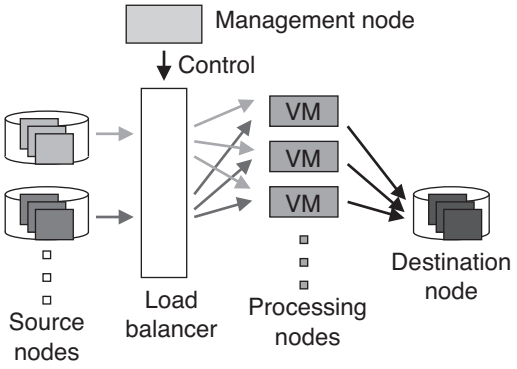

図 1 複数の VM を使った大容量計算システムの概略構成図

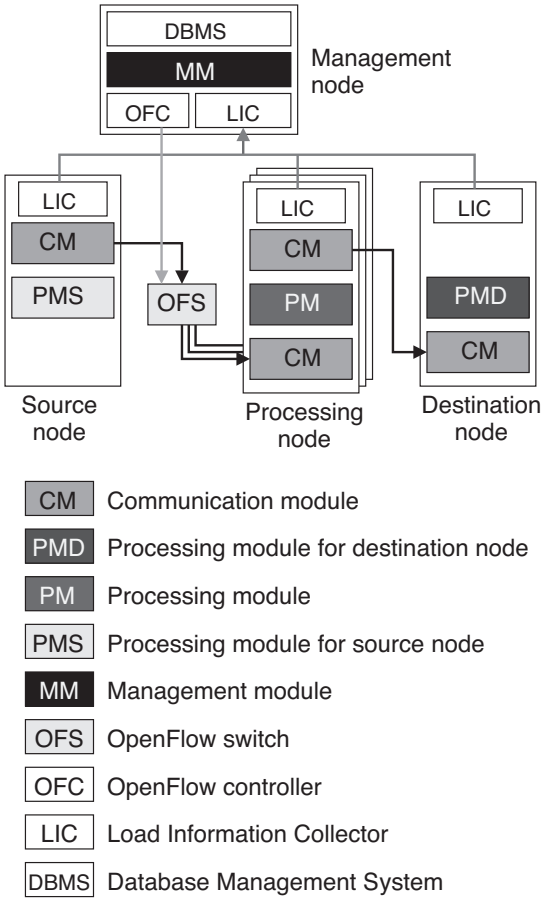

図 2 提案したソフトウエアフレームワーク

複数の VM を使用して並列計算ができることを実証するために，非圧縮 4K 映像の合成アプリケーションの試作実装とそのアプリケーションによる合成処理予備評価実験を行なった．以下の節では，実装方法およびこの評価実験についてその詳細を説明する．

3.1 実装

実装にあたっては，大容量並列計算アプリケーションを容易にできるようにすることを考慮して，Open source software（OSS）を活用して実装する方法を検討した，また，独自のハードウェアを使うことなく，全ての node を

Linux をインストールした汎用 PC サーバまたは VM を使って実現できることを条件に実装方法の検討を行った．

その結果，Management node においては，DBMS として OSS の Redis[5] を使用し，負荷情報や処理時間情報を収集するソフトウェアとして，Fluentd[6] を使用することとした．Redis を採用した理由は，メモリ上にデータベースを展開しており，データベースへの高速な書き込み，および，検索が可能であるからである．また，Management module として，各 VM の処理時間情報や負荷情報を元に VM を選択し，仮想的なオーバーレイネットワークの生成を指示するための HTTP メッセージを送信する制御ソフトウェアを開発した．

