平成２６年度修士論文

クラウド環境における計算資源の動的共有手法

大学院情報システム学研究科
情報ネットワークシステム学専攻

学 籍 番 号 :　1352021
氏 名:　中島 拓真
主任指導教員: 吉永 努 教授
指 導 教 員 :　長岡 浩司 教授
指 導 教 員 :　森田 啓義 教授
提出年月日 :　平成２６年８月２９日

(表紙裏)

目 次

図 目 次

表 目 次

第1章　序論

2005 年頃から物理マシン上で複数の仮想マシン（VM）を動作させる仮想化が普及しはじめた．VM の実態は計算機を構成するハードウェアをソフトウェアで定義したプロセスであり，Windows や Linux などの汎用 OS をインストールして利用できる．既存の物理マシンを VM に置き換えることで，1 台の物理サーバに複数の VM を稼働させ，サーバ台数の縮小による消費電力およびサーバ設置面積の削減が可能である．しかし，仮想化の普及に伴い，VM および VM ホストの数が増大したため，管理・運用コストが増大した．これを軽減するため，2006 年にアメリカ国立標準技術研究所（NIST）が多数のノードを集中管理できるクラウドコンピューティングを提唱した [1]．同時期から米 Amazon.com 社が Amazon EC2 や Amazon S3 といったクラウドサービスの提供を開始し，2009 年頃から一般に広く普及を始めた．

クラウドは，ネットワークで接続された VM ホスト，それらの VM ホスト上で動作する VM，および VM ホストと VM を管理するクラウド管理サーバで構成される．プロセッサやメモリなどの計算資源はクラウド管理サーバにより管理され，VM に適切な計算資源を割り振ることで，計算資源利用の効率化が行われる．また，VM のライブマイグレーション機能を用いて実行中の VM を別の VM ホストへ移動させることで，VM の稼働率および VM ホストの保守性の確保と，クラウド全体の消費電力削減を図っている [2]．クラウドは Infrastructure-as-a-Service（IaaS），Platform-as-a-Service（PaaS），Software-as-a-Service（SaaS）などの形態があり [1]，それぞれ Amazon Web Services[3]，Heroku[4]，Dropbox[5] といった多様なサービスが提供されている．

クラウドを対象に取り組まれている研究課題として，VM の負荷状況に合わせた計算資源の動的な割り当て技術を適用した Resource-as-a-Service (RaaS) クラウドの実現がある [6]．RaaS クラウドでは，クラウドが VM の負荷状況を監視し，状況に応じて計算資源の割当量を動的に変更する．VM は割り当てられた計算資源を活用することで，計算資源の利用率が高く，柔軟な計算資源利用が可能なクラウドを実現できる．しかし，RaaS クラウドの実現のためにはクラウドと VM それぞれでソフトウェア・ハードウェアの対応が必要であり，現在利用されている技術の組み合わせでは容易に解決できない．このため，RaaS クラウドを実現したクラウドは現在提供されておらず，その実現が望まれている．

RaaS クラウドは，負荷状況が大きく変動する VM に対する計算資源の割り当て量を最適化し，1 台の物理サーバにより多くの VM を格納することを可能にする．一般に，大量のアクセスが予想される VM や，画像や動画を処理するような高負荷な状況が予想される VM を作成する際は，あらかじめ多くの計算資源を割り当てるのが一般的である．しかしながら，クラウド上で動作する VM の負荷状況は変動するため，一度設定した計算資源量が常に適切であることはない．また，常に多くの計算資源を確保することは計算資源の利用率低下につながるため，低負荷な状況では利用していない計算資源を融通する技術が開発されている．

VM の負荷状況に合わせて計算資源を割り当てる技術として，CPU オーバーコミット [7]，メモリバルーニング [8]，透過的ページ共有 [9] などがある．これらの技術の利用により，VM で使用されていない計算資源を回収し，別の VM に割り当てることで，VM ホストの VM 収容数を拡大

可能である．しかし，これらの既存技術により VM の高密度な集約を行った場合，高負荷な状況において VM の計算性能の低下を招く可能性がある．また，VM から外部の計算資源を利用することで，VM の性能を向上させる技術も研究されているが，計算資源の共有が困難である，利用のためのオーバヘッドが大きいなどの問題が未解決であるため，既存技術による RaaS クラウドの実現は困難である．

本論文では，RaaS クラウドを実現するため，上記の技術的課題を回避し，クラウドの計算資源を高効率に利用する手法を提案する．本手法では，クラウドを構成する物理マシンを以下の 2 種類に分離し，VM に割り当てる計算資源を縮小することで，VM ホストの VM 収容数を拡大する．

1. VM を格納する VM ホスト
2. 高負荷な計算を実行する計算ホスト

これら 2 種類の物理マシンで構成されるクラウドを用いて，VM が高負荷な処理を実行する際に，VM が稼働しない計算ホストに処理を依頼するシステムを構築した．画像処理を対象とした実験の結果，VM から計算ホストの計算資源が低オーバヘッドで利用できること，計算ホストが備える GPU を複数の VM で共用できること，計算処理の実行中においても VM のライブマイグレーションが可能であること，計算ホストを 1 台から 2 台に増強することにより，VM から依頼するジョブの実効性能を約 2 倍にできることなどを示す．

本論文は以下のように構成される．2 章では，関連研究および関連技術について述べる．3 章では，提案手法の概要について述べ，4 章で実装について述べる．5 章では，提案手法を組み込んだクラウドを構築し，評価を行う．6 章では，評価を受けて，議論を行い，7 章で結論を述べる．

第2章　関連研究

2.1　Resource-as-a-Service Cloud

そこで，VM に割り当てる計算資源量を動的に変更するクラウドとして，Resource-as-a-Service（RaaS） Cloud が提案されている [6]．通常，クラウド上で VM を運用する際は，あらかじめ VM の計算負荷を想定し，計算資源量を割り当てる．しかし，VM の負荷状況に応じて必要な計算資源量が大きく変動するため，細粒度な時間間隔で計算資源の必要量を算出し，動的に割当量を変更することでクラウド上の計算資源を効率的に利用できる．ただし，技術的な理由から RaaS 型クラウドは提案段階にとどまっており，実装には至っていない．

RaaS 型クラウドを実現すると，以下の効果が期待できる．

1. クラウドの利用コストの削減 VM には負荷状況に応じた計算資源が自動的に割り振られる．このため，クラウド利用者は必要な計算資源を必要時間だけ利用でき，利用しない計算資源に対するコストが発生しない．その結果，クラウド利用にかかるコストを削減し，より強力な計算資源を効率的に利用できる．

2. 高効率な計算資源の利用とクラウドの省電力化 VM の負荷が低い状況ではクラウド上の計算資源が解放されている．このため，クラウド提供者は余剰計算資源を別の VM に割り当てるなど，計算資源の有効運用が可能である．その結果，計算資源の利用率が向上し，省電力なクラウドが実現できる．

RaaS 型クラウドの実現のためには，VM に割り当てる計算資源について，以下の 3 つの条件を満たす必要がある．

1. 短い時間間隔での利用
   現在のクラウドでは，1 日もしくは 1 時間単位での課金が一般的であるが，クラウドにおける計算資源の課金単位時間は短縮傾向にある．現在，CloudSigma[10] 社が 5 分単位での課金を提供しており，今後は 1 分単位，1 秒単位と細粒度化することが予想される．

2. 細粒度な割り当て単位
   VM に割り当てる計算資源量は，あらかじめ用意されたスペック一覧から選択する形式が一般的である．しかし，作成する VM の用途に合致するスペックがあらかじめ用意されているとは限らないため，CPU を 50MHz 単位，メモリを 50MB 単位といったように，細かい単位で計算資源を割り当てられることが望ましい．

3. 計算負荷による動的な変更
   通常，VM に割り当てる計算資源量を変更するには，VM の利用者が明示的に VM スペックを変更する必要がある．また，VM 稼働中における変更では計算資源の拡大後に縮小できないなどの課題点が存在する．

本論文では，高負荷な計算を行う際にのみ VM から強力な計算資源を利用することで，上記の条件に適合可能な計算資源の動的共有手法を提案する．

2.2 VM の高密度化に関する技術

2.2.1 CPU オーバーコミット

CPU オーバーコミットは，VM ホストで利用できる CPU コア数を超過して VM を稼働させる技術である [7]．この技術を利用すると，VM ホストでより多くの VM を動作させることができるため，VM の集約密度が向上し，クラウドの省電力化に効果がある．しかし，複数の VM が動作する状況においては性能低下の原因となるため，適切な運用が必要である．たとえば，図 2.2.1 に示すように，VM ホストの CPU コア数が 4，その上で CPU コア数 1 の VM と CPU コア数 4 の VM が稼働させる．すると，CPU コア数 1 の VM が動作中は VM ホストの CPU コアが残り 3 つ利用可能であるが，VM に割り当てた CPU コアすべてが利用可能な状況にないと動作に支障をきたすため，2 つの VM はタイムシェアリングで動作する．このため，VM ホスト上で複数の VM を動作させると，複数コアを割り当てた VM で十分な性能を得ることが難しい．

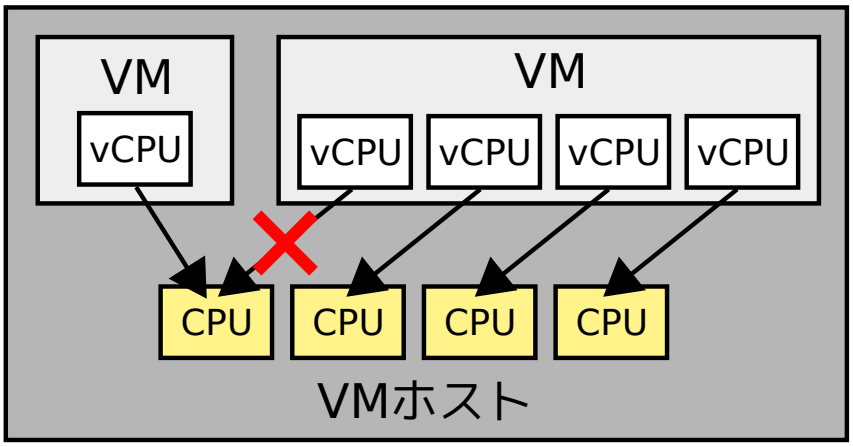

図 2.2.1: CPU オーバーコミット時の衝突

2.2.2 メモリバルーニング

メモリバルーニングを利用すると，1 台の VM ホストに収容する VM に割り当てるメモリ容量の総量が VM ホストに搭載したメモリ容量を超過できる [8]．VM に割り当てたメモリはすべて使用されるわけではないため，VM ホストでは VM 上で使用するメモリのみを VM に割り当て，空きメモリを別プロセスで利用する．その結果，VM ホストに収容できる VM 数が増加し，計算資源の高効率な利用が可能になる．たとえば，32GB のメモリを搭載する VM ホスト上で，2GB のメモリを割り当てた VM を 32 台，合計 64GB 分のメモリを VM に割り当てることが可能である．ただし，各 VM には 2GB 分のメモリ容量が割り当てられるため，すべての VM が 2GB のメモリを消費すると VM ホストの物理メモリ容量を超過するため，図 2.2.2 に示すように VM のメモリ領

