

複数クラウドを使った大容量リアルタイム並列分散処理フレームワークの提案と評価

君山 博之¹ 丸山 充³ 小島 一成³ 藤井 竜也²

概要：必要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。我々は、大容量リアルタイム処理を実現可能な、複数のクラウド上の VM に対するモニタリングとの組み合わせで適応的に負荷分散する仕組みと、様々なアプリケーションを簡単に実装できるソフトウェア構成などを含む大容量リアルタイム並列分散処理フレームワークを提案している。さらに、このフレームワークを実利用するために必要となる、VM のリソース共有状態を考慮したシステム全体性能評価のためのシミュレーション法もあわせて提案している。非圧縮 4K 映像合成アプリケーションを例に提案シミュレーション法を使ったシミュレーションを行い、VM 数を変えた時の性能のスケラビリティの確認、クラウドまでの RTT による性能変動や複数クラウドを使った時のスケラビリティの確認を行い、十分なスケラビリティがあることを確認した。このフレームワークを使った非圧縮 4K 映像の合成処理アプリケーションを実装を行い、北陸 StarBED 技術センター内の 64 台の VM と Interop 会場の 80 台の VM とに処理を動的に分散することによって、秒あたり 44.1 フレームの合成処理ができることを確認した。加えて、この実証実験システム性能を上記のシミュレーションプログラムを使って評価し、5%差で実験システムの性能を評価することができることを実証し、このシミュレーション法および提案フレームワークの有効性を実証した。

1. はじめに

近年、クラウドサービスが、企業から一般ユーザまで幅広く利用されるようになってきている。初期導入コストも維持コストも不要で、使いたいときに使いたい時間だけ利用できるクラウドサービスを導入するユーザは、年々増え続けている。実際、令和元年度版の総務省情報通信白書 [1] によれば、企業の 58.7% がクラウドサービスを導入しており、その導入割合も年々増加していることが報告されている。一方で、日々生成されるデータ量も、近年、急激に増加している。例えば、ハイビジョンの 4 倍以上のデータ量をもつ 4K や 8K 映像による放送が始まり、また、防犯のための監視カメラの急増 [2]、モニタリング用途での IoT デバイスの急増も見込まれており [1]、処理すべきデータ量が爆発的に増加していることは想像に難くない。データ量が増大していく中で、映像制作におけるレンダリング処理

や、事件発生時の防犯カメラ映像の解析、AI 導入時の学習処理など、イベント発生時に一時的に大容量の計算が必要な業務が多く存在する。そのような業務に対して、クラウドサービスを活用できれば、低コストでこのような業務に参入できるようになり、産業の活性化につながると考えられる。

一般的に、クラウドサービスにおいてプロバイダから提供されるのは、物理サーバではなく仮想マシン（以下、VM）である。VM は物理サーバのハードウェアリソースの一部を他の VM と共有しており、他の VM の負荷によって VM の性能が低下することが指摘されている [3]。そのため、短時間で計算結果を導き出すことを目的に、複数の VM を使用し並列処理により計算を実行する場合、VM ごとに性能が異なるだけでなく、ハードウェアの共有状態によって VM の性能も変動するため、その変動を考慮した適応型の並列分散処理が必要となる。

そのようなクラウド利用ニーズに応えるため、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、並列分散リアルタイム処理を可能にするためのフレームワークを提案している [4]。我々が提案しているフレームワークは、システム構成法とソフトウェア実装法から構成されている。このフレームワークでは、クラウドまでのネット

¹ 大同大学
Daido University, Nagoya-shi, Aichi 457-8530, Japan

² NTT 未来ねっと研究所
NTT Network Innovation Laboratories, Yokosuka-shi, Kanagawa 239-0847, Japan

³ 神奈川工科大学
Kanagawa Institute of Technology, Atsugi-shi, Kanagawa 243-0292, Japan

ワークがボトルネックとなりやすいため、複数のクラウドの複数の VM を使った並列分散処理を可能にしている。我々は、このフレームワークにもとづいて映像合成処理システムを実現し、幕張メッセおよび NICT 北陸 StarBED センタに配置した合計 144 の VM を使用し、ハイビジョンの 4 倍の空間解像度を持つ圧縮されていない 4K 映像フレームを、1 秒あたり約 44 枚合成できることを実証している。

さらに、構築前のシステムの全体性能評価や、システムへの VM 追加、削減に伴う全体性能の増減量を評価するための性能シミュレーション法を提案した。このシミュレーション法では、ハードウェアリソース共有状態による VM 性能変動を考慮することによって、共有状態を考慮したシステム性能評価を可能にしている [5][6]。しかし、前述の実証実験で得られたシステム性能とシミュレーションで得られたシステム性能との間には 50%以上の乖離があり、シミュレーション精度の向上が課題となっていた。