なお，本試作実装における Processing node の選択アルゴリズムとして，最初の一順は Processing node のリストに掲載されている順に選択し，その後は計算が終了したものから順番に選択するアルゴリズムを実装している．そして，Load balancer として，ソフトウェアにより OpenFlow switch 機能を実現しており広く利用されている Open vSwitch（OVS）[7] を，OVS を制御するための OpenFlow controller ソフトウェアとして Ryu[8] を採用した．また，Load balancer 用に，Management node からのメッセージを受信し，その指示を OVS のフローテーブルに書き込むための制御ソフトウェアを開発した．さらに，前述した HTTP によりデータの送受信を行う CM ソフトウェア，および，2 種類の非圧縮映像フレームを別々の Source node から受信し，1 フレームに合成し，その結果を Destination node への転送する PM ソフトウェアを，設計，開発した．PM ソフトウェアでは，CM 経由で受信した背景映像フレームデータと，アルファチャンネル（合成のためのマスク情報）が付加された前景映像とを合成し，1 枚の映像フレームに変換し，そして，CM 経由で送信する処理を実行する．CM に関しては，単純な API を持つ Shared object 形式として実装を行い，C または C++言語を使って PM の処理部分だけを実装すれば容易にアプリケーションを実現できるようにするとともに，再コンパイルすることなく処理の入れ替えも容易にできるようにしている．さらに，Destination node 用に，Processing node で処理された映像を受信し，特定のディレクトリに書き込む Processing module for Destination node も設計，開発した．Source node に関しては，前述した CM を使わずに curl[9] を用いて REST API により映像データを Load balancer に送る Shell scrip によって実現している．

3.2 予備評価実験

我々は，提案フレームワークにしたがって試作実装したシステムが設計通り動作することを確認するとともに，地理的に離れた複数のクラウド環境を使った場合でも，シームレスに分散処理が行えるかどうかを確認することを目的に，予備評価実験を実施した．実ネットワーク環境上に以下に示す評価システムを構築し，Interop Tokyo 2016 の期間中に，その会場の神奈川工科大学ブース内に設置したサーバ群をクラウドに見立てて実験を行なった．

3.2.1 実験システム構成

評価実験の全体システム構成を図 3 に，Interop Tokyo 会場内に設置したシステムの概観を図 4 に示す．この実験では，図 3 に示すように，Interop Tokyo 会場と NICT 北陸 StarBED 技術センター内のサーバ群をそれぞれ仮想的なクラウドとみなし，Interop Tokyo 会場内に，2 台の Source node，Management node，Load balancer，Destination node，80 の Processing node を設置し，NICT 北陸 StarBED 技術センター内に 64 の Processing node を設置した．2 台の Source node にはそれぞれ合成処理に必要な前景映像素材と背景映像素材を蓄積し，1 映像フレーム単位で Source node から Processing node へ送信を行うようにした．

この実験の Processing node として利用した PC サーバのハードウェアスペックおよび使用ソフトウェアを表 1 に示す．Interop Tokyo 会場側は HP 社 DL380p gen8 を 3 台，StarBED 側では Dell 社 PowerEdge C6220 を 4 台使用し，Interop Tokyo 側では 1 台の PC サーバあたり 32 または 16 の VM を構築し，NICT 北陸 StarBED 技術センター側では 1 台の PC サーバあたり 16 の VM を構築している．Processing node として使用した VM のスペックは，全て同一でメモリ 2GB，仮想 CPU 数は 2 である．Management node および Load balancer には，表 1 のスペックの HP 社 DL380p gen8 を使用し，Source node には Supermicro 社の PC サーバ，Destination node には HP 社の Z820 WorkStation を使用している．

図 5 に Interop Tokyo 会場内の接続配線，および，図 6 に NICT 北陸 StarBED 技術センター内の接続配線を示す．Interop Tokyo 会場内の Processing node はネットワークがボトルネックにならないように，10GbE 2 本で接続している．さらに，Source node や Destination node がボトルネックにならないように，それぞれ 40GbE で接続するとともに，映像を格納するストレージとして Source node には SSD 16 本から構成される RAID ストレージを，Destination node には RAM disk をそれぞれ用意している．