域が VM ホスト側でスワップされる．その結果，VM のメモリアクセス速度が大幅に低下し，性能低下の原因となる．

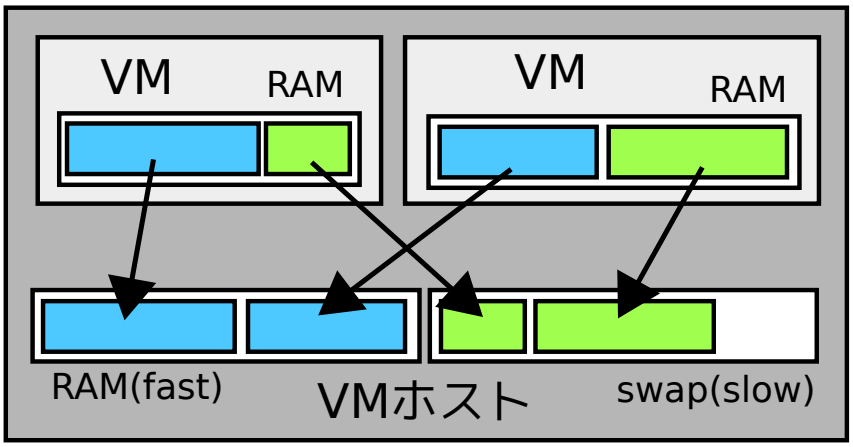

図 2.2.2: メモリバルーニングのスワップ動作

2.2.3 透過的ページ共有

透過的ページ共有は，複数 VM のメモリページの共通部分を重複させることで，利用可能なメモリ領域を拡大する技術である [9]．たとえば，VM ホスト上で同一 OS，同一プロセスが動作する，同一のファイルを扱うなどの条件を満たす複数の VM を扱う場合，それらの VM が利用するメモリページは類似することが予想される．そこで，VM がメモリにデータを書き込む際，対象となるメモリページのハッシュ値を計算し，同一ハッシュ値のデータが既に保存されていればその領域を利用する．その結果，利用可能なメモリを仮想的に拡張できる一方，ハッシュ値の計算によりメモリアクセス速度が低下し，性能低下の原因となる．

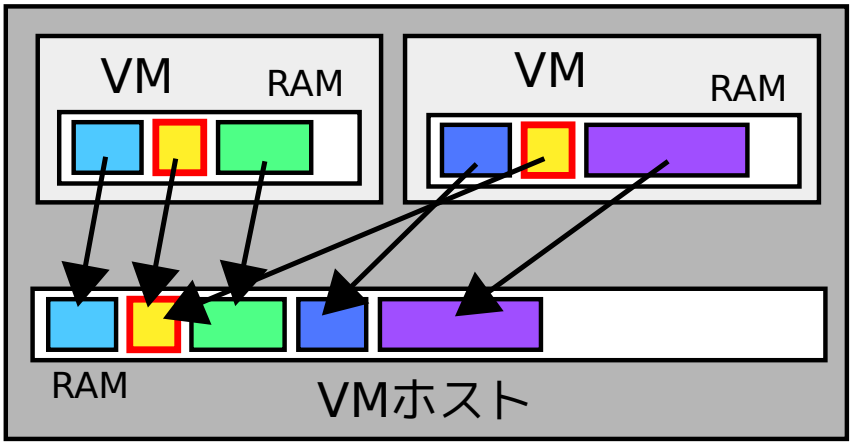

図 2.2.3: 透過的ページ共有の動作

2.3 VM から外部ハードウェアを利用する技術

2.3.1 PCI パススルー

PCI パススルーは，VM ホストに接続された PCI デバイスを VM に直接接続する技術である．たとえば，図 2.3.1 に示すように VM ホストに接続された GPU やキャプチャボードなどの PCI デバイスを VM に接続することで，それらのデバイスを VM から利用できる．また，PCI パススルーでは VM 上での PCI アクセスが対象 PCI デバイスに直接転送されるため，オーバヘッドが小さく，VM ホストで利用した場合とほぼ同等の性能が得られる [11, 12, 13, 14]．ただし，1 台の PCI デバイスは単一の VM にしか接続できないため，計算資源の共有には制約がある．また，PCI デバイスの接続および切断は VM の停止状態でのみ行うことができるため，VM 間での計算資源の共有と VM のライブマイグレーションに制約が発生する．NVIDIA GRID[15] のように，1 枚の GPU を論理的に分割し，複数の仮想 GPU として利用する技術が開発されているが，このような GPU はまだ高価である上，ライブマイグレーションの制約は解決されない．

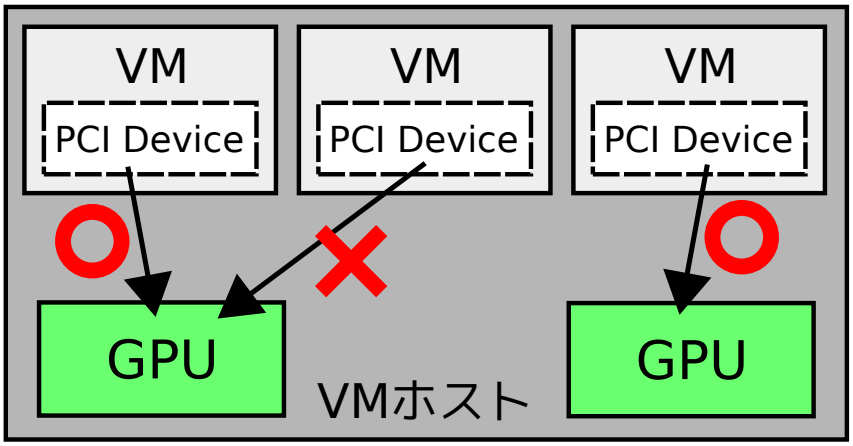

図 2.3.1: PCI パススルーの動作

2.3.2 API インターセプト

API インターセプトは，VM から GPU のようなハードウェアを利用する代替手法である．通常，GPU などの計算デバイスを利用するアプリケーションは，CUDA や OpenCL，OpenGL などのライブラリを利用する．そこで，VM 内に配置されたライブラリを置き換え，計算デバイスを利用する API 呼び出しをインターセプトする．その後，GPU を搭載したホストへリクエストを転送し，計算を行うことで，VM から GPU が利用できる（図 2.3.2）．VM と計算デバイスはネットワークで接続されるため，VM のライブマイグレーションが可能であり，GPU を複数の VM で共用できる [16, 17, 18, 19]．また，1 台の VM から複数の GPU を利用する研究も行われている [20, 21]．しかし，VM と GPU ホスト間の通信がネットワークを経由するためオーバヘッドが大きく，GPU の性能を十分に利用することが困難である [11]．

RaaS クラウドの実現のためには柔軟な計算資源利用を実現する必要があるが，計算資源の性能を十分に利用することができなければ計算資源の高効率な利用は実現できない．上述のように，

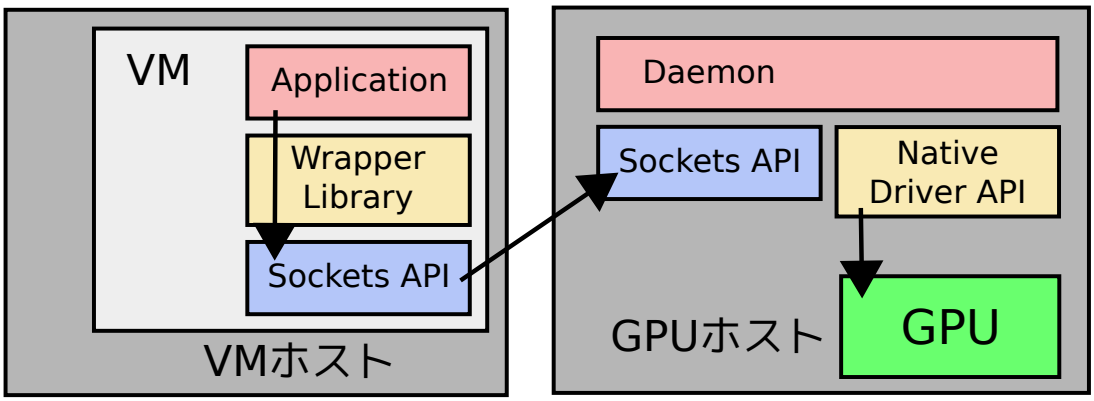

図 2.3.2: API インターセプトによる GPU の利用

PCI パススルーでは VM と計算デバイスの結合が密であるため，計算性能に利点があるが柔軟性に課題がある．一方，API インターセプトでは VM およびアプリケーションと計算デバイスの結合が疎であるため，柔軟性に利点があるが計算性能に課題がある．本論文では，これら 2 つの技術の概念を取り入れることで，

1. VM と計算デバイスが疎に結合され，
2. アプリケーションと計算デバイスが密に結合される

ような手法を提案し，計算性能と柔軟性の両立を図る．

第3章　計算資源の動的共有手法

3.1　設計方針

2 章で述べた RaaS 型クラウドの実現を目的として，(1) 短い時間間隔での利用および (3) 計算負荷による動的な変更に着目し，多数の VM が強力な計算資源を共有する仕組みを提案する．この際，以下の 4 点に注意して計算資源を利用する手法を考える．

(1) 計算資源の共有が可能であること

(2) 高負荷時に他の VM の稼働状況に大きな影響を与えないこと

(3) 計算資源利用にかかるオーバヘッドが計算時間に対して十分に小さいこと

(4) 計算処理中における VM のライブマイグレーションが可能であること

(1) および (4) を実現するため，VM から計算資源をネットワーク経由で利用する．(2) の制約を満たすため，VM から利用する計算資源は VM ホスト外に計算資源を提供するホストを利用する．(3) の実現のため，アプリケーションは強力な計算資源をもつ物理マシン上でネイティブ実行する．本研究ではこれらを考慮し，クラウド上の物理マシンを VM ホストと計算ホストの 2 つに分離し，高負荷な計算処理は VM から計算ホストに依頼する方式をとる．ただし，VM から依頼したジョブは計算ホスト上でネイティブに実行するため，VM 領域にアクセスできない．このため，必要に応じて計算ホストから VM の実行ディレクトリをマウントする機構を実装し，実行ジョブの VM 上の領域へのアクセスを可能にする．

3.2　動作の概要

3.2.1　クラウドの構成要素

先に述べた設計方針の下，図 3.2.1 に示すようなシステムを構成する．このシステムは，高負荷なジョブを計算ホストに依頼する VM，VM から依頼されたジョブを実行する計算ホスト，VM からのアクセスを複数の計算ホストに分散するロードバランサからなる．

また，図 3.2.2 に示すように，VM と計算ホストは互いに疎通可能になるようネットワークで接続する．VM 上では目的のジョブを計算ホストに依頼するためのクライアントエージェントが動作し，計算ホスト上では VM からのジョブを受けて実行するサーバエージェントが動作する．

VM に割り当てる計算資源は OS の動作と低負荷なプロセスに必要な量に限定し，1 台の VM ホストに収容する VM 数を拡大するとともに，ライブマイグレーションに要する時間を削減する．計算ホストは強力な計算資源を備えた物理マシンで構成し，VM からネットワーク疎通可能な位置に設置する．