そこで、文献 [6] では考慮されていなかった、TCP フロー制御による通信性能の低下や処理データを VM に送信するための Source node の性能限界の影響をこのシミュレーション法に導入し、シミュレーションで得られたシステム性能と実験システムの性能とが 5%以内で一致することが確認できた。

本論文では、第 2 章において、我々が提案しているフレームワークの概要を説明し、第 3 章では、新たなシミュレーションプログラムによる性能評価シミュレーション結果について説明する。第 4 章では、本フレームワークを元に開発した映像合成処理システムについて説明し、実 PC および実ネットワークを使った実証実験の概要とその結果を説明する。さらに、シミュレーション結果との比較と考察を行い、最後の章でまとめを行う。

2. 提案フレームワーク

2.1 対象とする大容量計算

我々が提案するフレームワークを説明する前に、本フレームワークが対象としている大容量計算について簡単に説明する。まず、この提案フレームワークが対象とする処理は、処理する元データが 1～数カ所に保存されており、その元データが分割可能で、分割単位ごとに独立して処理可能なデータ処理とする。例えば、大規模ログからのキーワード抽出処理、監視カメラ映像からも特定の物体が映っているシーンの抽出処理、映像の合成処理などがこのフレームワークの対象とする処理である。有限要素法を使った流体解析のように処理を独立させることが難しい問題に対しては、本フレームワークの適用は難しいが、インシデント発生時のデータ解析など適用範囲は幅広いと考えている。

また、我々が提案しているフレームワークでは、処理前

のデータがネットワーク上の特定の計算機に、ある程度まとまって蓄積されていることも前提条件としている。例えば、ログ解析の場合は解析対象のログの蓄積されているサーバに蓄積され、映像合成であれば合成前のキャラクターの映像を制作する映像制作会社のサーバに、また、背景画像を制作する映像制作会社のサーバにも蓄積されていることを前提としている。この条件は適用対象のデータ処理ユースケースを考えれば自然な条件であると言える。

2.2 システム構成と負荷分散方式の提案

我々が提案しているフレームワークのシステム構成を図 1 に示す [6]。このシステムは図中の複数の Source node に処理前データを蓄積し、そこから Processing node (VM) へデータを転送し、並列的に処理を実行させ、処理結果を Destination node へ転送し、結果を集約させるシステムとなっている。このシステムでは、負荷分散を制御する Management node が VM やネットワークの負荷状態を常時モニタリングすることによって、処理能力が異なる VM を複数組み合わせる場合でも、効率よく負荷を分散できるようにしている。クラウドを使った分散処理システムとして、Apache Hadoop[8] のような分散システムを構築し、そこに予めデータをアップロードして MapReduce 方式 [9] により分散処理を実行する方式がある。しかし、この方式を使う場合、データを一括してアップロードする必要があるとともに処理をアサインする VM を適応的に変えることが難しいため、我々は処理する VM を動的に選択し、転送しながら処理を実行する方式を採用している。

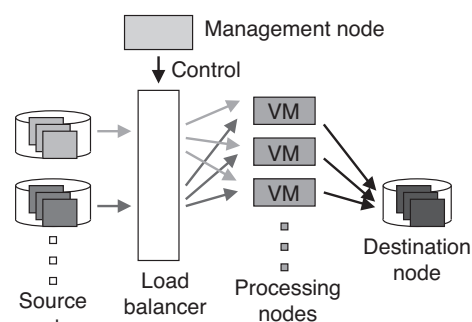


図 1 提案システムの概略構成図

本提案システムでは、文献 [7] で提案している負荷分散方式を採用している。この方式は、図中の Management node が処理を実行させる VM を決定し、その決定にしたがって Load balancer がデータ転送を実行する方式である。そのため Source node は、送信先 VM を意識することなく Load balancer 宛にパケットを送信するだけで、Management node が選択した VM にパケットを送信することが可能となっている。つまり、Source node に専用のアプリケーションをインストールすることなく、一般的な

ファイル転送アプリケーションがあれば、Source node を簡単に実現できることを意味している。異なる別な負荷分散方法として、Management node が、直接 Source node に対してどの VM へ転送するかを指示する方法がある。この負荷分散方式では、Management node と通信するためのインターフェースを用意し、その指示にしたがって送信先 VM を変えることのできるファイル転送アプリケーションを専用に開発する必要がある。このことは、Source node に一般的なファイルサーバを適用するのを困難にするため、我々は Load balancer を使った方法を選択した。なお、この提案負荷分散方式では、Load balancer として汎用の OpenFlow スイッチが適用可能である。

