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

君山　博之[1,†1,a)]　北村　匡彦[1,b)]　小島　一成[2,c)]　丸山　充[2,d)]　藤井　竜也[1,e)]

概要：近年，サーバやネットワーク機器，IoT 機器のログや監視映像，映像制作における素材映像など大きなデータが日々生成されており，それを処理するための計算機パワーがより必要となってきている．今後，デッドラインまでに，一時的に増えたデータを自前の計算機で処理しきれないケースが増えてくることが想定される．このような場合，自前の計算機を使わずに時間貸しの計算機サービスであるクラウドサービスを使って，増えたデータをクラウドで処理することは可能である．しかし，ベストエフォートのネットワークと VM を使っていることから，想定よりもクラウドまでのネットワークの帯域が狭い，あるいはクラウド内の VM の速度が遅いことによって，自前の計算機で処理した方が早く処理できる可能性がある．そこで，我々は複数のクラウド上の VM に対して処理を動的に分散させることによって，効率的に全体の処理速度を向上させるためのフレームワークを提案する．このフレームワークでは，VM の負荷の状態，ネットワークの状態を常時モニタリングすることによって余力のある VM を動的に発見し，OpenFlow スイッチを使って動的に負荷分散させることを可能にした．加えて，様々な処理を簡単に実装できる API や装置間インタフェース設計および，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った．このフレームワークの動作を確認するため Interop Tokyo 2016 会場にて実証実験デモを行った．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 フレームの非圧縮 4K 映像の合成処理ができることを確認し，提案フレームワークが実際の大容量計算アプリケーションに適用可能であることを実証した．

1. はじめに

近年，日々生成されるデータが大容量化しており，それに伴って大きな計算機パワーを必要とする業務が増加の一途をたどっている．バックボーンネットワークの容量は 10Gbps から 100Gbps を超え，さらに 400Gbps の時代が間近に迫っており，単位時間あたりに生成されるネットワーク装置（スイッチ，ルータ，ファイヤーウォール等）のログも現在よりも 10 倍以上増えると考えられる．IoT 時代になれば，多くの機器がネットワークにつながるだけでなく，これらの機器から生成されるログデータも，当然，増えてくると考えられる．また，映像制作業務においては制作対象がハイビジョンから高解像度の 4K や 8K 映像に移行しつつあり，これらのデータ量は 8 倍あるいは 32 倍以上に増えると思われる．これらの大容量データは保管するためのストレージを必要とするだけで無く，当然，処理する必要があることからそのための計算機パワーが必要となってくる．例えば，ネットワーク機器のログの場合には，そのログを解析し，該当の装置の異常や不正アクセスの兆候がないかをリアルタイムに検査する必要があり，決められた時間内でその解析処理を終わらせる必要がある．映像の場合は，現像，編集，合成，エンコードなど，制作の過程で映像データを処理していくだけでなく放映時間までに作業を完了する必要がある．つまり，決められた制作時間内で仕上げるため必要があり，ログの解析と同様にデータが増えればその分計算機パワーも増やしていく必要がある．

一方，計算機の時間貸しサービスであるクラウドサービスが数多くのプロバイダから提供されており，このサービスで提供可能な仮想計算機（VM）数も年々増加の一途をたどっている．クラウドサービスは VM を時間単位で貸し出すサービスであり，一時的に計算機を使いたいユーザに

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

とっては自前の設備を持つことなく計算機を使うことができる非常に有用なサービスである．そのクラウドサービスで提供されている VM を借り受けようとするユーザは，各 VM に割り当てる CPU 数，メモリ容量，ストレージ容量などを指定し，VM を生成し，それを借り受けることになる．クラウド内の複数の VM を使って大容量計算を実行する場合，計算するためのデータが格納されているローカル計算機から VM までデータを転送し，計算を実行し，結果をローカル計算機に戻す必要がある．そのため，計算にかかる時間は，VM で実行する計算の処理時間だけでなく，データを VM まで転送する時間や VM 間で転送する時間も考慮する必要がある．クラウド上の VM を使って効率良く分散処理させるためには，データを VM まで送信するためのネットワーク帯域，遅延，VM そのものの性能が必要であるが，これらはスペックデータからでは判らないために事前にプロファイリングする必要がある．また，一般的なクラウドサービスでは，複数の VM 間で足回りのネットワークを共用していることからデータの送受信のために利用可能な帯域も時間に対して変動することが考えられ，たとえ事前のプロファイリングを実施したとしても効率的な並列分散処理を保証することは困難である．しかしながら，クラウド上の VM を大容量計算に使えるようになれば，ログの解析や映像制作，あるいはインシデントやアクシデントに伴う一時的な大容量データ解析等，様々な業務において，その作業時間の短縮が期待できる．そこで，我々は，複数のクラウド上の複数の VM を使った効率的な大容量計算を実現するために，その計算時間やデータの転送時間をリアルタイムにモニタリングし，その結果を使って動的に分散処理を行うためのフレームワークを提案する．