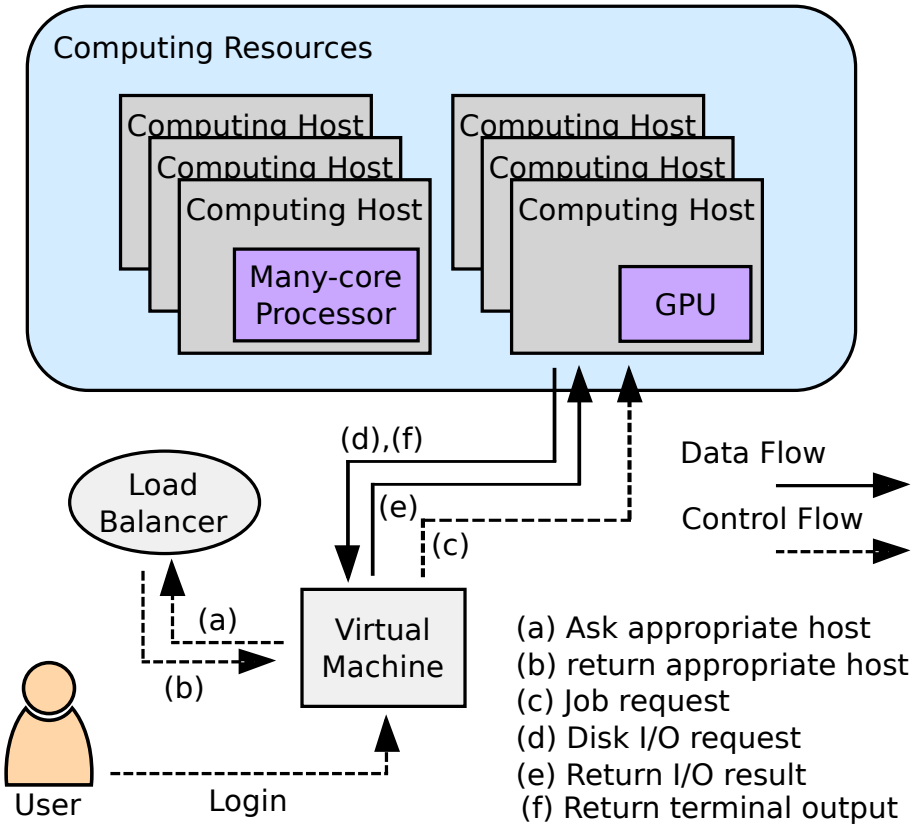

図 3.2.1: 提案するクラウド環境の構成

3.2.2 計算処理の実行方式

VM から計算ホストへのジョブ実行依頼の際は，専用のジョブ依頼コマンド使用し，図 3.2.1 に示す流れに沿って計算ホストに実行を依頼する．

(a) VM はロードバランサに対し，適切な依頼先計算ホストを問い合わせる

(b) ロードバランサが計算ホストの負荷状況をもとに適切な依頼先を返答する

(c) VM から計算ホストに対し，ジョブの実行を依頼する

(d) 計算ホスト上で実行されるジョブが発行するファイル I/O を VM に転送する

(e) VM 上で実行した I/O 結果をジョブに返答する

(f) 計算ホスト上で実行したジョブの出力を VM に転送する

VM から計算ホストの計算資源を利用する際には以下の 2 つの場合がある．

(1) ユーザが VM にログインして高負荷なコマンドを実行する

(2) VM 内にあらかじめ設定されたプロセスが高負荷なコマンドを実行する

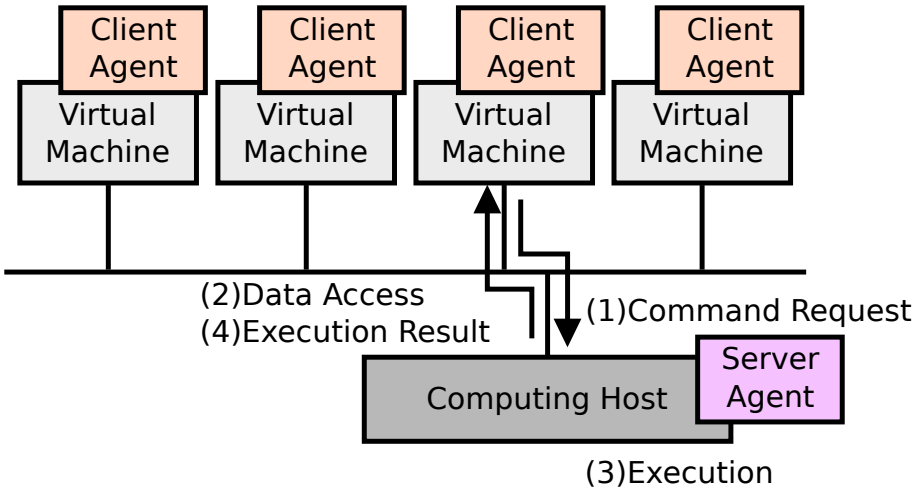

図 3.2.2: 想定するネットワーク構成

(1) の場合，ユーザは対象コマンドに対し，ジョブ依頼コマンド使用することで計算ホストを利用する．(2) の場合，プロセスが実行する対象コマンドに対して以下のいずれか変更を加える必要がある．

(2-1) プロセスのソースコードを書き換え，対象コマンドの直前にジョブ依頼コマンドを追加する

(2-2) 対象コマンドを計算ホストに依頼して実行するスクリプトを作成し，対象コマンドと置換する

3.2.3 ジョブ依頼先計算ホストの選択

クラウド内に複数の計算ホストを配置する場合，各計算ホストの負荷が均一になるようにジョブ依頼先を選択することが望ましい．このため，クラウド上にロードバランサを配置し，VM がジョブを依頼する際にロードバランサにジョブ依頼先計算ホストを問い合わせる．ロードバランサは利用可能な計算ホストの一覧とその負荷状況を保持し，最も低負荷な計算ホストを選択する．その後，VM で発行されたジョブが計算ホストに転送され，計算結果が VM に返される．

3.2.4 計算ホストにおけるジョブ実行

VM からジョブの実行依頼を受けると，ジョブは計算ホスト上の CPU，メモリ，GPU，およびその他の計算資源を直接利用できるため，高負荷なジョブを高速に実行できる．この際，計算ホスト上で実行されるジョブは VM 上のファイルにアクセスする可能性がある．このため，計算ホストがジョブを受け取ると，発行元となる VM に問い合わせ，ジョブが発行されたディレクトリを取得し，そのディレクトリを計算ホスト上にマウントする．マウントしたディレクトリ上でジョブを実行することで，ジョブが発行したファイル I/O は VM に転送される．このようにして，計算ホストで実行されるジョブは VM の実行ディレクトリ上にあるファイルにアクセスする．また，ジョブ実行結果のうち，標準出力および標準エラー出力は，VM の標準出力にリダイレクトされ

る．このため，ジョブの実行を依頼したユーザまたはプロセスからは，ジョブが VM 上で実行されたように認識される．

計算ホストに依頼されたジョブは，計算ホストでネイティブに実行されるため，ジョブの実行に必要なコマンドやライブラリはあらかじめ計算ホストにインストールしておく必要がある．実行できるコマンドが限定される一方，計算ホストの構成にチューニングされたプログラムを利用できる，VM にインストールされていないプログラムでも計算ホストに存在していれば実行可能である，という利点がある．

第4章　実装

4.1　計算ホストを利用した処理の実行手順

VM から計算ホストにジョブの実行を依頼する際は，後述するジョブ依頼コマンドを使用する．VM と計算ホスト間のやり取りは，クライアントサーバ型の通信を介して実装した．VM ではクライアントエージェント（CA）を，計算ホストではサーバエージェント（SA）を動作させる．CA はジョブの依頼とデータアクセスの管理を，SA はジョブの受け付けと実行を行う．また，適切な計算ホストを選択するため，クラウド内にロードバランサを配置する．それぞれのエージェントは図 4.1.1 に示すようなソフトウェア構成で実装した．VM から計算ホストにジョブを依頼する際のシーケンス図を図 4.1.2 に示す．ジョブの実行は，CA と SA を介して以下の手順で行われる．

(1) ユーザはジョブ依頼コマンドを入力する

(2) CA はジョブ依頼先となる計算ホストをロードバランサに問い合わせる

(3) CA はジョブを実行するディレクトリが SA からアクセスできるようにアクセス権の設定を行う

(4) CA は依頼するジョブをロードバランサから返答された計算ホストの SA に転送する

(5) SA は CA から受け取ったジョブをジョブキューに追加する

(6) SA は新規ディレクトリを作成し，CA の実行ディレクトリをマウントする

(7) SA はジョブキューからジョブを取り出し，(3.2) でマウントしたディレクトリ上で実行する．この際，実行ジョブが発行する I/O は (3.2) でマウントしたディレクトリを介し，VM に転送される

(8) SA で実行したジョブの標準出力と標準エラー出力を CA に転送する

(9) CA は SA から受け取ったジョブの出力を表示する

(10) ジョブ実行後，SA は CA の実行ディレクトリをアンマウントし，マウントしていたディレクトリを削除する

上記手順により，VM から処理対象となるデータをあらかじめ転送することなく計算ホスト上で利用できるため，処理対象データの転送に要するコストが縮小される．また，計算ホスト上でアプリケーションを実行するため，計算ホスト上にインストールしたアプリケーションに GPU が必須であるなど実行環境に制約がある場合も問題なく処理が可能である．さらに，ロードバランサの利用により，複数計算ホストで構成されるクラウドに対応できるため，クラウドが高負荷になった場合に計算ホストを追加することで対応できる．

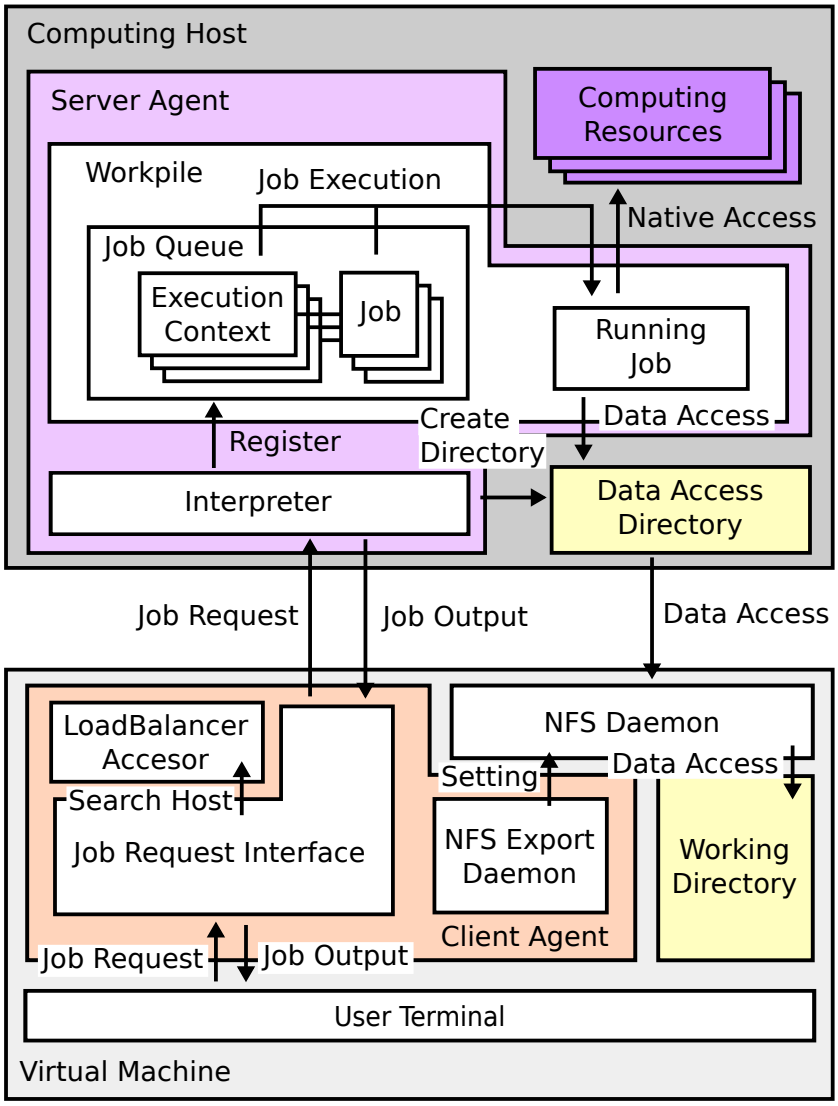

図 4.1.1: エージェントのソフトウェア構成

4.2 クライアントエージェントの実装

CA はユーザからジョブの実行依頼を受け，計算ホスト上で動作する SA にジョブ実行依頼を転送する．また，計算ホストから VM 領域の I/O 要求を受けて応答する機能が必要である．このため，CA はユーザからジョブの実行を受け付けて SA と通信を行うモジュールと，I/O 要求に応答するモジュールの 2 つで構成する．ジョブを計算ホストに依頼するためのモジュールはジョブ依頼用コマンドとして，計算ホストで実行するジョブにファイル I/O を提供するモジュールは NFS を自動設定するデーモンとして実装した．ジョブ依頼コマンドは VM から計算ホストにジョブを依頼する際に使用し，表 4.1 に示す書式で使用する．実行するジョブの実行コマンドと引数をそれぞれ指定することで，自動的に計算ホストに実行が依頼される．たとえば，input.jpg という画像ファイルを縦横それぞれ 5%に縮小する処理は convert コマンドを用いて

```
convert -resize 5% input.jpg output.jpg
```