Interop Tokyo 会場内の全ての node は 40GbE L2 スイッチを介して接続され，合成処理を行う映像データは Source node から L2 スイッチ経由で Load balancer に送信され，会場内または StarBED 内の Processing node に送信される．Interop Tokyo 会場と NICT 北陸 StarBED 技術センターとの間は，Interop Tokyo 会場内のネットワーク（ShowNet）を介して，100Gbps の学術研究ネットワーク（R&E network）により結ばれている．なお，NICT 北陸 StarBED 技術センター側の Processing node は，配線の制約上，10GbE 1

本で接続し，そのため，ネットワークがボトルネックにならないよう 1PC サーバあたりの VM 数を Interop Tokyo 会場よりも少ない 16 としている．

表 1 Processing node スペック

Interop Tokyo 会場	
Model	HP DL380p gen8
CPU	Intel Xeon CPU E5-2630L 2.00GHz (6 Core) x 2
Memory	32 GByte
Network Interface Card	Intel X520 10G NIC (Dual port)
OS	Ubuntu server 14.05 LTS
Hypervisor	KVM
NICT 北陸 StarBED 技術センター	
Model	Dell PowerEdge C6220
CPU	Intel Xeon CPU E5-26350 2.00GHz (8 Core) x 2
Memory	128 GByte
Network Interface Card	Intel X520 10G NIC (Dual port)
OS	Ubuntu server 14.05 LTS
Hypervisor	KVM

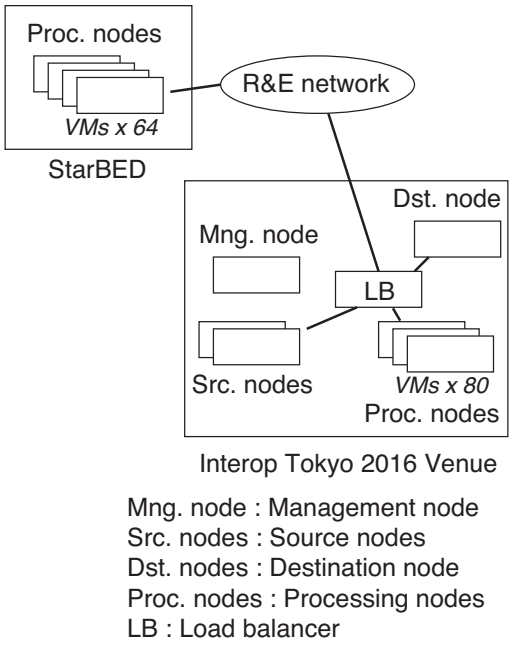

図 3 評価実験システム構成

3.2.2 評価実験結果

このシステムを使って，1,207 映像フレームから構成される映像データの合成処理を行った．使用した映像データは，水平方向 3,840 画素，垂直方向が 2,160 画素，RGB カラーで各色 8 ビットの解像度を持つ TIFF フォーマットの連番画像ファイルから構成されている．前述したように前景映像には RGB 以外に α チャンネルのデータが含まれており，映像フレーム 1 枚あたりのデータ量は前景が 33,177,600 バイト，背景が 24,883,200 バイトである．このシステム全体