分散処理システムの分野では Hadoop/MapReduce 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と分散処理が一体となったシステムで，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システムである．ファイルシステムと一体で性能を発揮するものであることから，常にその環境に蓄積できれば最適な分散処理が可能である．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を全てネットワークを介して，Hadoop 環境に転送してから処理をする必要があるため効率的ではなく，蓄積のための余計な時間が必要になると思われる．

我々が並列分散化する処理は，例えば映像合成処理のように，1 台または数台のローカルサーバ（またはローカルストレージ）に格納されている大量の映像素材に合成・加工処理を施し，その結果を別なストレージに出力する処理，つまり，ローカルサーバに格納されている大量のデータに対してなんらかの処理を行い，その結果を再び蓄積するような一連の処理を対象とする．この絞り込みは，問題を単純化しモデル化をしやすくする効果をもたらす．複雑な問題に対しては，この単純な処理を繰り返し実行することで対応可能である．例えば、パケットキャプチャデータから特定の送信元アドレス，送信先アドレスを持ったパケットを抽出し，その中から特定のデータが含まれているパケットを抽出するなど，この組み合わせ方式によって複雑な処理にも対応ができると考えている．

そこで，我々は，多くのユーザに簡単に使ってもらえるような動的な分散処理環境を提供することを目的に，その分散処理の実現方式を検討し，様々なアプリケーションが実装可能な広域分散処理フレームワークとその実装方法を検討した．我々が提案するフレームワークは，負荷分散装置として標準化されたメッセージによりソフトウエア制御が可能な OpenFlow スイッチを用い，負荷状況のリアルタイムモニタリングシステムとの組み合わせにより，処理させる VM を動的に選択可能な環境を提供し，様々な処理を容易に実装出来るアプリケーションプラットフォームを実現可能にするものである．このフレームワークの有効性を実証するため，アプリケーションとして 1 フレームあたり 800 万画素の非圧縮 4K 映像を合成するソフトウエアを実装し，Interop Tokyo 2016 の会場で公開実験を行った．Interop Tokyo 2016 の会場で立ち上げた 80 台の VM および NICT 北陸 StarBED 技術センター内に立ち上げた 64 台の VM に対して処理を動的に分散することによって秒あたり約 44 枚の速度で合成処理できることを確認し，提案フレームワークが実際の大容量計算アプリケーションに適用可能であることを実証した．

本報告では，次章で動的に負荷分散処理を行うためのモニタリングと連携した分散処理方式について説明し，続く第 3 章では OpenFlow スイッチを使った負荷分散処理の実装方法，ノード間の通信方式，ソフトウエア実装方式について説明する．第 4 章では合成処理アプリケーションを使った実証実験について説明し，その結果と考察について記述する．最後にまとめと今後課題について記述する．

2. 動的負荷分散方式の検討

我々が実現しようとしている分散処理システムの概略構成を図 1 に示す．この図に示すように，我々が実現しようとしている分散処理システムは，以下のように処理を実行する．まず，最初に適当に分割された処理すべきデータ（入力データ：Input data）を持っている計算機（素材蓄積ノード：Source node）から，複数の VM（計算ノード：Processing node）のうちの 1 台を選択し，その入力データを送信する．次に，そのデータを受信した VM はそのデータに対して演算処理を行い，結果を別な計算機（結果蓄積

ノード：Destination node）に送信．これらの処理を並列的に実行して，全ての入力データに対して処理が完了したら，全体の処理を完了する．前述したように，複雑な計算を行わせたい場合は，このシステムを多段に組み合わせることによって実現可能であることから，我々は，まず，この基本的なシステムを実現することとした．このシステムを使って効率よく分散処理を実行させるためには，負荷の低い VM に対して入力データをより多く送ればよい．そのためには，VM の負荷状況をモニタリングし，データ入力から処理，結果の出力が完了するまでの一連の処理時間（ターンアラウンドタイム）が短そうな VM を，その都度選択することによって，ターンアラウンドタイムの短い VM に対して入力データがより多く送信されるようにすればよい．そこで，我々は，VM からのモニタリングデータの収集方法の検討を行い，モニタリングデータにもとづいた動的な負荷分散実装方式について検討を行った．