のようになるが，これを計算ホストに依頼するには

```
offld -- convert -resize 5% input.jpg output.jpg
```

のように変更すればよい．ここで--は，それ以降をすべて引数として解釈することを示すコマンドライン引数である．この引数以降で--help などのオプションを指定しても，その文字列はオプションではなく引数として解釈される．

NFS 自動設定デーモンは，ユーザによりジョブの実行が依頼されると，計算ホストからジョブの実行依頼が発行されたディレクトリをマウントできるよう設定を行う．具体的には，インタプリタから実行ディレクトリのパスを受け取り，/etc/exports に追記した後に exportfs -ra コマンドを実行する．処理の終了時も同様にパスが通知され，/etc/exports の該当項目の削除と exportfs -ra コマンドの実行による反映を行う．計算ホストはジョブ実行依頼を行った VM 領域をマウントしてジョブを実行するため，ジョブ実行に必要なファイルは，VM の実行ディレクトリ以下に配置しておく必要がある．

通常，VM の HDD（Virtual Hard Disk; VHD）はストレージサーバ上に保存されており，その領域を各 VM ホストがマウントしている．各 VM ホストはマウントしたディレクトリ上に保存されている VHD を使用して VM を起動する．計算ホストがストレージサーバ上の領域をマウントすれば VM の HDD に直接アクセスできるため，I/O 性能が向上することが予想できるが，VHD で使用されているファイルシステムが通常の ext4 や NTFS などの場合，複数のホストから同時に読み書きが発生すると，ジャーナルに不整合が生じ，データが破損する恐れがある．このため，本研究における実装では VM が NFS サーバを提供し，計算ホストから VM 領域をマウントする方法を採用した．

4.3 サーバエージェントの実装

SA は VM 上で動作する CA からジョブの実行依頼を受け，計算ホスト上でジョブの実行を行う機能を提供する．このため，SA は CA からの通信を解釈しジョブの実行依頼を受け付けるインタプリタと，計算ホスト上でジョブの実行管理を行うワークパイルの 2 つのモジュールで構成する．

インタプリタは CA からの接続およびジョブの依頼を受け，ワークパイルに対して実行ディレクトリおよびジョブの登録を行う．また，ワークパイルに登録されたジョブの出力はインタプリタにリダイレクトされ，CA に転送される．インタプリタが解釈可能なメッセージを表 4.2 に示す．

- **checkin** メッセージは接続ごとに実行され，各接続で使用するファイルを分離するディレクトリを作成する．このディレクトリに VM 領域をマウントし，処理対象となるデータにアクセスできるようにする．
- **mount** メッセージは，引数で与えた VM 上のパスを checkin で作成されたディレクトリで NFS マウントする．以下の exec メッセージはマウントしたディレクトリ上で実行されるため，VM 上のファイルにアクセスできる．
- CA からは **file** メッセージを使用して CA から SA にファイルをアップロードできる．この際，plain もしくは base64 を指定し，平文もしくは Base64 でエンコードされたテキストでファイ

ルを送信できる．ファイルの権限は 644，755 などの形式で指定し，filename で指定したファイル名で保存される．ただし，このメッセージはテスト用に実装されたものであり，5 章の評価では一切使用していない．

- **exec** メッセージは計算ホスト上においてコマンドの実行を行う際に使用する．依頼されたコマンドは checkin メッセージで作成したディレクトリ上で実行される．このディレクトリは先の mount メッセージで VM 上のディレクトリとリンクしているため，VM からは見かけ上，コマンドが VM 上で動作しているように認識される．
- **umount** メッセージはマウント済のディレクトリをアンマウントする．
- **checkout** メッセージは，checkout メッセージでディレクトリを削除する．

ワークパイルはインタプリタから受け付けたジョブをジョブキューへ登録し，順次実行する．ジョブの実行時は，CA から通知されたデータアクセス用のマウントポイントを一時ディレクトリにマウントし，そのディレクトリへ移動してからジョブキューに登録されたジョブを実行する．実行ジョブのファイル I/O は NFS によりマウントされたディレクトリで発生するため，VM の実行ディレクトリに転送される．ジョブの標準出力および標準エラー出力はインタプリタにリダイレクトし，CA に転送する．

4.4 VM に加える変更

VM から計算ホストにジョブを依頼する際はジョブ依頼用コマンドを使用する．ユーザが VM にログインして計算ホストにジョブを依頼する場合は，実行コマンドの先頭に `offld --`を追加すればよいことは 4.2 節で述べた．しかし，Web サーバの CGI のように，ユーザが VM にログインしていない環境で計算ホストにジョブを依頼する場合，あらかじめ CGI を書き換えておくか，目的のコマンドが自動的に計算ホストに実行が依頼されるよう設定しておく必要がある．

計算ホストに実行依頼するコマンドの実態が`/usr/bin/convert` に存在する場合，VM 上で以下の設定を行うことで，`convert` コマンドが自動的に計算ホストに依頼される．なお，ジョブ依頼コマンドは`/usr/local/bin/offld` に配置されているとする．

1. `/usr/bin/convert` を`/usr/bin/convert.orig` にリネームする
2. 図 4.4.1 のようなスクリプトを`/usr/local/bin/auto_offload` として保存する
3. `/usr/local/bin/auto_offload` のシンボリックリンクを`/usr/bin/convert` に作成する

このようにすると，任意のプロセスが`/usr/bin/convert` を実行した際に，図 4.4.1 のスクリプトが実行される．このスクリプトでは，‘`basename $0`‘により実行時の自スクリプトのファイル名からコマンド名を抽出し，スクリプト実行時に設定した引数も`$@`で展開した上で，ジョブ依頼コマンドの引数に設定して実行する．このため，VM に `convert` コマンドがインストールされていなくても，自動的に `convert` コマンドの計算ホストに依頼され，計算ホスト上で実行できる．

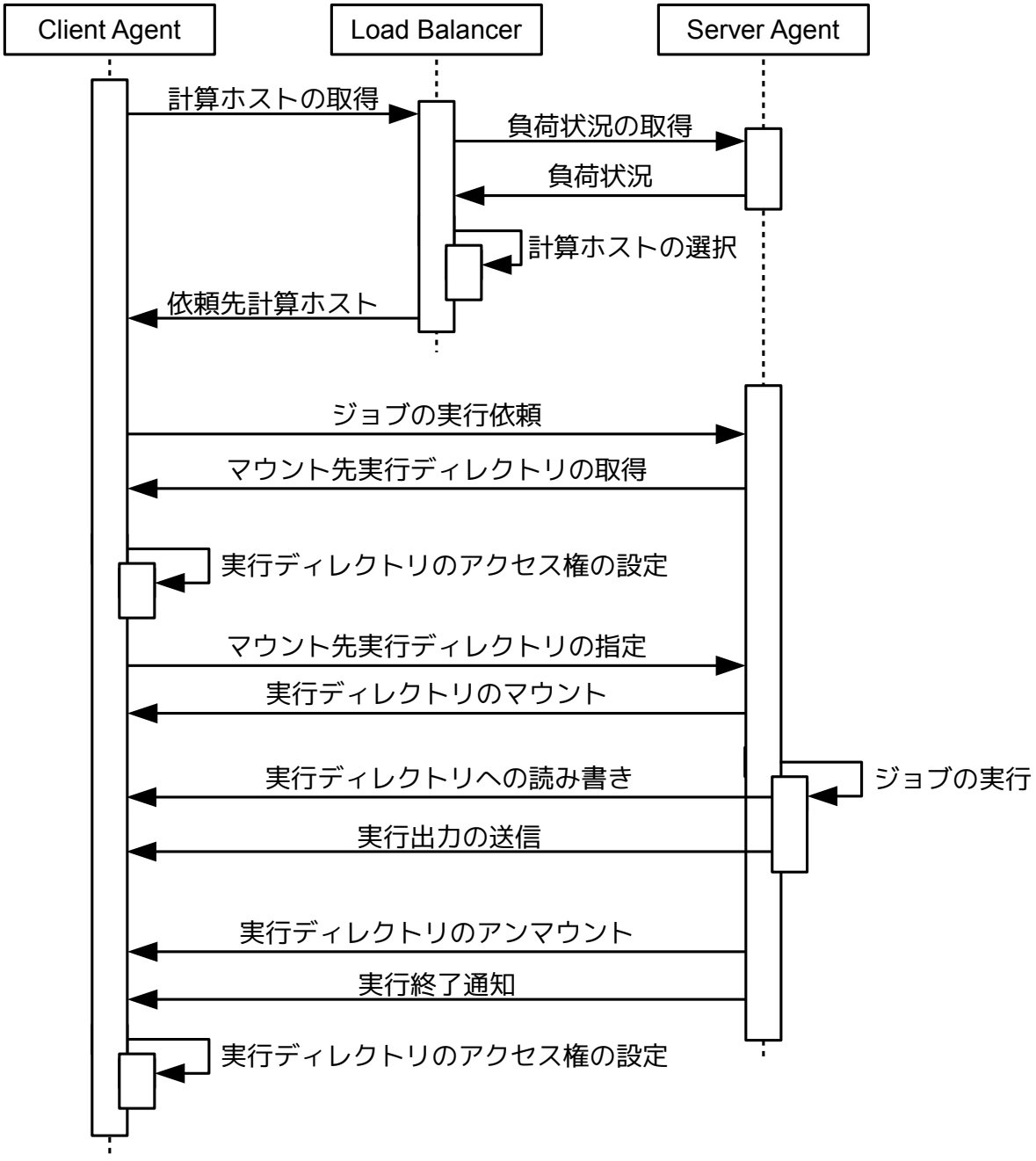

図 4.1.2: CA と SA のシーケンス図

表 4.1: ジョブ依頼コマンドの書式

usage:
$ offld [オプション..] [--] 実行コマンド [実行コマンドの引数]

オプション	内容
-c, --config=<filename>	<filename> から設定ファイルを読み込む
-p, --profile	実行プロファイルを表示する
-H, --host=<host>[:<port>]	ジョブを依頼する計算ホストを指定する
-L, --load-balancer=<host>[:<port>]	ロードバランサを指定する
-n, --no-mount	マウントせずに実行する
-o, --output=<filename>	標準出力と標準エラー出力を <filename> に書き込む
-Q, --no-logging	ログファイルを作成しない
-q, --quiet	出力しない
-d, --debug	デバッグメッセージを表示する
-h, --help	ヘルプを表示する

表 4.2: インタプリタが解釈可能なメッセージ

メッセージ	動作内容
checkin	VM 領域のマウント用ディレクトリの作成
mount <mount point>	VM 領域のマウント
file {plain,base64} <mode> <filename>	ファイルのアップロード
exec <command and arguments>	コマンドの実行
umount	VM 領域のアンマウント
checkout	接続の切断とディレクトリの削除

```
#!/bin/bash

EXEC_COMMAND=‘basename $0‘
COMMAND_ARGS=$@
OFFLOAD_COMMAND=/usr/local/bin/offld

$OFFLOAD_COMMAND -- $EXEC_COMMAND $COMMAND_ARGS
```