Source node, Processing node, Destination node 間の通信プロトコルには、HTTP (Hypertext Transfer Protocol) をベースとした独自の REST (Representational State Transfer) API[10] を定義している。REST API を使用する理由は、処理前のデータが格納している計算機 (例えば、ログサーバや映像素材サーバ) は、一般的にファイアウォールの内側であるイントラネット内に設置されていることが多く、TCP を下位プロトコルとして利用している HTTP を用いることは、Source node がセキュアな環境下にあっても本システムが利用可能になるからである。さらに、前述した負荷分散方式との併用によって、専用のソフトウェアを使わずに curl[11] などのオープンソースソフトウェア (OSS) を使った Shell script などのスクリプトのみでの Source node の実装を可能にしている。以上のように、本システムは新たに Source node 設備を開発、設置しなくても、クラウドの手軽さをそのままに、簡単にシステム導入できるように設計されている。

2.3 ソフトウェアフレームワーク

次に、このシステム構成上でアプリケーションを実現するためのソフトウェアフレームワークについて説明する。図 2 に提案するソフトウェアフレームワークを示す。このソフトウェアフレームワークでは、本システムを、Communication module, Processing module, Processing module for source node, Processing module for destination node, Management module, および、負荷情報を蓄積するための OSS の DBMS である Redis[12] ソフトウェア、負荷情報を収集するための OSS である Fluentd ソフトウェア [13]、OpenFlow スイッチを制御するための OSS の Ryu OpenFlow controller ソフトウェア [14] から構成することを提案している。Communication module は、前述した REST API を用いて node 間の通信を実行するためのソフトウェアであり、全 node 間の通信に利用する。また、Communication module は Shared object として実装され、アプリケーションが変更になった場合でも、Communication module をリンクするだけで再利用可能

としている。Processing module, Processing module for source node, および、Processing module for destination node は、各 node 固有の処理を実行するソフトウェアモジュールである。例えば、映像合成アプリケーションを実現する場合は、以下のような機能を実装する。まず、Processing module for source node には順番に合成する映像をストレージから読み出し、Communication module を介して、Processing node へ転送する機能を実装する。Processing module には受信した映像をデータを合成し、Destination node へ合成結果を送信する機能を実装する。そして、Processing module for destination node には、受信した合成結果を順に保存する機能を実装する。

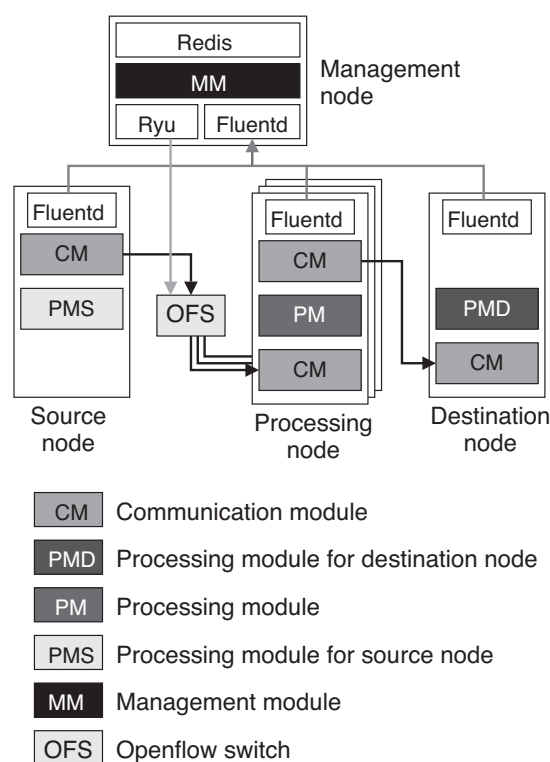


図 2 ソフトウェアフレームワーク

3. 性能シミュレーション

この提案システムの全体性能を事前に評価するだけでなく、リアルタイム性維持のための適応制御には VM の増減による性能変動を計算によって評価する環境が必要がある。そこで、我々は、このシステムの性能をシミュレーションする方法について検討を行った。シミュレーション法を確立するにあたっては、同一ハードウェア上で実行されている VM の負荷状態によって、個々の VM の性能がどのように変動するかモデルが必要である。そのモデルの構築のため、我々は後述する映像合成アプリケーションを使って VM の負荷状態と VM 性能の関係を実測し、そのモデル化を行った [5]。以下の節では、VM 性能評価実験結果とその結果から求められる VM の処理性能モデルに