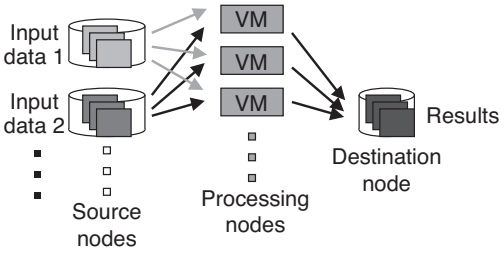

図 1　分散処理システム概略構成図

2.1　モニタリングデータ収集方式の検討

処理を行う VM を決定するため，我々は，VM を管理するための装置（管理ノード：Management node）を別途，用意することとした．管理ノードは，動的な負荷分散を実現するために各 VM の処理時間等のモニタリングデータ（負荷情報）を集めるとともに，どの VM が使われていて，どの VM が空いているのかを管理し，どの VM を使うべきかを決定するための装置である．そのために各 VM において計測された処理時間等の負荷情報を管理ノードに収集して保存する必要がある．管理ノードでデータを収集する方式として，方式 1:管理ノードから各 VM に対して問い合わせる方式，方式 2:各 VM から定期的に管理ノードに計測データを送信する方式の 2 方式が考えられる．方式 1 は，管理ノード主体で VM に問い合わせをかけ，その結果を VM から取得することになるため，負荷情報を一旦 VM に蓄積しておき，管理ノードからの非同期の問い合わせに対して，その応答として返す必要がある．一方，方式 2 は VM 主体で管理ノードに負荷情報を送るため，VM 側で負荷情報が生成された時点，つまり，処理が完了した時点で管理ノードにその情報を送信すればよいことから，一旦，VM でその情報を蓄積する必要がないという利点がある．この方式は，管理ノードに非同期の負荷をかけることになるが，実装が簡単になることが想定されことから本システムでは方式 2 を採用し，管理ノードに各 VM の処理時間情報を蓄積することとした．

2.2　負荷分散方式の検討

次に，前述した VM 毎の負荷情報を使って複数の VM に処理を分散する方式を検討した．処理を分散させる方式として，図 2 に示すように，管理ノードが，素材蓄積ノードに対して送信先を適宜指定する方式．および，図 3 のように，素材蓄積ノードと計算ノードとの間に負荷分散装置（Load balancer）を挿入し，管理ノードは負荷分散装置に対して負荷分散の制御を行い，素材蓄積ノードは負荷分散装置に一旦データを送信し，負荷分散装置がその時点でターンアラウンドの短い VM を選択して転送する方式（方式 2）の 2 方式が考えられる．方式 1 は，新たな装置を追加する必要がないため，少ない装置数でシステムを構成でき，コストも抑えられる反面，負荷分散機能を持った専用のファイル転送アプリケーションを素材蓄積ノード用に準備する必要がある．一方，方式 2 は，新たな装置を導入する必要があるが，素材蓄積ノードを市中ファイル転送ツールや Open Source Software（OSS）により実装できる可能性がある．そこで，我々は，素材蓄積ノードとして，業務に利用している一般のファイルサーバがそのまま使えるように方式 2 を選択し，その実現のための負荷分散装置の実装方式の検討を行った．