図 4.4.1: コマンドを自動的に計算ホストに依頼するスクリプト

第5章　評価

5.1　評価環境

提案する計算資源の動的共有手法の有効性を評価する目的で，図 5.1.1 に示す実験的なクラウド環境を構築した．実験環境は，クラウド管理サーバ 1 台，VM ホスト 4 台，ストレージサーバ 1 台からなる．ゲートウェイサーバは既存のネットワークとプライベートクラウドを接続しており，既存のネットワークを介してプライベートクラウドにインターネットアクセスを提供している．また，上部のネットワークに接続されているノードは一般的なクラウド環境であるが，ゲートウェイサーバを介して 2 台の計算ホストをプライベートクラウドのネットワークに接続し，クラウドを拡張した．計算ホスト 2 台は既存ネットワークに接続されているが，ゲートウェイサーバを介してプライベートクラウド側のホストと自由に通信できる．また，上部のネットワークと下部のネットワークはゲートウェイサーバによって接続されているが，ゲートウェイサーバは十分な処理性能があるため，ネットワークの遅延およびスループットの低下はほとんど発生しない．

上記のようなクラウド環境を構築した上で，各計算ホストで SA を，VM で CA を動作させた．また，クラウド上に表 5.2 に示す環境 D の VM を 1 台作成し，ロードバランサを動作させた．各サーバ環境の構成を表 5.1 に示す．

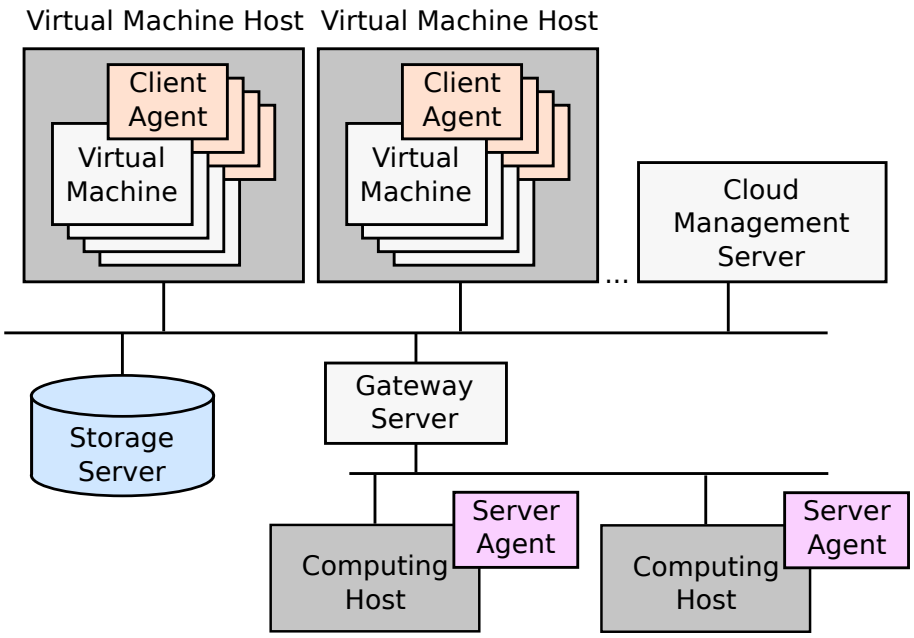

図 5.1.1: 構築したクラウド環境

実験ではすべての処理対象データと制御情報をネットワークを介して伝送するため，ネットワークの速度は計算時間に大きな影響を与える可能性が高い．一方，計算時には主に計算ホストの CPU 時間を消費するため，VM の CPU はジョブの制御にのみ利用される．このため，VM に割り当てる CPU 性能は計算時間に大きな影響を与えないことが予想できる．

これを受けて，評価に用いる VM の構成を表 5.2 に示すように 4 種類用意した．すべての値は上限値であり，リソース使用量が表中の値を超過することはない．また，評価に使用した VM の CPU 数はすべて 1 である．現在，クラウドで利用できる VM のネットワークの多くは，100Mbps

表 5.1: クラウドを構成する物理サーバの構成

VM ホスト：	
OS	CentOS 6.4
CPU	Intel Core i7-3770 @3.40GHz
RAM	32GB
Network	1GbE
Hypervisor	Linux KVM
計算ホスト：	
OS	Ubuntu 12.04
CPU	Intel Core i7-3770 @3.40GHz
RAM	32GB
Network	1GbE
GPU	AMD Radeon HD 6930
クラウド管理サーバ：	
OS	CentOS 6.4
CPU	Intel Core i5-2410 @2.30GHz
RAM	4GB
Network	1GbE
ストレージサーバ：	
CPU	Marvell 2.0GHz
RAM	512MB
HDD	3TB × 4
RAID	RAID10
Network	1GbE
ゲートウェイサーバ：	
OS	CentOS 6.4
CPU	Intel Core i5-2400 @3.10GHz
RAM	8GB
Network	1GbE

および 1Gbps であることから，これら 2 種類のネットワークを用意し，その影響を検証する．また，CPU の動作周波数についても 2GHz 以上かつ複数コア提供される VM が多いが，1GHz，500MHz に動作周波数を抑えた 2 種類の CPU 環境を用意し，比較を行う．

評価には実クラウド上での高負荷な計算処理を想定し，画像処理を採用した．処理対象の画像として，図 5.1.2 に示すカラー画像ファイルを用意した．この画像のサイズは 10000x10000 ピクセルであり，容量は 8MB 程度である．画像処理アプリケーションは，Linux で利用される代表的な画像処理アプリケーションである ImageMagick を利用し，convert コマンドを用いてぼかし処理とリサイズ処理を行う．また，複数の VM から GPU を利用する実験では，AMD が提供する OpenCL SDK である AMD APP SDK にサンプルとして付属する MatrixMultiplication をそのまま利用した．なお，これらのアプリケーションは計算ホストにのみインストールしたため，VM 単体では利用することができない．しかし，本手法を適用することで VM から計算ホストの計算資源とアプリケーションが利用可能であるため，VM からは上述のアプリケーションがインストールされているものとしたジョブの実行が可能である．

図 5.1.2: 実験に使用した画像

表 5.2: 実験環境の VM 設定

Environment	VM CPU clock	VM Memory	Network throughput
A	500MHz	1GB	100Mbps
B	500MHz	1GB	1000Mbps
C	1GHz	1GB	100Mbps
D	1GHz	1GB	1000Mbps

5.2 fio を使用した I/O 性能の予備評価

本手法では，計算ホストから VM のハードディスク（Virtual Hard Disk; VHD）の格納領域まで以下の手順を踏んでアクセスを行う．

1. VM ホストから VHD 格納領域の NFS マウント
2. VM ホストが NFS 領域上に保存された VHD を使用して VM を起動
3. VM は VHD 上にファイルシステムを構成し，その上でファイルを保存
4. 計算ホストは VM が提供する NFS 領域をマウント

上記の各手順では，それぞれ I/O 性能が低下することが広く知られている．手順 3 までは通常のクラウドと同様であるが，本手法では手順 4 が追加されるため，I/O 性能の低下が予想される．このため，評価に先立って，大きな I/O 性能低下がないか予備評価を行った．

予備評価にはディスク I/O ベンチマークとして広く利用されている fio を使用し，VM 上での I/O 性能と，計算ホストから VM 上の領域を NFS マウントした際の I/O 性能をそれぞれ測定した．測定する I/O 性能は，ランダム読み込み，ランダム書き込み，シーケンシャル読み込み，シーケンシャル書き込みの 4 種類とした．また，測定の際はなるべく OS が提供する I/O キャッシュが利用されないように VM および計算ホストでキャッシュをクリアした上で，読み書きがキャッシュを経由せず直接行われるように設定を行った．このため，出力された結果は通常実行時の性能より低く出ており，キャッシュが利用できる状況では得られた結果よりも高い I/O 性能が期待できる．fio の実行の際は表 5.3 に示すパラメータを設定した．3 回の計測結果の平均値を図 5.2.1 に示す．ここで，rd_r，rd_w，sq_r，sq_w はそれぞれランダム読み込み，ランダム書き込み，シーケンシャル読み込み，シーケンシャル書き込みを示す．

図 5.2.1 より，VM が直接 VM の HDD 領域を読み書きする際は VM ホストで I/O がキャッシュされるため，比較的高速な I/O 性能が得られる．また，計算ホストから VM 上の領域に NFS でアクセスする場合，ストレージサーバ，VM ホスト，計算ホストの順に 2 重に NFS マウントされることになる．このため，キャッシュを利用せず直接書き込む設定が作用し，ランダム書き込みおよびシーケンシャル書き込み時の性能が大きく低下している．しかし，通常実行時においては OS の I/O キャッシュを利用できるため，読み書きともに性能低下率が小さく抑えられることが期待できる．

表 5.3: fio 実行時のパラメータ

ioengine	libaio
direct	1
invalidate	1
size	1GB
bs	4k
iodepth	32

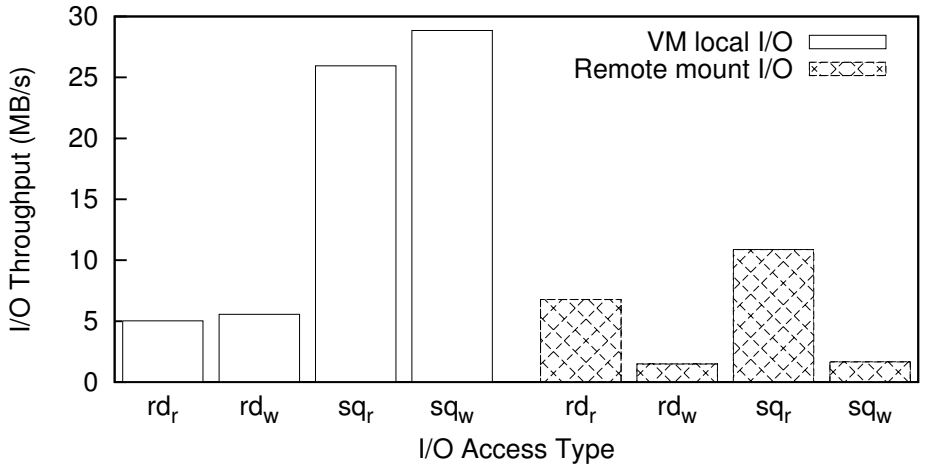

図 5.2.1: VM 領域の I/O 性能

5.3 仮想マシン 1 台における画像処理

VM に本手法を適用した際の基本的な性能向上率とオーバヘッドを評価するため，図 5.1.2 に示す 10000x10000 ピクセルの画像ファイルについて，表 5.2 に示す 4 種類の VM 環境から計算ホストに対してジョブを依頼する評価を行った．なお，計算中におけるライブマイグレーションは発行せず，ジョブの依頼は 1 台の VM から 1 台の計算ホストに順次実行依頼を行った．実行するジョブは 5x5，10x10，20x20 近傍におけるぼかし処理と，元の画像の 5%，10%，20%へのリサイズ処理である．計測は 12 回行い，最大値と最小値を取り除いた 10 データの平均値をグラフ化した．実行時間とオーバヘッドの割合を図 5.3.1 に，その際のオーバヘッドの内訳を図 5.3.2 に示す．各グラフの横軸の A-D は表 5.2 の仮想マシン環境に対応する．図 5.3.1 中の Host は計算ホスト上で同じコマンドをネイティブ実行した際の時間である．また，表 5.4 に計算ホストにジョブを依頼せず，環境 D の VM 上でジョブを実行した際の実行時間を示す．

表 5.4 に示した計算ホストを利用しない環境 D における VM 上での実行時間と，図 5.3.1 を比較すると，VM 上で実行するよりも計算ホスト上を利用した方が短時間で計算が終了している．高速

表 5.4: 各画像処理の出力画像の詳細と実行時間

処理名	画像サイズ	ファイルサイズ	実行時間（秒）	
			環境 D	計算ホスト上
blur5x5	10000x10000	6307KB	778.1	5.8
blur10x10	10000x10000	5219KB	810.3	7.6
blur20x20	10000x10000	4379KB	806.4	11.1
resize5p	500x500	90KB	28.6	1.7
resize10p	1000x1000	238KB	26.1	7.6
resize20p	2000x2000	707KB	37.7	11.3