図 4 実験模様

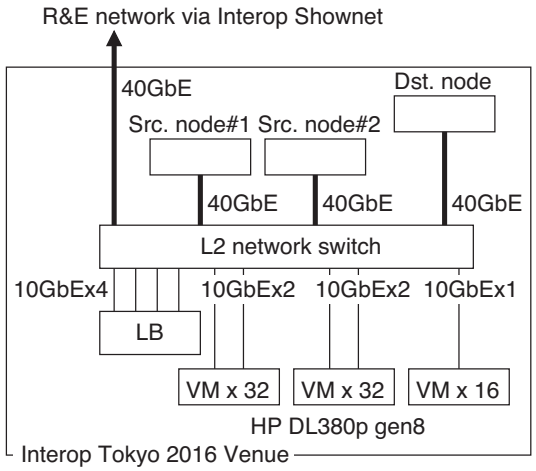

図 5 Interop 会場内システム詳細構成

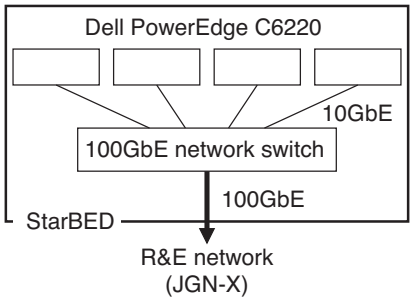

図 6 StarBED 内システム詳細構成

で合成処理された映像フレームの累積数の時間変化を図 7 に示す．この図の横軸は経過時間，縦軸は処理された映像フレームの累積数である．この図から判るように，時間を追ってほぼ直線的にフレーム数が処理されていっていることが確認できた．この実験の結果，1,207 映像フレームの処理を 27.3 秒で完了できることを確認し，フレームレートに換算して 44.1 フレーム毎秒の処理能力が得られたことを確認した．この速度は，秒 60 枚の 30 分の映像コンテンツであれば約 40 分で完了できる性能である．この映像の合成処理を一般的な映像制作ソフトウェアで実行すると 1 秒あたり 1〜2 フレームの合成速度であることから，十分

高速に合成処理が行えることを確認できた．

次に，単位時間あたりにシステム全体で合成処理されたフレーム数の時間変化を図 8 に示す．この図の縦軸は単位時間あたりに処理された映像フレーム数であり，横軸が経過時間である．点線は StarBED 内で処理された映像フレーム数，破線は会場内で処理された映像フレーム数，黒の実線はこれらの合計である．今回の実験では，1,207 フレームのうち 785 フレームが会場内の VM で，422 フレームを StarBED 内の VM で処理されていた．つまり，65%が会場内の VM で，35%が StarBED の VM で処理されたことになる．VM 数は会場内が 80，StarBED 内が 64 なので，割合にすると 56%が会場内，44%が StarBED 内に存在することになる．これらの結果から，Source node から遠い StarBED 内の VM よりも，近い会場内の VM の方がより多く使われたことが確認された．図 8 のグラフから判るように，本実験中に単位時間あたりの処理数が 2 回低下する時間があったことが観測されている．この原因として，落ち込んでいる時間が秒オーダーであることから，ノード間通信がバースト状のトラヒックとなり，ネットワークの帯域を超えたため再送信が起きている可能性がある．また，別な可能性として，今回の実験では計算ノードの VM を物理コア数以上，実メモリ以上に作成しているため，そのリソースが空くのを待っていることも考えられる．いずれにしろトラヒックデータ等を使った詳細な解析が必要であると考えられる．

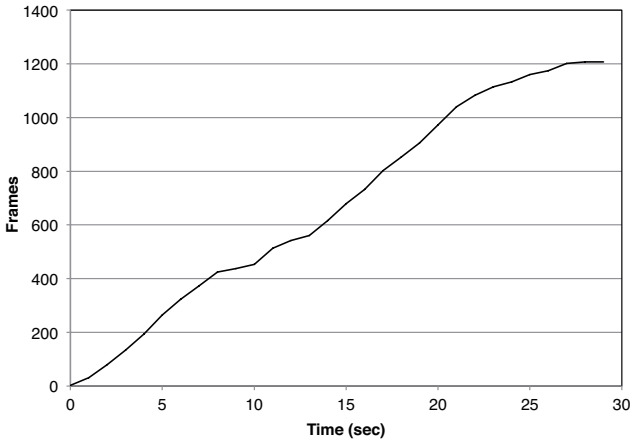

図 7 累積処理フレーム数の時間変化

4. シミュレーション評価

Processing node の数に対して，このシステムの全体性能がスケーラブルに向上できることが，このシステムの最も重要な特徴である．そこで，システム全体性能を評価するためのシミュレーションプログラムを作成し，評価を行なった．シミュレーションプログラムを開発するにあたっては，VM の単体性能が必要である．しかし，現時点で，VM の単体性能を記述する有効なモデルが存在しないことから，前述した合成処理のプログラムを使って VM の単体性能計測を行なった．そして，その結果にもとづいて VM の処理性能モデルを構築し，シミュレーションプログラムを設計，実装し，システム全体の性能評価を行なった．以下の節では，VM 処理性能モデルについて説明し，それにもとづいたシミュレーションプログラムの概要，そのプログラムを使ったシステム全体性能評価結果について説明する．