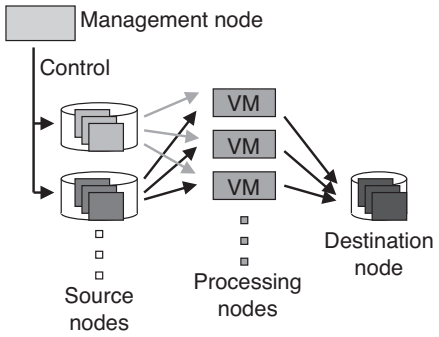

図 2　負荷分散方式 1

3.　実装方法の検討

上記の検討結果を元に，システムを実現するために必要な負荷分散方式の実装方法およびノード間の通信方式，ソフトウエアの実装方法の検討を行った．

実装にあたっては，以下の条件を考慮した．

条件 1：拡張性の確保　負荷分散のベースシステムをミドルウエアとして独立させ，アプリケーションを簡単に入

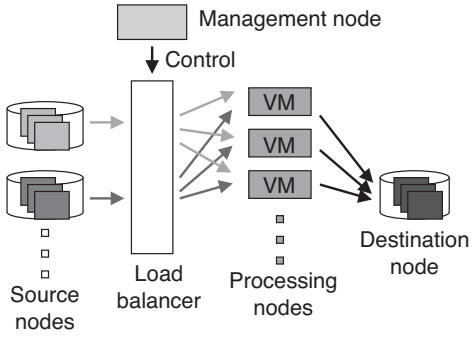

図 3 負荷分散方式 2

れ替え可能にするとともに，各ノードのソフトウエアを共用可能な設計にする．

条件 2：汎用プロトコルの採用とステートレスの実現 OSS や市中ツールを使えるようにノード間の通信インタフェースはできる限り汎用的なものを使用する．また，分散処理システムは多数のノード間で通信する必要があることから，ノード間で状態遷移を行うとエラー処理が複雑になるため，状態遷移を伴わないプロトコルを採用する．

条件 3：イントラネットへの対応 多くのユーザは，割り当てられているグローバル IP アドレス数の制限やセキュリティの確保のため NAT ルータや Firewall などの Gateway 装置を設置して，処理すべき入力データをイントラネットの内側に置いているケースがほとんどである．そのことから素材蓄積ノードを NAT ルータの内側（イントラネット側）に置いても利用できるようにする．

以下に，我々が提案する実装方式について説明する．

3.1 負荷分散方式の設計

本システムでは，計算をさせるための VM を動的に決定する必要があるため，例えば [1] のような独立型の Web サービス用の負荷分散装置を適用するのは難しい．そこで，文献 [2] で提案した OpenFlow スイッチによる負荷分散装置を採用することとした．この文献に示すように，OpenFlow スイッチを使って OpenFlow スイッチ内で IP パケット化された Input data の送信先情報を送りたい計算ノードの送信先情報に動的に書き換えることによって，所望の宛先へのデータ転送を実現する．この方式を使うメリットは，標準化された OpenFlow メッセージを使い OpenFlow スイッチ内の Flow テーブルを書き換えることによって負荷分散機能が実現できるため，汎用性が高くかつ管理ノードの実装が比較的容易にできることにある．さらに，素材蓄積ノードは計算ノードを意識すること無く，単に OpenFlow スイッチに対してのみにデータを送信するだけなので，素材蓄積ノードの実装も，後述するように専用のアプリケーションを作ることなく実現できるメリットがある．以上の理由から，OpenFlow スイッチを負荷分散装置として利用し，管理ノードには OpenFlow メッセージを使って OpenFlow スイッチを制御するための OpenFlow コントローラを実装するとともに，OpenFlow コントローラを制御するための専用の制御ソフトウエアを実装することとした．

管理ノードには，OpenFlow スイッチの Flow テーブルを書き換える処理のほか，前述したように各 VM からの負荷情報やアプリケーションの実行状況を収集する仕組みを実装する必要がある．前章で説明したように，各 VM からの情報を集める方法として，リアルタイムにデータを集めることに重点をおいた VM 主導の収集方式を採用した．そのため管理ノードは各 VM から非同期に上がってくる情報を受け取り，排他制御しながら格納する必要がある．そこで我々は，前述した OpenFlow スイッチ制御用のソフトウエアとともに DBMS を管理ノード内に置き，VM からの情報を DBMS に格納することとした．

以上のソフトウエアを使って素材蓄積ノードから計算ノードに対してデータを送る場合は，文献 [2] で提案した通り，以下のように処理を行う．OpenFlow スイッチは，素材蓄積ノードからの TCP SYN パケットを受信すると，OpenFlow コントローラを実装した管理ノードに対して Packet In の問い合わせを行う．次に，管理ノードは，DBMS から VM の負荷情報を読み出し，その情報を元に計算ノードを選択し，OpenFlow スイッチに対して，その計算ノードに入力データが流れるように Flow テーブルの書き換えを指示する．OpenFlow スイッチは，TCP SYN パケットの送信先アドレスを書き換え，選択された計算ノードにそのパケットを転送し，素材蓄積ノードと選択された計算ノード間のコネクションを確立させ，次節に示す REST（Representational State Transfer）様式 [3] のメッセージフォーマットを使った入力データの計算ノードへの転送が実行される．