ついて記述し、そのモデルを使ったシミュレーション法の概要と代表的なシミュレーション結果について説明する。

3.1 VM の処理性能モデル

映像合成アプリケーションでは、処理全体を、(1)Source node からの映像データ受信処理、(2)映像合成処理、(3)Destination node への合成結果の送信処理に三分割することができる。そこで、我々は、VM 性能の変動を (1)~(3) の処理時間の変動とみなし、処理実行中の VM 数とのその処理時間の関係を評価した [5]。文献 [5] で報告しているように、(1)~(3) の処理時間はある程度の変動幅を持っているものの、(1) の Source node からの映像データ受信処理時間の平均値が VM 数と比例すること（つまり、受信の平均速度が VM 数に反比例すること）が確認された。また、(2) の映像合成処理時間の平均値、(3) の Destination node への合成結果の送信処理時間の平均値は、処理実行中の VM 数にはあまり影響されないことも確認された。その理由としては、(2) の映像合成処理は主に CPU コアのみを使用するため、VM 間で CPU コアを取り合わない限り性能が変化しないと考えられる。そして、(3) の Destination node への合成結果の送信処理に関しては、(1) や (2) の処理時間の変動により、(3) の処理が実行されるときには、複数の VM で同一のハードウェア（この場合はネットワークインタフェース）を同時に利用する確率が小さく、他の VM の影響がほとんどなかったためと考えられる。我々は、これらの結果を元にシミュレーションの実装法を検討し、シミュレーションプログラムの開発を行った。

3.2 シミュレーション法の概要

このシミュレーションでは、時時刻々と変化する複数の VM の状態に依存する VM 性能変動を、ダイナミックに反映させる方法を導入する必要がある。そこで、我々は非常に短い時間間隔のタイムスロットを設定し、各タイムスロットごとに VM の状態を検査し、同一ハードウェアを利用している VM 数を求め、その数からタイムスロット内で処理できるデータ量を求め、全てデータが処理できたら処理完了とする方式を提案した [6]。その概略フローチャートを図 3 に示す。まず、VM 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、各状態においてタイムスロット内で処理できるデータ量を計算し、全てのデータの処理が完了したら次の状態に移移する方式を採用した。具体的には、「受信中」状態の場合のみ同一ハードウェア上で「受信中」状態にある VM 数を考慮し、その VM 数に応じてタイムスロット内で処理できるデータ量を増減させることによって、実行中 VM 数に応じた処理能力の増減を

再現した。また、合成処理時間に関しては、「受信中」から「処理中」に移移するときに実測した処理時間分布からランダムで求めることとし、求めた処理時間経過後に、送信中に移移することとした。「送信中」状態においては、同時に VM 数とは関係なく、タイムスロット時間で転送可能なデータ量を計算し、全てのデータの送信が完了した後、「未割当」状態に移移することとした。

さらに、文献 [6] の実装に、「受信中」および「送信中」状態においては、VM 数から計算される最大利用可能帯域、シミュレーションパラメータとして与える VM の TCP Window size と物理サーバとの間の Round Trip Time (RTT) から実効レートを計算し、タイムスロット内で処理できるデータ量を実効レートから導出するプログラムを加えるとともに、単位時間あたりに「未割当」から「受信中」に移移できる VM 数も設定できるように改良した。これらの処理を加えることにより、文献 [6] よりも、さらに実際のシステムに近いシミュレーションを実行できるようにした。なお、このシミュレーション法は映像合成処理に特化したものではなく、処理時間分布があれば他の処理にも適用可能な方法となっている。