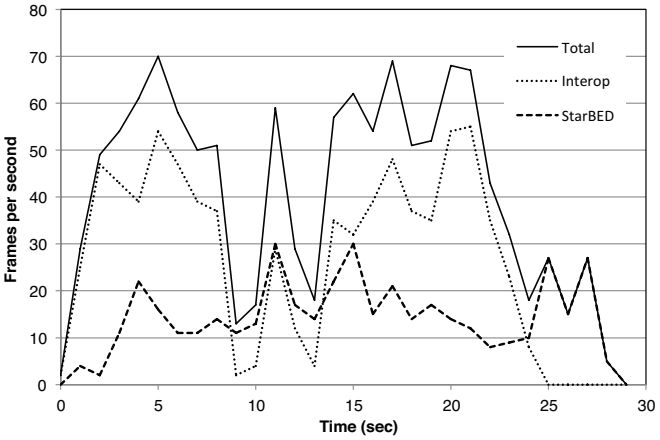

図 8 単位時間あたりの処理フレーム数

4.1 VM 処理性能モデル

このシステムにおける VM の処理は，(1) 計算前データの受信処理，(2) 計算処理，(3) 計算結果の送信処理の 3 つの処理に大別できる．我々は，同一物理サーバ上に構築した，Processing node を動作させた VM 数と上記の処理時間との関係を調査し，同時に稼働する VM 数に対して処理性能がどのように変動するかを実測により明らかにしている [2]．同時に動作する VM 数に対して，(1) の計算前データの受信処理が VM 数に依存して変化することが確認され，その平均処理速度が VM 数に反比例することが確認されている．また，(2) の計算処理時間，および，(3) の計算結果の VM からの転送処理の処理時間は，VM 数にはほとんど依存しないことが確認されている．

そこで，(1) の計算前データの VM への転送処理に関しては，同一物理サーバへ転送処理を行なっている VM 数によって，その物理サーバに接続されているネットワーク帯域が均等にシェアされるモデルを取り入れた．また，物理サーバまでの Round Trip Time（RTT）も考慮して，無負荷時の RTT の半分の時間をデータの転送時間に加算するようにしている．

(2) の VM での計算処理時間に関しては，1VM で合成処理を行ったときの処理時間計測によって得た実測値を用いた．図 9 に，表 1 に示した 2 種類の PC サーバで実測した処理時間確率分布を示す．この図の横軸は処理時間で，縦軸は 1msec 単位の処理時間に対する観測された確率を示し

ている．なお，この図中の VM1 は Dell PowerEdge C6220 で実測した値，VM2 は HP DL380p gen8 で実測した値である．シミュレーションでは，この分布を利用し，後述するように計算元データの受信が完了したときに乱数により合成処理時間を決定している．

(3) の計算結果の VM からの転送処理時間に関しては，[2] の VM 単体性能評価時には VM 数との依存関係は見られなかったが，転送処理自体が複数 VM で同時に行われていなかった可能性もあり，このシミュレーションでは (1) と同様に同一物理サーバにつながるネットワーク帯域を VM でシェアするモデルを導入した．Processing node（VM）の割り当ては，最大 1msec あたり 1 台までとし，処理が終了したものから順に割り当てる方式を採用した．