3.2 ノード間通信方式の設計

次に，ノード間の通信方式の実装方法について説明する．素材蓄積ノード，結果蓄積ノード，計算ノード間における通信は，ノード間でのデータの属性情報の受け渡しとデータそのものの受け渡しをするための通信である．そのため双方向通信が必要である．条件 3 に記載したとおり，NAT ルータへの対応が必要であり，かつ双方向通信をサポートする必要があることから，提案システムでは NAT ルータへの IP マスカレードの設定が必要な UDP/IP ベースの高速なファイル転送プロトコルでは無く TCP/IP ベースのプロトコルを採用することとした．さらに，その上位プロトコルとして HTTP を使った REST 様式のプロトコルを

採用することとした．REST メッセージにはデータとその属性情報とを重畳でき，条件 2 のステートレスを満たすこと，加えて RPC や SOAP に比べプロトコルが単純であることから REST を採用した．また，CURL などの OSS のツールを使うことによってプログラムを書くことなくデータを容易に送ることができることも REST を採用した理由である．

3.3 ソフトウエア設計

我々は，前述の負荷分散の仕組みを検証するため，素材蓄積ノード，計算ノード，結果蓄積ノード，管理ノード用のソフトウエアを設計し，その試作を行った．図 4 に，試作した本システムのソフトウエア構成を示す．負荷分散装置である OpenFlow スイッチには，ソフトウエアベースの Open vSwitch[4] または Lagopus[5] を利用できるようにした．素材蓄積ノード，計算ノード，結果蓄積ノード間の通信は REST プロトコルにて行われることから，ノード間通信機能を共通の通信モジュール（Communication module）として実装した．また，上記の 3 種類のノードから管理ノードへのログや負荷情報の転送には，事前の検討により OSS である Fluentd を用いた [6]．管理ノードの負荷情報 DBMS として，オンメモリで動作する Redis[7] を採用した．OpenFlow スイッチへのコマンド発行のための OpenFlow コントローラとして Ryu[8] を利用することとし，それ以外の機能である負荷分散制御の部分を新規開発した．一般的に利用されている SQL ベースの RDBMS ではなく Redis を採用した理由は，オンメモリで高速に動作するからである．

このシステムでは，管理サーバの処理が遅れた場合，OpenFlow スイッチから計算ノードへの SYN パケットの転送が遅れ，それにより素材蓄積ノードへの戻りの SYN ACK が遅れることから，素材蓄積ノードで SYN ACK 待ちタイムアウトになり，素材蓄積ノードから SYN が数秒後に再送信される．それにより素材蓄積ノードからの送信処理の数秒間開始されない問題が発生する．このような再送信を避けるため管理ノードでの処理時間を極力短くする必要があることから，我々はレスポンス時間の短い Redis を採用した．Redis を採用することによって，万が一の障害時に収集した負荷情報が失われることが想定されるが，直近の情報により計算ノードの選択が可能であることから大きな問題にはならないと考えている．VM の選択のアルゴリズムについては，図中の管理モジュール（Management module）に実装し，入れ替え可能な設計とした．

前述した通り，素材蓄積ノード，計算ノード，結果蓄積ノード用に通信機能を通信モジュールとして共通化したが，それぞれの独自機能は処理モジュール（Processing module）として実装した．この処理モジュールは簡単に入れ替えられるように，Shared object として実装可能とし，複数の通信モジュールから共有メモリ渡しでデータを受け取れるように API を定義するとともに，同様にメモリ渡しで処理済みデータを通信モジュールに渡せるように API を定義した．素材蓄積ノードや結果蓄積ノードには，ストレージからデータを読み出し通信部に渡す処理，および，通信部からデータを受け取りストレージに書き出す処理を Shared object としてそれぞれ実装した．次章で説明する評価実験のために，計算ノードには 2 枚の TIFF 画像を合成するソフトウエアを Shared object として実装した．