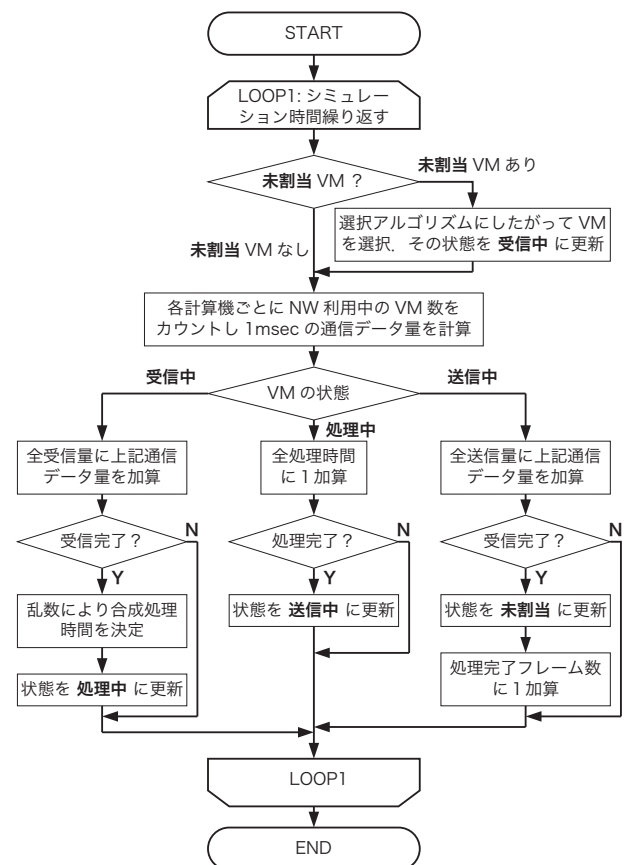


図 3 シミュレーションプログラムの概略フローチャート

3.3 シミュレーション結果

前述したシミュレーション法に従ってシミュレーション

プログラムを開発し、このシステムで非圧縮 4K 映像の合成処理を行った場合のスケラビリティの検証を行った。この検証では、合成処理時間分布として文献 [6] に示した無負荷時の実測データを用いている。図 4 に、単一クラウド上に構築した複数の VM を用いて合成処理を行った場合の単位時間あたりに合成できる映像フレーム数と VM 数の関係を示す。図の縦軸がシステム全体で 1 秒あたりに合成できた映像フレーム数、横軸が同時稼働する VM 数である。このシミュレーションでは、クラウド上の物理サーバあたりの利用可能帯域を 1Gbit/s, RTT を 2msec, 1 物理サーバあたりの VM 数を 8 としている。この章で示すシミュレーションでは、Source node の最小リクエスト間隔を 1msec に設定し、TCP の Window size は 512KByte, タイムスロットを 1msec, シミュレーション時間 1000 秒として評価を行っている。この評価結果から、VM 数を増やすことによって単位時間あたりに処理できる映像フレーム数がスケラブルに増加していくことが確認された。

図 5 に、物理サーバまでの帯域を変えた場合の単位時間あたりのシステム全体の合成処理数と、VM 数との関係を示す。このシミュレーションでは、RTT を 2msec, 1 物理サーバあたりの VM 数を 8, 物理サーバまでの利用帯域を 100Mbit/s, 1Gbit/s, 10Gbit/s に変化させて計算を行っている。この結果から、利用可能帯域の増減によってシステム全体性能が増減することが確認できた。

図 6 に、1 物理サーバあたりで動作する VM 数を変えた場合のシステム全体性能を示す。この図の縦軸がシステム全体で 1 秒あたりに合成できた映像フレーム数、横軸が 1 物理サーバあたりの VM 数である。このシミュレーションでは、物理サーバまでの利用可能帯域を 1Gbit/s, RTT を 2msec, VM 数を 96 に固定し、1 物理サーバあたりの VM 数を変えた場合のシステム全体性能を評価している。この結果から、1 物理サーバあたりの VM 数を増やすと、システム全体性能が減少することが確認できた。これは、VM の処理時間のほとんどが通信時間に費やされているため、1 物理サーバ上で動作する VM 数を増やすことによって 1VM に割り当てられる帯域が減少し、その結果データ転送にかかる時間が増加するものと考えられる。

図 7 に、クラウドまでの RTT を 0msec から 30msec まで変えた場合のシステム全体性能の変化を示す。このシミュレーションでは、物理サーバまでの帯域を 10Gbit/s, 1 物理サーバあたり 8VM として 96VM を使った場合のシステム全体性能を評価した。この結果から、RTT の増加で急激に性能が減少していくことが確認された。上記のシミュレーション結果から、映像合成のような使用するデータが大きい処理の場合、なるべく近いクラウド上の VM を利用するとともに、距離が同じであれば実効帯域が広いクラウド上の VM を利用することが、高い処理性能を確保するためには重要であることが確認された。