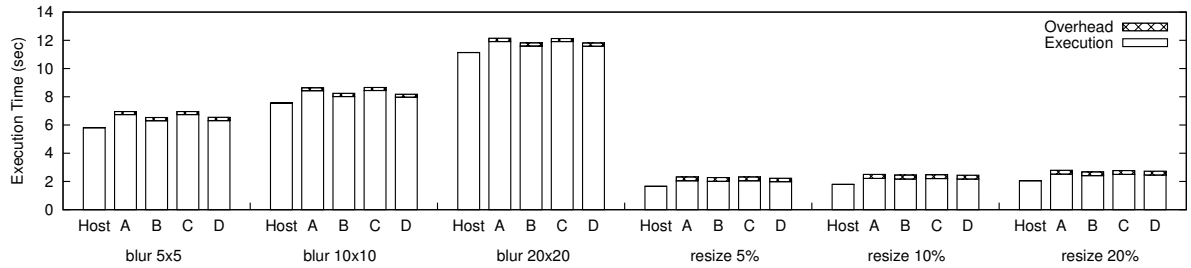

図 5.3.1: 10000x10000 ピクセルの画像について処理を行った際の実行時間

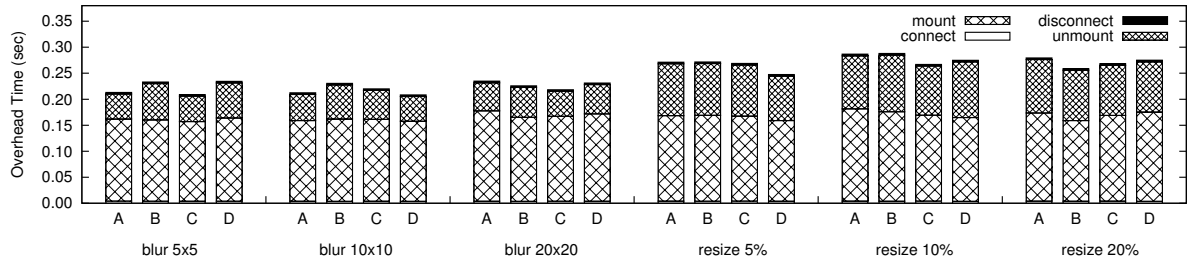

図 5.3.2: 10000x10000 ピクセルの画像について処理を行った際のオーバヘッドの内訳

化率は大きいもので 100 倍程度，小さいもので 3-4 倍程度である．評価に使用した ImageMagick は OpenMP による CPU の並列化の他，OpenCL による GPU を利用した計算に対応している．しかし，VM 上では CPU が仮想化されるため利用できる命令数に制約を受ける他，GPU が利用できないため，計算が並列化されない．一方，計算ホストは CPU および GPU，豊富なメモリ資源を利用できるため，大幅な高速化につながっている．

図 5.3.1 を見ると，計算ホスト上でネイティブ実行した場合と VM からジョブを依頼した場合で実行時間がほぼ同等であり，環境 A，C と比べてネットワーク速度の早い環境 B，D は，実行時間が短い．コマンド実行時に行われる処理のうちネットワークが関与するのは画像の読み込みと書き込みの部分のみであるから，これはファイル I/O に依存する差である．実行時間の大部分が計算時間であるアプリケーションでは同様の結果が得られるはずである．また，CPU 動作周波数が同じ 500MHz である環境 A，C と，1000MHz である B，D をそれぞれ比較すると，有意な差はみられない．これは，VM 内における計算時間がほとんどなく，VM に割り当てた CPU 動作周波数が実行時間にほとんど影響しないことを示す．以上の結果より，VM に割り当てる計算資源のうち，ネットワーク資源は計算時間に影響するため多く割り当てた方がよく，CPU 動作周波数は計算時間にほとんど影響しないため，割当量をさらに削減できる．

図 5.3.2 を見ると，どのジョブを実行した場合でも必要となるオーバヘッドが 0.2 秒から 0.3 秒程度で収まっている．これは，目的のジョブの実行時間のうち大部分が計算時間であり，VM 上で実行した際にかかる実行時間が数秒以上かかるようなアプリケーションならば十分に許容できる．このため，計算主体となるアプリケーションの実行においては VM 上で実行するよりも計算ホストにジョブを依頼する方が有利である．また，オーバヘッドのうち主要な処理時間がデータ領域のマウントに関する部分であるため，利用するファイルシステムを NFS から別のファイルシステムに変更することでジョブ依頼時のオーバヘッドを削減できる可能性がある．

5.4 複数台の仮想マシンでの評価

次に，1 台の計算ホストを複数の VM で共有した際の動作を確認するため，環境 D の VM を 10 台用意し，複数の VM から 1 台の計算ホストにジョブを依頼した場合の実行時間を測定した．この実験では，AMD APP SDK のサンプルに含まれる MatrixMultiplication を実行したため，計算時に GPU が利用された．実験は，複数の VM について dsh コマンドを用いて ssh のコネクションを確立し，計算ホストにジョブを依頼するコマンドを実行することで，複数の VM からほぼ同時に計算ホストにジョブを依頼した．ただし，計算ホスト上でのジョブの同時実行数は最大 1 つまでとして実験を行った．なお，このジョブを VM 上で計算すると，5 秒程度の時間を要することを事前に確認した．

図 5.4.1 は，計算ホストがすべてのジョブを完了するまでの時間を VM 数で割ったグラフである．計算ホストが受け付けるジョブ数が増えると，ジョブの平均実行時間が減少する．基本的なジョブ実行は，4.1 節に示したように以下のように実行される．

1. VM から計算ホストへの接続
2. 計算ホストから VM 領域のマウント
3. 計算ホスト上でのジョブ実行
4. VM 領域のマウント解除
5. 接続の切断

上記手順のうち 3 番目の計算ホスト上でのジョブ実行以外の部分がオーバラップされ，パイプライン実行されている．このため，ジョブ数が増えるとオーバヘッドが実行時間に隠蔽され，ジョブあたりのターンアラウンドタイムが短縮される．

図 5.4.2 にジョブを依頼してから完了までにかかる時間の平均値と VM 数の関係を示す．VM 数の増加に伴い，ジョブの待ち行列の待機時間が長くなるため，平均実行時間も増加する．しかし，計算ホストにジョブを依頼せずに VM 上でジョブを実行した時間と比較すると，10 台の VM が同時にジョブを依頼した場合の平均実行時間の方が短い．また，実行順が後になった場合は VM の台数分だけジョブの実行待ちが発生するため，このジョブに関しては 5 台以上の VM から同時に実行する際は，計算ホストにジョブを依頼せず，VM 上でジョブを実行した方が短時間で計算が終了する．しかし，画像処理のように計算サイズが大きい場合は 10 台程度同時に計算ホストにジョブを依頼しても，VM 上で実行するより計算ホストでの実行の方が早く終わると予想できる．

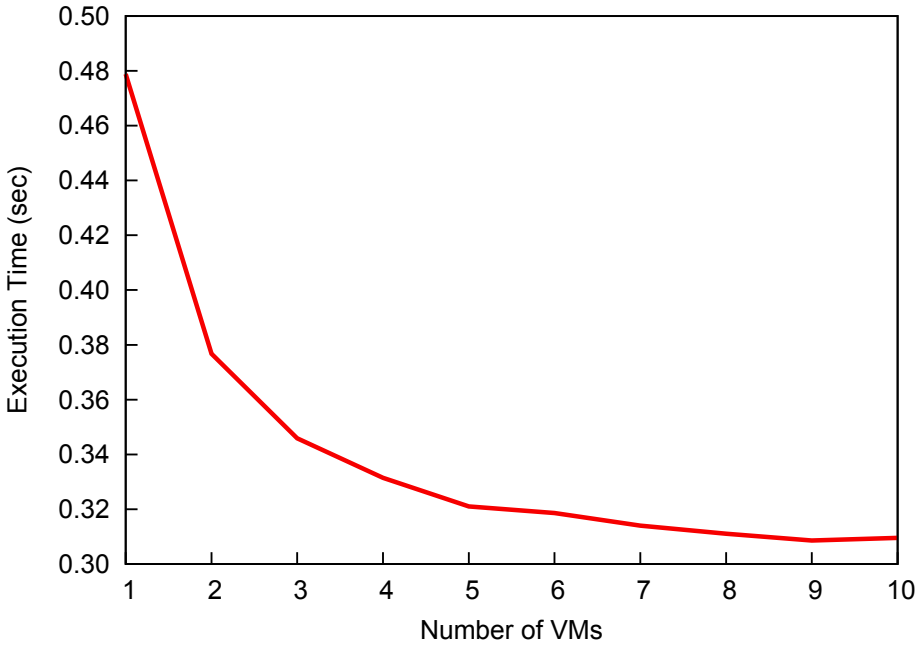

図 5.4.1: ジョブあたりの平均実行時間

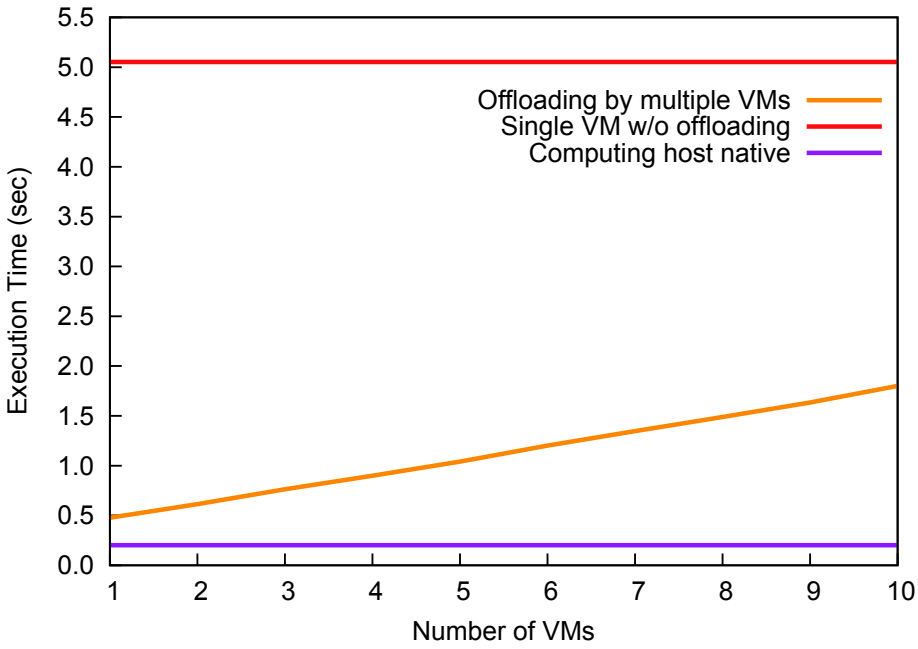

図 5.4.2: ジョブの待ち時間を含む実行時間の平均値

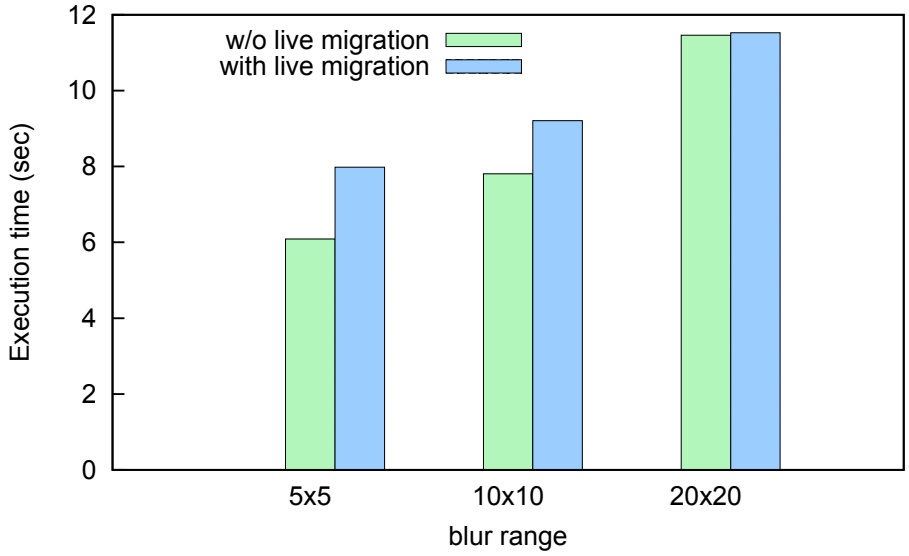

図 5.5.1: ライブマイグレーションの有無による実行時間の変化

5.5 ライブマイグレーション時の評価

クラウド上では VM の負荷状況や VM ホストの保守作業の際に，VM が別の VM ホストにライブマイグレーションされる状況がある．この際，計算資源の状態は維持されるため，VM からはライブマイグレーションしていることは認識されない．また，ネットワークは一時的に切断されるが，TCP 通信においては再送が発生するため，通信は切断されない．ライブマイグレーションが発行された場合，VM ホスト間で VM の CPU 状態とメモリ状態がコピーされ，コピーが完了した瞬間に VM の稼働位置が移行される．また，ライブマイグレーション中はメモリコピーによるトラフィックが発生するため，ネットワークを使用するアプリケーションは性能低下の可能性がある．