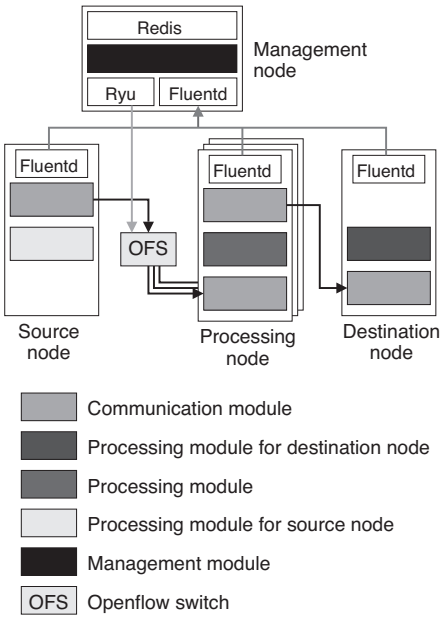

図 4 ソフトウエア構成

4. 評価実験

4.1 実験システム

我々は，前述した提案フレームワークの動作実証を行うため，図 5 に示す実験システムを構築し，Interop Tokyo 2016 の期間中，神奈川工科大学のブースにて公開実験を行った．この実験では，ブース内の 2 つの素材蓄積ノードに格納されている 2 種類の TIFF 連番フォーマットの非圧縮 4K 画像（3840x2160 画素，RGB 各色 8 ビット）を，ブース内および北陸 StarBED 技術センター内の複数の VM を計算サーバとして使い，処理を分散させることによって合成処理を行った．神奈川工科大学ブースと北陸 StarBED 技術センターとの間は，研究教育ネットワークにより接続した．Interop Tokyo 2016 会場との間の往復遅延時間は 14msec であった．StarBED からは Interop Tokyo 2016 会場までの間は，JGN-X，SINET5，JPIX を経由して，会場のネットワークである ShowNet まで，物理回線速度 100Gbps で接続され，ブースの手前のスイッチで 40Gbps に速度を落とした．JGN-X とブース内の各ノードを L2 で

接続するため VPN 装置を JGN-X の大手町 NOC 内および会場内に設置した．この VPN 装置により最大の帯域は 10Gbps に制限されており，また，SINET5，JPIX は他のトラヒックと帯域を共有していた．ブース内の計算ノードには HP 社 DL380p gen8 を 3 台，StarBED 内の計算ノードには Dell 社 PowerEdge C6220 を 4 台割り当てた．計算ノードのハードウエアスペックを表 1 に示す．ブース内に設置した負荷分散装置（OpenFlow スイッチ）および管理ノードにも同じ HP 社 DL380p gen8 を使用した．図 6 に実験の様子を示す．図中右側のラック内には，計算ノード，負荷分散装置，管理ノード，素材蓄積ノードを設置し，左の机の下に結果蓄積ノードを，机上には 4K モニタを設置し，合成結果をその場で確認できるようにした．

表 1 計算ノードスペック

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)
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)

ブース内の素材蓄積ノードおよび結果蓄積ノードには，10Gbit/s 以上の転送性能が必要であったことから Mellanox 社の 40 Gigabit ethernet カードを搭載し，負荷分散装置には 4 本分の表 1 に示した Intel 社の複数の 10 Gigabit ethernet カードを搭載した．ブース内の計算ノード用の 2 台の HP 社 DL380p には 32 台の VM を，残り 1 台には 16 台の VM を立ち上げ，ブース内には合計 80 台の計算ノード用の VM を立ち上げた．StarBED の計算ノード用の Dell 社 PowerEdge C6220 には 1 台あたり 16 台の VM を計算ノード用に立ち上げた．StarBED および Interop 会場の詳細構成を図 7 および図 8 に示す．計算ノードの VM 数は接続する 10GbE network interface 毎に 16 台とした．これは，事前評価によって，10GbE network interface に割り付ける VM が 16 台を超えると，帯域の限界により VM 1 台あたりのデータ転送速度が低下することが確認されたためである．

HyperVisor として Ubuntu Linux および KVM を使用し，これらの VM には仮想 CPU2 台と 2GByte の仮想メモリを割り当て，Ubuntu Linux と前章で説明した計算ノード用のソフトウエアをインストールした．負荷分散装置には，Ubuntu linux および Open vSwitch をインストールした．