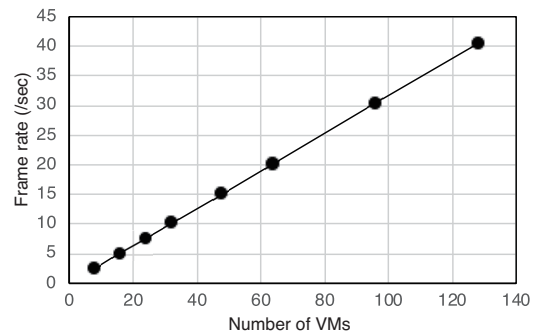


図 4 単一クラウドを使用した合成処理性能シミュレーション結果

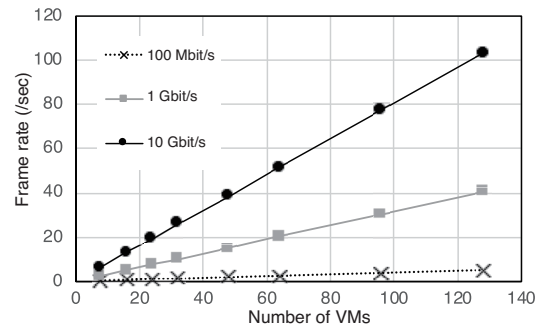


図 5 クラウドまでの利用可能帯域を変化させた場合の合成処理性能

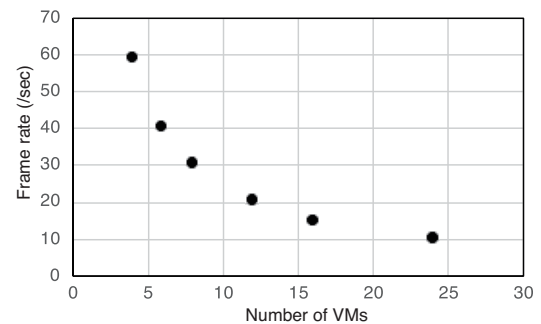


図 6 1 物理サーバあたりの VM 数を変化させた場合の合成処理性能

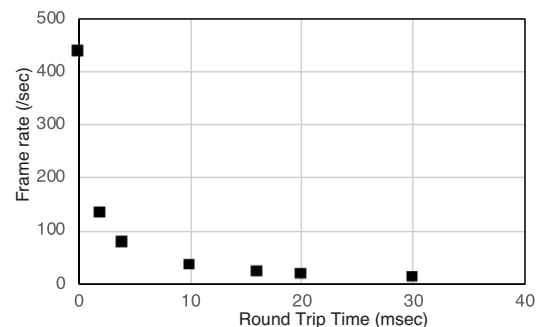


図 7 RTT を変化させた場合の合成処理性能

次に、複数のクラウドを使った場合の代表的なシミュレーション結果を示す。図 8 に 2 箇所のクラウドに VM を配置し、一方のクラウド（クラウド 1）は 96VM を固定とし、もう一方のクラウド（クラウド 2）内の VM 数を変化させた場合の単位時間あたりの合成処理フレーム数の変化を示す。この図の縦軸は、システム全体で 1 秒あたりに合

成できた映像フレーム数、横軸がクラウド2のVM数である。表1に詳細なシミュレーション条件を示す。また、クラウド1のVMまでのRTTは10msecとしたが、クラウド2のRTTは16msecと30msecに変えてシミュレーションを行っている。この結果、クラウド2側のVM数を増加させれば性能向上が図れるものの、遠い場所のクラウドのVMを使用すると性能向上の割合が下がることも確認された。なお、上記の評価では、RTTの短いクラウド上のVMを優先に選択する負荷分散アルゴリズムを用いているが、全てのシミュレーションにおいてVMの使用率は100%となり常に空きVMがない状態であったため、VM選択アルゴリズムを変えてもシステム全体の処理性能には大きな変化はなかった。

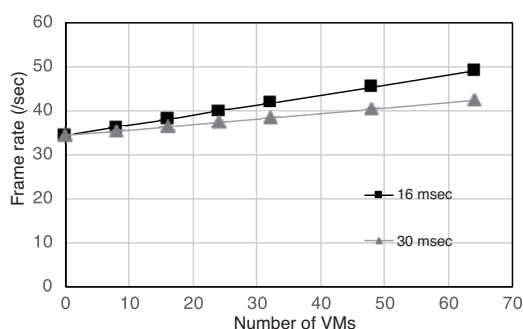


図8 複数クラウドを用いた場合の合成処理性能

表1 シミュレーションで使っ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
物理サーバあたりの帯域	10 Gbit/s
RTT	10 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
物理サーバあたりの帯域	10 Gbit/s
RTT	16 または 30 msec

4. 実証実験

提案フレームワークを使って、実際に大容量計算ができることを実証するため、映像合成処理アプリケーションの実装を行い、Interop Tokyo 2016の会場で実証実験を行った。以下の節では、この実証実験の概要とその結果について記述し、前章のシミュレーションプログラムを用いた評価結果との比較について述べる。

4.1 実証実験の構成と結果

実証実験の全体システム構成を図9に示す。この実験では、図9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となるVMを配置し、NICT北陸StarBED技術センター内に64のVMを配置した。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から各Processing nodeへ送信するようにした。