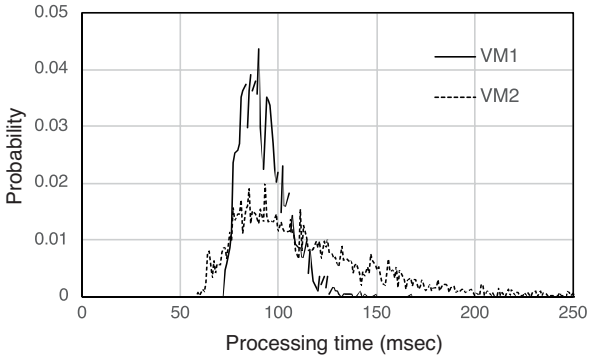

図 9 VM における合成処理時間分布

4.2 シミュレーションプログラムの概要

第 1 章で記述したように，我々は，複数のクラウド上に存在する VM も活用して，並列分散処理を実行可能な計算基盤の実現を目指している．そこで，我々は，複数のクラウド上に存在する複数の VM に対して，Source node から処理データを受信し，VM で処理を実行し，その後，Destination node へデータを送信する処理を模擬するシミュレーションプログラムを作成した．シミュレーションパラメータとして，クラウドの数，クラウドと Source node（Destination node）間の RTT，クラウド内の物理サーバの数，物理サーバ上の VM の数，VM における計算処理時間の確率分布，および，シミュレーション時間をパラメータとして与えられるようにした．また，VM の選択アルゴリズムも入れ替え可能とし，様々なシステム構成を模擬できるようにしている．

図 10 にこのシミュレーションプログラムの概略フローチャートを示す．このシミュレーションプログラムでは，モンテカルロシミュレーションの手法を導入し，時間経過に従って VM の状態を変化させることによって処理が完了した VM 数の累計（つまり，処理された映像フレーム数の累計）を計数し，その値から処理性能の評価を行なっている．VM の状態として，未割当（何も処理していない状態），受信中（Source node からデータを受信している状態），処理中（受信したデータを合成処理している状態），送信中（処理結果を Destination node に送信している状態）の 4 状態を定義して，1msec ごとに処理済みデータ量を計算し，1 映像フレーム分の処理が完了すると VM の状態を更新していき，Destination node への送信が完了した回数を計数する．また，上記のモデルを反映させるため，各時刻ごとに，同じ物理サーバのネットワークを使用している VM 数をカウントし，その数から 1msec 間に受信するデータ量を最初に計算し，その値をもとにその時間までに受信したデータ量を計算し，受信完了の判定を行なうこととしている．VM から Destination node への送信も同様である．初期値として設定したシミュレーション時間分，繰り返し実行して，合成処理が完了した回数の合計値を求め，性能値として単位時間あたりの合成数を求めるようにした．

4.3 シミュレーション結果

このプログラムを使って，単一クラウド上に物理サーバを複数用意し，物理サーバあたり 4VM を構築し，物理サーバ数を 1〜32 まで増加させた場合の合成処理性能評価結果を図 11 に示す．図の横軸は VM 数，縦軸は 1 秒あたりの平均合成処理数である．この図の値は，同じパラメータで 10 回シミュレーションプログラムを実行したときの平均値である．なお，このシミュレーションでは，すべての VM の合成処理時間の確率分布として図 9 の VM1 の分布を使用し，物理サーバのネットワーク帯域として 1Gbps を設定している．この図から，VM 数を増やすことによって合成処理性能を線形に増加可能であることが確認できた．

図 12 にクラウドを 2 箇所用意し，1 箇所のクラウドは VM 数を固定し，もう 1 箇所のクラウドの VM 数を変えた場合の合成処理性能評価結果を示す．図 11 と同様に，図の横軸は VM 数，縦軸は 1 秒あたりの平均合成処理数である．各クラウドごとに用意した物理サーバの台数および物理サーバあたりの VM 数，各物理サーバの最大ネットワーク帯域，RTT を表 2 に示す．このシミュレーションでは，VM の合成処理時間の確率分布として図 9 の VM1 の分布を双方のクラウドの VM に適用した．この結果から，本システムは複数のクラウドを用いた場合でも，VM の追加によってシステム全体の処理性能を線形に増加させることが可能であることが確認できた．