そこで，本手法による計算実行時にライブマイグレーション命令が発行された際の計算性能を評価する実験を行った．環境 D の VM から計算ホストにジョブを依頼した直後に VM のライブマイグレーション命令を発行し，ジョブの実行中に現在の VM ホストから別の VM ホストに VM を移行した際の計算時間を測定した．ライブマイグレーションの有無における blur の適用範囲と実行時間の関係を図 5.5.1 に示す．

計算サイズが大きくなりジョブの実行時間が長くなるにつれ，マイグレーションの有無による実行時間の差が小さくなっている．これは，blur 処理を依頼した際の計算ホストの動作によるものである．計算ホストがジョブを受け付けると，計算ホスト内でコマンドが実行される．blur を適用するコマンドは，ソース画像をファイルから読み込み，メモリに展開した後，メモリ上の画像に blur 処理が適用される．ファイルに結果が書き込まれるのは blur 処理の完了後であるため，計算時間とライブマイグレーションがオーバラップされ，計算時間が長くなると VM のライブマイグレーションの影響が小さくなっている．なお，ライブマイグレーション時に処理が伸びているのは，計算終了後に結果の画像を書き込む際，ライブマイグレーションによって VM のネットワーク的な位置が変更された結果がネットワークに反映されておらず，書き込み時に待ちが発生したためである．このため，頻繁にファイルアクセスが発生するアプリケーションを実行中にライブマイグレーションを実行すると，実行時間にかかる影響が大きくなると予想される．

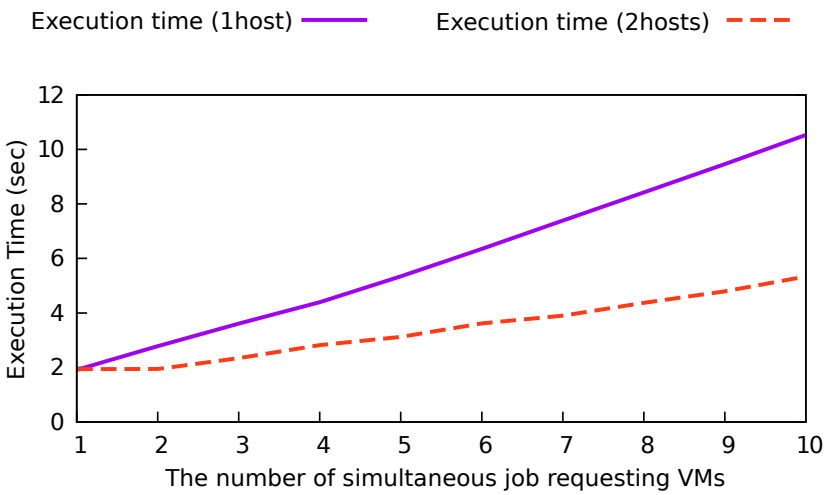

図 5.6.1: 計算ホスト台数によるジョブ実行時間（resize5p）

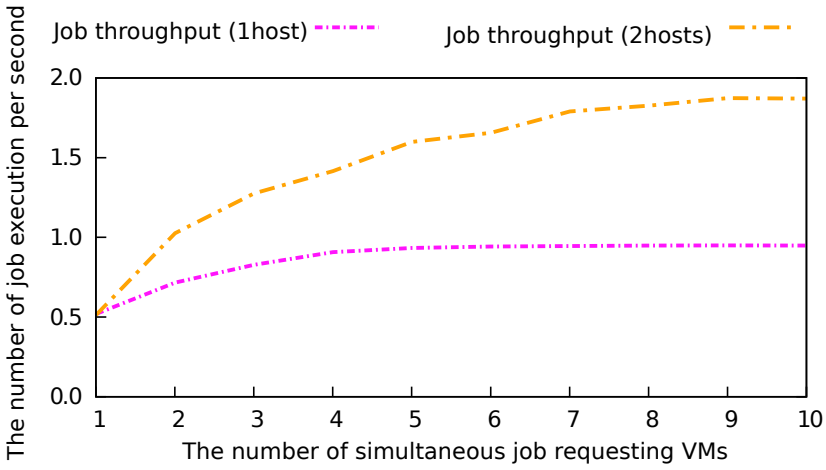

図 5.6.2: 計算ホスト台数によるジョブ実行スループット（resize5p）

5.6 計算ホスト台数による評価

本手法では，計算ホストを新たに追加することで，クラウド環境全体の性能向上が期待できる．計算ホストが 1 台の環境と 2 台の環境において，環境 D の VM1-10 台から同時にジョブを依頼し，計算ホスト台数と実行時間の関係を評価した．ジョブの依頼はロードバランサにより各計算ホストに分散される．評価には blur5x5 と resize5p の 2 つの画像処理を使用した．各画像処理における実行結果の 10 回の実行時間の平均値を図 5.6.1,5.6.3 に，実行スループットの平均値を 5.6.2,5.6.4 に示す．グラフの横軸は同時にジョブを依頼した VM 数，左軸はそれぞれジョブ全体の実行時間，1 秒あたりのジョブの処理数を示す．

図 5.6.1,5.6.3 を見ると，計算ホスト台数が 1 台から 2 台になると resize5p, blur5x5 それぞれでジョブの実行時間がほぼ半分になっている．また，図 5.6.2,5.6.4 を見ると，resize5p, blur5x5 それぞれでジョブの実行スループットが 2 倍になっている．これらの結果から，計算ホスト台数を増

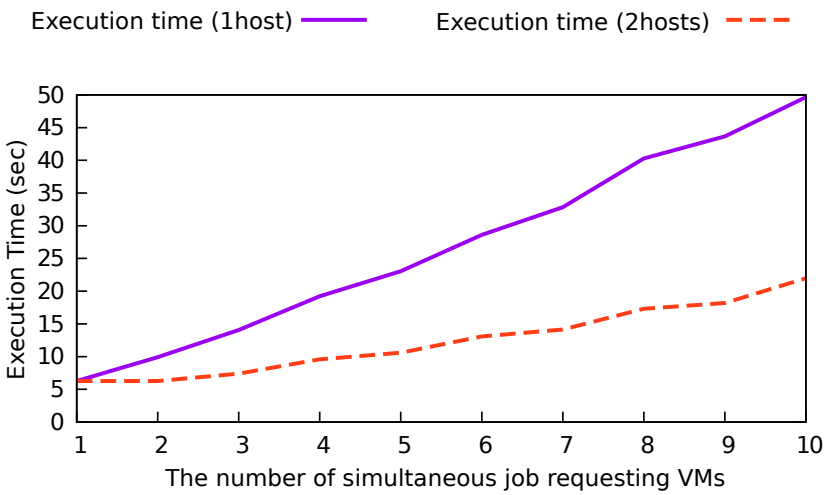

図 5.6.3: 計算ホスト台数によるジョブ実行時間（blur5x5）

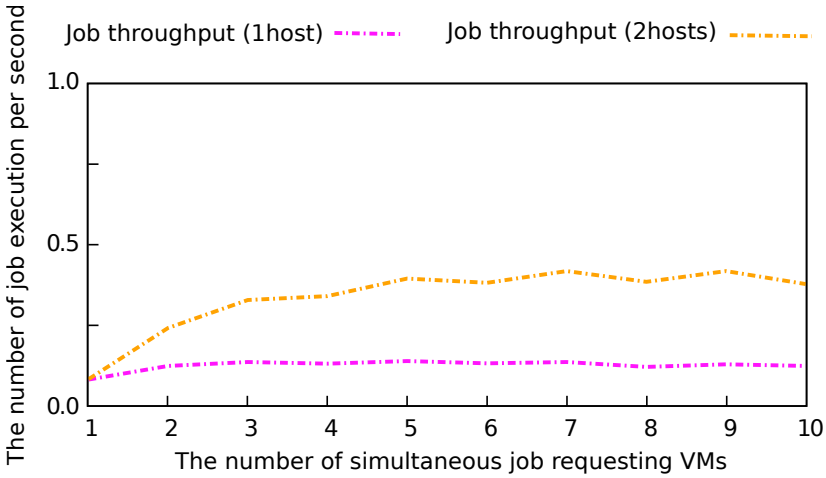

図 5.6.4: 計算ホスト台数によるジョブ実行スループット（blur5x5）

強することで，計算性能が概ね線形に向上することが予想される．

次に，図 5.6.2,5.6.4 に示す実行スループットのグラフを見ると，resize5, blur5x5 それぞれのグラフにおいて 1 ジョブでの実行よりも 2 つ以上のジョブを同時に実行する方がスループットが高くなっている．これは，計算ホストに搭載された CPU は物理 4 コア論理 8 コアのものであるため，1 つの CPU で同時に複数のプロセスを実行できる．多数のジョブを実行すると，2 つ以上コア数を活用できることから，マルチコアプロセッサを搭載した計算ホストに対しては複数のジョブを同時に依頼することで計算資源の利用率が改善できる．

また，図 5.6.2,5.6.4 を比較するとグラフ形状が類似しており，図 5.6.2 のグラフのジョブ依頼 VM 数を増加させると図 5.6.4 のグラフのように一定の実行スループットが維持されることが期待できる．また，これら 2 つのグラフの類似性から，blur のグラフの VM 数を増加させると resize のグラフ形状と類似するため，図 5.6.1 と図 5.6.3 のグラフにおいても同様に，VM 数を増加させても大きなオーバヘッドを受けることなく実行時間が線形に伸びることが予想できる．

第6章　議論

6.1　大規模クラウドへの対応

提案した計算資源の動的共有手法では，VM がライブマイグレーション中でもジョブの実行が継続可能である．計算ホストはネットワークを介して接続できればよいため，既存のクラウド環境に本手法を導入するのは容易である．計算ホストが複数ある場合でもロードバランサにより適切に処理が分散できるため，クラウドの規模に応じて計算ホストを追加すればよい．また，計算ホストの負荷状況に応じて，クラウドに設置する計算ホスト台数を動的に変更することで，計算資源の利用効率の高い大規模なクラウドを実現できる．

クラウドの大規模化に伴い，VM と計算ホスト間の通信性能が低下し，計算性能が低下するおそれがある．しかし，大規模なクラウド環境は通常複数のサーバラックで構成されており，ラック内は高速な通信環境が提供される．このため，大規模なクラウドにおいては，サーバラック内の物理サーバを VM ホストと計算ホストと分離することで，大きな I/O 性能の低下は表出しないと予想できる．また，将来的にはサーバラック内のネットワークだけでなく，データセンタ内のネットワークも十分高速になるため，I/O ボトルネックは低減され，高い計算性能を容易に利用することができるようになる．