この実験でProcessing nodeとして利用したPCサーバのスペックを表2に示す。Interop Tokyo会場側では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物理サーバあたり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2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10にInterop Tokyo会場内の接続配線、および、図11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をそれぞれ使用している。

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を経由して、100Gbit/s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、配線の制約のため10GbE 1本で接続したため、ネットワークがボトルネックにならないよう1サーバあたりのVM数を16としている。

このシステムを使い、1映像フレーム約32MByteの非圧縮4K映像の合成処理実験を行った。図12に単位時間

あたりに処理された映像フレーム数の時間変化を示す。この図の縦軸は単位時間あたりに処理したフレーム数であり、横軸は経過時間である。この結果から、一時的な落ち込みが観測されたものの、1秒あたり44.1フレームの非圧縮4K映像の合成処理ができることが確認された。リアルタイムである秒60フレームの合成処理はできなかったが、一般的なPCを使った場合、1秒あたり数フレームしか合成処理できないことを考えれば十分短時間で処理できたことが確認できた。なお、この実験では1,207フレームから構成される映像の合成を行ったが、その中の785フレーム(65.0%)が会場内のVMによって、422フレーム(35.0%)がStarBED内のVMによって処理されている。

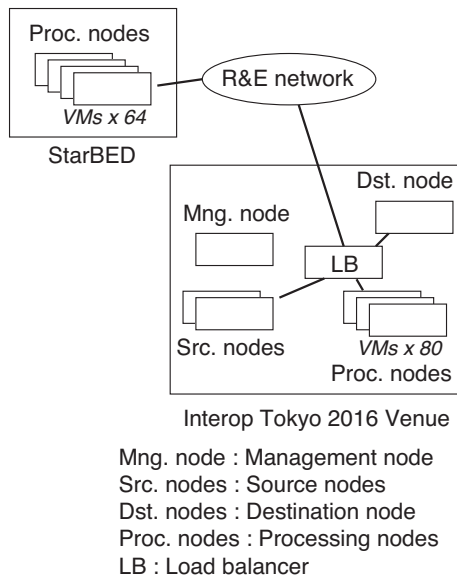


図9 評価実験システム構成

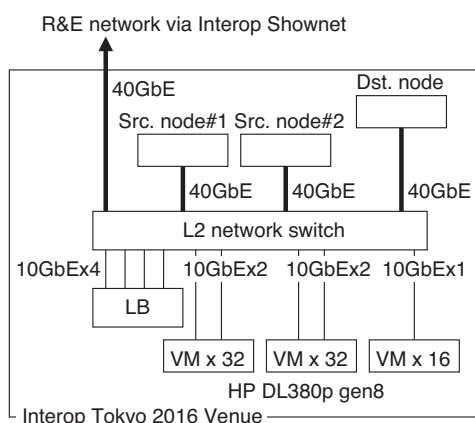


図10 Interop会場内システム詳細構成

4.2 シミュレーションとの比較

上記の実証実験を再現するために、会場内のVMへのRTTを4msecに、StarBEDへのRTTを8msecとして、

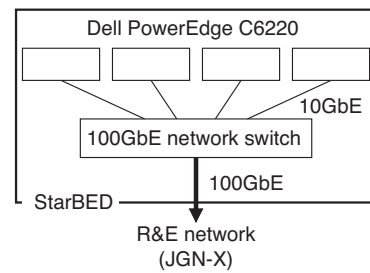


図11 StarBED内システム詳細構成

表2 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

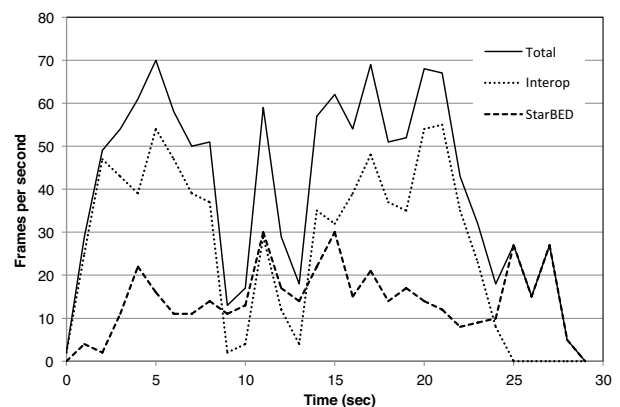


図12 単位時間あたりの処理フレーム数

10Gbit/sのインタフェースの接続された物理サーバが会場に5台、StarBEDに4台とし、それぞれの物理サーバに16VMを構築した構成でシミュレーションを行った。各VMに対するRTTは負荷分散装置であるソフトウェアスイッチを経由するため、一般的なネットワークスイッチやルータを経由する場合よりも大きめの値を使用している。その結果、システム全体の処理性能は46.3フレームとなり実測された値と5%以内で一致した。また、処理された映像フレームの割合は会場側が69.6%、StarBED側が30.5%であった。実証実験において会場側で65.0%、StarBED側で35.0%の映像フレームが処理された結果と