しかしながら，図 12 において，クラウド 2 の VM 数が 48 の場合（つまり，全 VM 数が 144 の場合），1 秒あたりの合成処理数は 67.3 フレームとなり，VM 構成に若干の差異はあるものの前章で示した実機を使った実験結果の 44.1 フレーム/秒よりも約 50%高い値となった．このシミュレーションプログラムでは，ファイル転送に利用している TCP のフロー制御，Source node や Load balancer の性能限界

などが考慮されていない．また，1msec あたり VM の割り当て数を 1 としたこと，つまり，VM の割り当て処理時間を 1msec としたことによって，全 VM がほぼ常時稼働する状況となったためシミュレーション結果として，実験データよりも高めの結果が導出された可能性があると考えられる．フロー制御の導入により転送速度低下が発生し，また，割り当て処理時間が増えることによって未割り当て状態の VM 数を増加させることとなり，いずれもシステム全体の性能は今回の結果よりも低くなると考えられる．今後は，フロー制御を導入するとともに，現実に近い VM 割り当て処理時間を導入することによって，シミュレーションの精度の向上を図り，システム構築時の事前の評価ツールとしても活用できるように，整備していきたいと考えている．

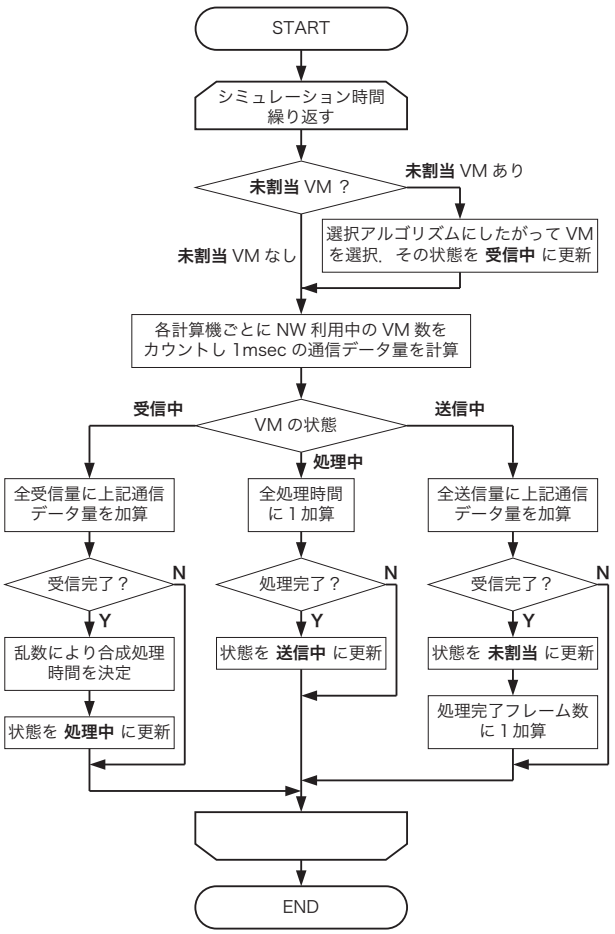

図 10 シミュレーションプログラムの概略フローチャート

5. まとめ

計算機を利用したいときに利用可能なクラウドサービスを複数利用して，一時的に発生する大容量計算を並列分散的に実行可能な環境の実現を目指し，そのためのフレームワークを提案した．このフレームワークとして，我々は，計算前のデータを格納する複数の Source node，クラウドの複数の VM 上に展開し，計算処理を実行する Processing node，Processing node の利用状況をモニタし，割り当てを行う Management node，Source node から Processing node へ仮想的なオーバーレイネットワークを実現する Load balancer，計算結果を集約する Destination node から構成されるシステムを提案した．さらに，Open source software を活用した各 node を実現するためのソフトウェアの構成法も併せて提案した．