合成処理した TIFF 画像のサイズは，背景が 1 枚あたり 25MByte，前景は，RGB 以外にアルファチャンネルのデータが含まれていることから 1 枚あたり背景よりも大きい 33MByte である．このシステムを使って，2 台の素材蓄積ノードから 2 枚の TIFF 画像を負荷分散装置を経由して 144 台の計算ノードに送信し，合成処理を行い，その合成結果を結果蓄積ノードに送信する実験を行い，その処理時間を計測した．この実験における計算ノードの選択アルゴリズムは次の通りである．1 巡目は，全ての計算ノードに対してアドレス順に並列的に REST メッセージとデータを送り合成処理を実行させた．2 巡目以降は，前の周回のデータ転送時間も含めた処理時間の短い順に計算ノードを選択する方式を採用した．事前のプロファイリング無しで負荷分散を行えるようにすることがこのシステムの目標であり，前の周回の処理結果を次の処理分散に利用するためである．

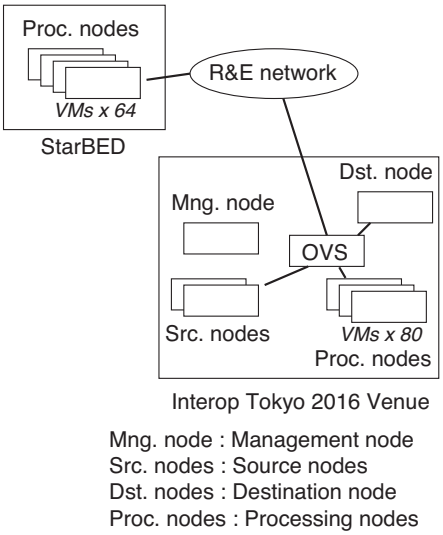

図 5 評価実験システム構成

図 6 実験模様

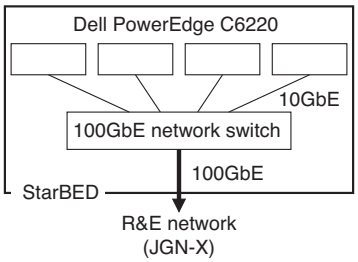

図 7 StarBED 内システム詳細構成

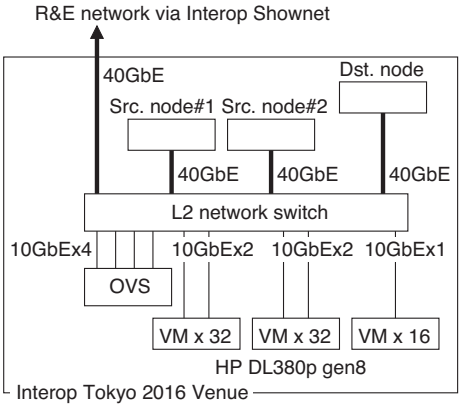

図 8 Interop 会場内システム詳細構成

4.2 実験結果

このシステムを使って，1207 映像フレームの TIFF 画像の合成処理を行った．そのときのシステム全体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 9 に示す．この図の縦軸は処理された映像フレームの累積数であり，横軸が経過時間である．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，1207 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 40 分で完了できる性能であることを意味しており，映像制作業務において十分使える速度であると思われる．

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 10 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 で処理された映像フレーム数，破線はブース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，1207 フレームのうち 785 フレームがブース内の VM で，422 フレームを StarBED の VM で処理されている．これは割合にすると 65%をブース内の VM で，35%を StarBED の VM で処理したことになる．VM 数はブース内が 80，StarBED 側が 64 なので，割合にすると 56%がブース内，StarBED が 44%である．これらの結果から，遠い StarBED よりも近い神奈川工科大学ブース内の VM の方が，より多く使われたことが確認され，本フレームワークが実現可能であることが確認された．図 10 中に，2 回単位時間あたりの処理数が落ち込む時間があることが観測された．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．今後，本実験データの詳細な分析を行い，上記の分散割合が妥当かどうかの検証を行うとともに，さらに効率の高い負荷分散アルゴリズムを検討し，負荷分散割合の最適化を行うことによって全体処理能力の向上を図っていく予定である．