比較すると、会場側で処理されたフレーム数が実験よりもシミュレーションの方が多く見積もっており、その結果、割合も絶対数も多くなったと考えるのが妥当であろう。一方で、実証実験の結果である図 12 を見たときに、理由は不明であるが会場側の VM の処理が5秒ちかく止まっている時間が観測されている。もし、この停止がなければ実証実験において会場側で処理できたフレーム数はさらに増えると思われ、それによってシステム全体で処理されるフレーム数も増えると考えられる。これによって実システムの性能とシミュレーション評価結果とは、さらに近づくと考えられる。実際のシステムの性能は途中のネットワークの混雑や同じハードウェアを使用しているユーザの使用状況にも左右されるものの、事前にシステム性能を把握したい場合や VM を追加、削除したときの性能変動を把握するには、十分な精度であると考えられる。

また、シミュレーション評価に必要な時間については、Intel 社製 Core i5-8500B (3GHz, 6 コア) を搭載したデスクトップ PC で 3.6 秒であり、シミュレートする時間を短くすれば、運用中に実施するリアルタイム性確保のための系構成変更に伴う性能評価にも十分利用可能なものであると考えられる。

5. まとめ

我々は、高精細映像合成処理などの大容量計算を行うことを目的として、複数クラウド上の複数 VM を使って並列分散処理を行うためのフレームワークを提案し、このフレームワークを使った非圧縮 4K 映像合成処理システムを実現した。さらに、このフレームワークで構築したシステムの全体性能を評価するためのシミュレーション法を提案し、この方法を実装したシミュレーションプログラムを用いて、非圧縮 4K 映像合成処理システムの特性を評価し、複数箇所のクラウドを用いた場合でも、VM 数に応じてニアに性能向上できることが確認できた。また、クラウドまでの RTT と実効帯域によりシステム全体性能が大きく左右されることも確認された。

我々は、この非圧縮 4K 映像合成処理システムを 2 拠点に構築した合計 144 の VM を持つ仮想的なクラウド上にこのシステムを展開し実証実験を行い、秒あたり 44.1 フレームの合成処理ができることを実証し、フレームワークの有効性を実証した。シミュレーションの正当性を検証するために、上記のシミュレーションプログラムを用いてこの実証実験システムのシステム性能評価シミュレーションを行った。その結果、5%以内でシステム性能を評価することができることを確認し、シミュレーション評価の妥当性を実証した。今後は、他のアプリケーションにこのフレームワークを適用し、提案フレームワークやシミュレーションの有効性を検証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ました。

参考文献

- [1] 総務省 | 令和元年版 情報通信白書 | PDF 版 (入手先 <http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r01/pdf/01honpen.pdf>) (2019.07.09).
- [2] 監視カメラ世界市場に関する調査を実施 (2018 年) | ニュース・トピックス | 市場調査とマーケティングの矢野経済研究所 (入手先 https://www.yano.co.jp/press-release/show/press_id/1868) (2019.07.09).
- [3] 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).
- [4] 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).
- [5] 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).
- [6] 君山博之, 丸山充, 小島一成, 藤井竜也: 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, マルチメディア, 分散, 協調とモバイル (DICOMO2019) 論文集, pp.1181–1189 (2019).
- [7] 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).
- [8] Apache Hadoop (入手先 <https://hadoop.apache.org/>) (2019.07.13).
- [9] Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, Proc. the 6th conference on Symposium on Operating Systems Design & Implementation (OSDI '04), pp.137–150 (2004).
- [10] Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST) (入手先 https://www.ics.uci.edu/fielding/pubs/dissertation/rest_arch_style.htm) (2016.07.13).
- [11] :// curl (入手先 <https://curl.haxx.se/>) (2019.07.13).
- [12] Redis (入手先 <https://redis.io/>) (2019.07.15).
- [13] Fluentd | Open Source Data Collector | Unified Logging Layer (入手先 <https://www.fluentd.org/>) (2019.07.15).
- [14] Ryu SDN Framework (入手先 <https://osrg.github.io/ryu/>) (2019.07.15).
- [15] Open vSwitch (入手先 <https://www.openvswitch.org/>) (2019.5.2).