このフレームワークの実現性を確認するため提案方式のもとづいた実装を行い，実サーバと実ネットワークを使用して，Interop Tokyo 2016 において公開実験を実施した．Interop Tokyo 会場および NICT 北陸 StarBED 技術センターの 2 箇所に分散配置した 144 の VM を使って，ハイビジョンの 4 倍の解像度を持つ非圧縮 4K 映像の合成処理を行い，1 秒あたり 44.1 映像フレームの合成処理ができることを確認し，このフレームワークの有効性を実証した．

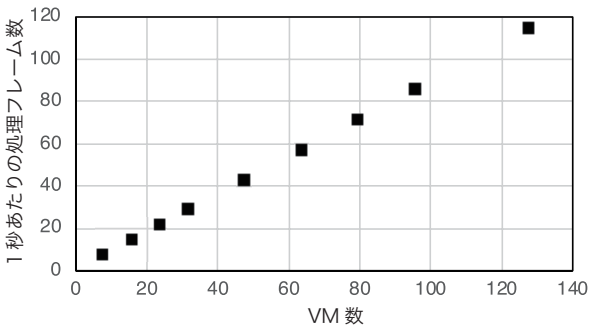

図 11 単一クラウドを使用した合成処理性能シミュレーション結果

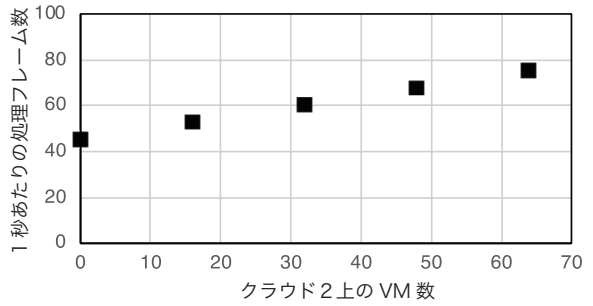

図 12 複数クラウドを使用した合成処理性能シミュレーション結果

表 2 シミュレーションで使用したクラウドのスペック

クラウド 1	
物理サーバ台数	12
サーバあたりの VM 数	8
全 VM 数	96
最大ネットワーク帯域	1 Gbps
RTT	0 msec
クラウド 2	
物理サーバ台数	0〜8
サーバあたりの VM 数	8
全 VM 数	0〜64
最大ネットワーク帯域	1 Gbps
RTT	30 msec

さらに，実 VM を使った処理時間実測結果にもとづいたシミュレーションプログラムを開発し，このシステムのスケーラビリティを検証した．その結果，VM の数の増加に対して，スケーラブルにシステム全体の処理性能を向上させられることを確認した．しかしながら，シミュレーションプログラムにより算出したシステムの処理性能の絶対値と，実計算機とネットワークで構成した環境において計測された処理性能の間には，約 50%の差があった．この差の解消に向けて，このシミュレーションプログラムで考慮していなかった，精度の高い VM の割り当て処理時間の導入，TCP フロー制御の導入，Source node，Destination node の限界性能の導入を行い，システム全体性能の事前の評価ツールとして活用可能なプログラムにしていく予定である．

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た．

参考文献

[1] 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).
[2] 君山博之，小島一成，丸山充，藤井竜也: 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，第 26 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.80–89 (2018).
[3] Apache Hadoop, 入手先 ⟨https://hadoop.apache.org/⟩ (参照 2019.5.8).
[4] 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).
[5] Redis, 入手先 ⟨https://redis.io/⟩ (参照 2019.5.2).
[6] Fluentd |Open Source Data Collector |Unified Logging Layer, 入手先 ⟨https://www.fluentd.org/⟩ (参照 2019.5.2).
[7] Open vSwitch, 入手先 ⟨https://www.openvswitch.org/⟩ (参照 2019.5.2).
[8] Ryu SDN Framework, 入手先 ⟨https://osrg.github.io/ryu/⟩ (参照 2019.5.2).
[9] :// curl, 入手先 ⟨https://curl.haxx.se/⟩ (参照 2019.5.4).