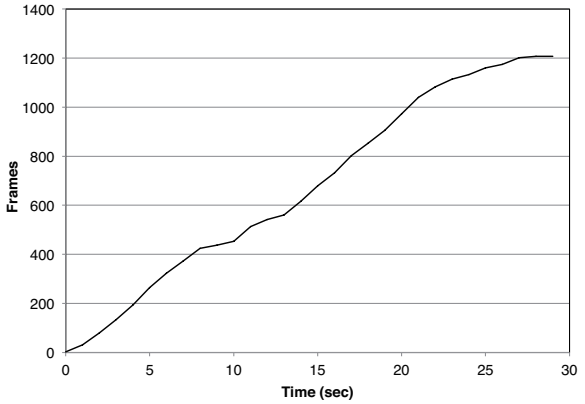

図 9 累積処理フレーム数の時間変化

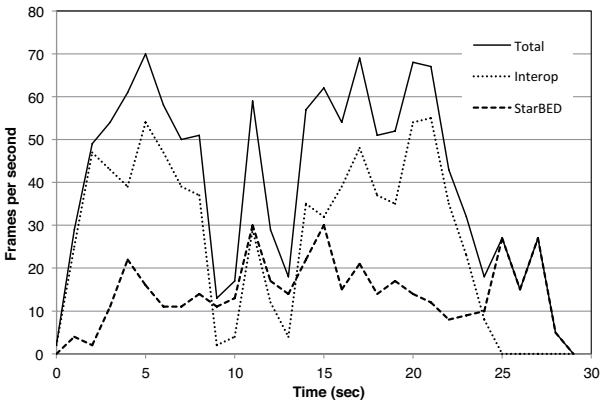

図 10 単位時間あたりの処理フレーム数

5. まとめ

様々なネットワークでつながり様々な性能を持つ計算機環境を使って効率よく並列分散処理を行うことを目標に，並列分散処理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，素材蓄積ノード，負荷分散装置，計算ノード，結果蓄積ノード，管理ノードから構成される分散処理システム構成を提案した．さらに，モニタリングデータを使った動的な負荷分散方式を提案するとともに，OpenFlow スイッチを使った動的な負荷分散システムの実装方式，計算機の負荷状況のモニタリングシステムの実装方式，および，応用可能なソフトウエアの実装方式について提案した．このフレームワークを使って，1 フレームあたり 33MByte の非圧縮 4K 画像の分散合成処理を行うソフトウエアを実装し，Interop Tokyo 2016 の会場にて，実証実験を実施した．会場に設置した PC サーバおよび北陸 StarBED 技術センター内の PC サーバ上に作成した仮想マシン（VM）144 台を使って動的な分散処理を行い，単位時間あたり 44 映像フレームの合成が可能であることが確認できた．この実験では 65%を会場の VM で，35%を StarBED の VM で処理され，近いところに配置した VM の方がより多く選択されることが確認され，提案したフレームワークが実現可能であることが実証された．

今後は，本実験で採用した負荷分散アルゴリズムを見直し，より効率の良い負荷分散を本フレームワークに実装することによってさらなる性能向上を目指すとともに，本フレームワークを使った新たなアプリケーションの開発，実業務でのトライアルを行い，本フレームワークのブラッシュアップを図っていきたいと考えている．

謝辞　本実証実験をサポートして頂いた国立研究開発法人 情報通信研究機構 小林和真教授，北陸 StarBED 技術センター長 宮地利幸様ならびに同センターの皆様，JGN NOC メンバーの皆様，SINET5 NOC メンバーの皆様，Interop Tokyo ShowNet NOC メンバーの皆様，NTT アイティ株式会社の河野隆様，林丈樹様，ピュアロジック株式会社 上野 賢司様，神奈川工科大学 丸山研の皆様に感謝します．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）の一環として実施しました．

参考文献

[1] アプリケーション配信 (Thunder ADC)| 製品一覧 |A10 ネットワークス 次世代スイッチ |DDoS 防御 |SSL 可視化 (入手先 ⟨http://www.a10networks.co.jp/products/thunderseries/thunder-adc.html⟩) (2016.06.17).
[2] 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）.
[3] Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST) (入手先 ⟨http://www.ics.uci.edu/fielding/pubs/dissertation/rest_arch_style.htm⟩) (2016.06.17).
[4] Open vSwitch (入手先 ⟨http://openvswitch.org/⟩ (2016.6.23).
[5] Lagopus switch (入手先 ⟨https://lagopus.github.io/⟩ (2016.6.23).
[6] 伊東亮，岩崎裕也，北村匡彦，君山博之，丸山充 :クラウド上のリアルタイム映像処理を支援する非均質計算機環境のリソース監視手法，電子情報通信学会 2016 年総合大会 情報・システムソサイエティ特別企画 学生ポスターセッション予稿集，Vol.ISS-SP-162，(2016)．
[7] Redis (入手先 ⟨http://redis.io/⟩) (2016.6.23).
[8] Ryu SDN Framework (入手先 ⟨https://osrg.github.io/ryu/⟩) (2016.6.23).