6.2　計算資源の測定方法

RaaS 型クラウドを実現するためには，利用した計算資源を詳細に測定できる必要がある．しかし，現在の実装では，計算ホスト上で実行するジョブに割り当てる計算資源量は設定していない．Linux の control groups (cgroups) [22] 機能を用いると，アプリケーションに割り当てる CPU 動作周波数，メモリ容量，ネットワーク帯域，ディスク I/O 速度などの値を設定できる．このため，計算ホストで実行するジョブに割り当てる計算資源量を cgroups で設定すれば，実行時間と割り当てた計算資源の測定が可能である．また，課金のために保存した計算資源利用のログデータを利用して，各 VM における計算資源の利用特性を学習できる．その結果，ロードバランサがジョブを依頼する計算ホストを決定する際に計算資源の利用率が向上するよう選択を行うことで，クラウドの計算資源利用がさらに効率化できる．

6.3　提案手法におけるセキュリティ

現在はセキュリティを考慮した実装になっていないため，悪意あるジョブが計算ホストで実行された場合，他のユーザ領域にアクセスされる危険がある．このため，実運用の際は SELinux などの強制アクセス制御機構の利用により適切なアクセス権を設定する必要がある．なお，クラウドにおけるネットワークはユーザ毎に分離されるため，ネットワーク中のパケットを盗み見られ

る可能性は低い．しかしながら，グループ内や組織内で相互に疎通可能なネットワークが用意される場合もあるため，ネットワークの構成によっては通信内容を暗号化する必要がある．

また，悪意のあるユーザやアプリケーションが計算ホストに大量のジョブを発行し，計算資源を占有する DoS 攻撃が予想できる．現在提供されているクラウドでも，VM を大量に起動したり高い CPU 使用率を維持するなどの状況が発生する．これらの状況は，1 ユーザが使用できる計算資源量に上限を設定することである程度回避できるが，計算ホストが依頼を受けたジョブに悪意があるかどうかの判断は困難であるため，完全な解決は難しい．

6.4 計算ホストで動作するジョブに関する制約

VM から計算ホストに依頼するジョブは，計算ホストにあらかじめインストールされている必要がある．現在はジョブ実行時に設定したディレクトリのみが計算ホストにマウントされるため，特定の場所から設定ファイルを読み出すようなプログラムの利用には制約がある．設定ファイルを引数で指定可能なプログラムの場合，あらかじめ設定ファイルを実行ディレクトリにコピーしておき，引数で指定するようにする必要がある．このため，計算ホストで実行するジョブの I/O を監視し，実行ディレクトリ外のファイルにアクセスを試みた場合は VM にリダイレクトするような仕組みが必要である．

第7章　結論

本研究では，計算資源を効率よく利用できる RaaS 型クラウドの実現のために，VM と計算ホストを分離して適切に計算資源を利用する手法を提案し，評価を行った．

提案した計算資源の動的共有手法では，VM で発行した高負荷な処理を計算ホストで実行することで，アプリケーションの実行時のみ VM に必要な計算資源を動的に割り当てることができる．割り当てる計算資源は CPU やメモリなど既存のクラウドで通常利用できる範囲に限らず，GPU やアクセラレータなどの PCI 計算資源も高い計算性能を維持したまま利用可能である．その結果，VM に割り当てる計算資源を縮小できるため，1 台の VM ホストに収容できる VM 数が向上し，余剰 VM ホストを計算資源として利用することで新たに多数のノードを追加することなく計算ホストをクラウドに設置できる．また，既存のアプリケーションに対する変更が必要なく，そのまま利用可能であるため，アプリケーションのアップデートやパッチの適用などのメンテナンスが容易に実行できる．

評価実験では，計算ホストを利用するオーバヘッドが小さく，VM から高い計算性能を利用できることを確認し，複数の VM から 1 台の GPU を共用できること，ライブマイグレーション中の計算が低オーバヘッドで実行できること，複数の計算ホストとロードバランサを設置した評価では，設置した計算ホスト台数分の性能向上が確認した．その結果，計算資源を追加する際には計算ホストを追加すればよく，クラウドの大規模化に対応できることを示した．

今後は，計算ホストに依頼したジョブが VM 上の任意のファイルにアクセスできるよう，ジョブの I/O をインターセプトして VM に転送する機構を開発し，実用性の向上を目指す．

謝辞

本研究を進めるにあたり，ご指導，ご助言をいただきました，吉永努教授に感謝の意を表します．ゼミでの議論や論文の添削通し，多くの示唆を頂きました．また，同講座入江英嗣准教授，吉見真聡助教にも研究を進めるに当たり多くの助言をいただき，研究がより良いものとなりました．ありがとうございました．日常の議論を通し，多くの指摘を下さいました吉永研究室・入江研究室の先輩方，同期の皆様，後輩の皆様に感謝します．本当にありがとうございました．

参考文献

[1] Peter Mell and Timothy Grance. The NIST definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology, Gaithersburg, MD, 9 2011.

[2] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: A consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp. 41–50, 2009.

[3] Amazon web services. http://aws.amazon.com.

[4] Heroku. https://www.heroku.com.

[5] Dropbox. https://www.dropbox.com.

[6] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. The resource-as-a-service (RaaS) cloud. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing*, HotCloud'12, pp. 12–12, 2012.

[7] John Borhek. Cpu overcommitment vmware vsphere 5.1 - vmsources. https://www.vmsources.com/resources/doc_download/49-cpu-overcommittment-in-vmware-vsphere-51.

[8] M. Schwidefsky, H. Franke, R. Mansell, H. Raj, D. Osisek, and J. Choi. Collaborative memory management in hosted linux environments. In *Proceedings of the Linux Symposium*, pp. 313–328, 2006.

[9] A. Arcangeli, I. Eidus, and C. Wright. Increasing memory density by using ksm. In *Proceedings of the Linux Symposium*, pp. 19–28, 7 2009.

[10] Cloudsigma. https://www.cloudsigma.com.

[11] M.S. Vinaya, N. Vydyanathan, and M. Gajjar. An evaluation of CUDA-enabled virtualization solutions. In *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference*, pp. 621–626, 2012.

[12] Chao-Tung Yang, Hsien-Yi Wang, Wei-Shen Ou, Yu-Tso Liu, and Ching-Hsien Hsu. On implementation of GPU virtualization using PCI pass-through. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference*, pp. 711–716, 2012.

[13] 鈴木勇介, 加藤真平, 山田浩史, 河野健二. GPU の完全仮想化. 情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム], Vol. 2013, No. 14, pp. 1–6, Jul 2013.

[14] Miao Yu, Chao Zhang, Zhengwei Qi, Jianguo Yao, Yin Wang, and Haibing Guan. VGRIS: Virtualized GPU resource isolation and scheduling in cloud gaming. In *ACM Symposium on High-Performance Parallel and Distributed Computing 2013*, 2013.

[15] Hector Marinez. Top Enterprise Technology Companies Embrace NVIDIA GRID.

[16] Lin Shi, Hao Chen, Jianhua Sun, and Kenli Li. vCUDA: GPU-accelerated high-performance computing in virtual machines. *IEEE Transactions on Computers*, Vol. 61, No. 6, pp. 804–816, 2012.

[17] J. Duato, A.J. Pena, F. Silla, R. Mayo, and E.S. Quintana-Orti. rCUDA: Reducing the number of GPU-based accelerators in high performance clusters. In *High Performance Computing and Simulation (HPCS), 2010 International Conference*, pp. 224–231, 2010.

[18] C. Reano, A.J. Pea, F. Silla, J. Duato, R. Mayo, and E.S. Quintana-Orti. CU2rCU: Towards the complete rCUDA remote GPU virtualization and sharing solution. In *High Performance Computing (HiPC), 2012 19th International Conference*, pp. 1–10, 2012.

[19] Cong Wang, Tao Jiang, and Rui Hou. V-OpenCL: a method to use remote GPGPU. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, ICS'13, pp. 493–494. ACM, 2013.

[20] A. Kawai, K. Yasuoka, K. Yoshikawa, and T. Narumi. Distributed-shared CUDA: Virtualization of large-scale GPU systems for programmability and reliability. In *FUTURE COMPUTING 2012, The Fourth International Conference on Future Computational Technologies and Applications*, pp. 7–12, 2012.

[21] M. Oikawa, A. Kawai, K. Nomura, K. Yasuoka, K. Yoshikawa, and T. Narumi. Ds-cuda: A middleware to use many gpus in the cloud environment. In *Proceedings in High Performance Computing, Networking, Storage and Analysis (SCC)*, pp. 1207–1214, Nov 2012.

[22] B. Singh and V. Srinivasan. Containers: Challenges with the memory resource controller and its performance. In *Proceedings of the Linux Symposium*, Vol. 2, pp. 209–222, 7 2007.

[23] CloudStack Documentation. http://docs.cloudstack.apache.org/.

付 録A　CloudStackのインストール手順

A.1　CloudStackの概要

CloudStack はクラウドを構築するためのオープンソースソフトウェアである．CloudStack は現在 XenServer，Linux KVM，VMware vSphere，Hyper-V など複数のハイパーバイザに対応しており，これらのハイパーバイザを使用して図 A.1.1 に示すような共通の WebUI から VM の作成を行うことができる．また，REST API による操作が可能であるため，CloudStack 用のコマンドラインツールである CloudMonkey を利用することで，ターミナルから CloudStack を操作できる．

CloudStack は Cloud.com 社により開発されていたが，2011 年 7 月に米 Citrix 社が Cloud.com 社を買収した．その後 2012 年 4 月に Citrix 社が CloudStack のソースコードを Apache Software Foundation (ASF) に寄贈し，オープンソースのクラウド基盤ソフトウェアとして無償で利用できるようになった．

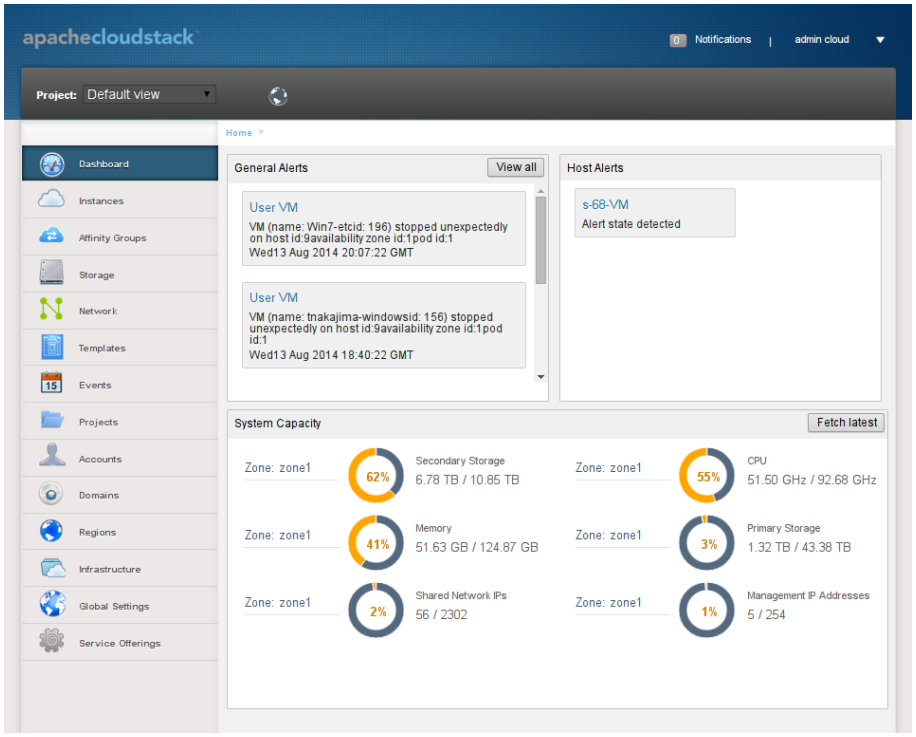

図 A.1.1: ログイン後のダッシュボード画面

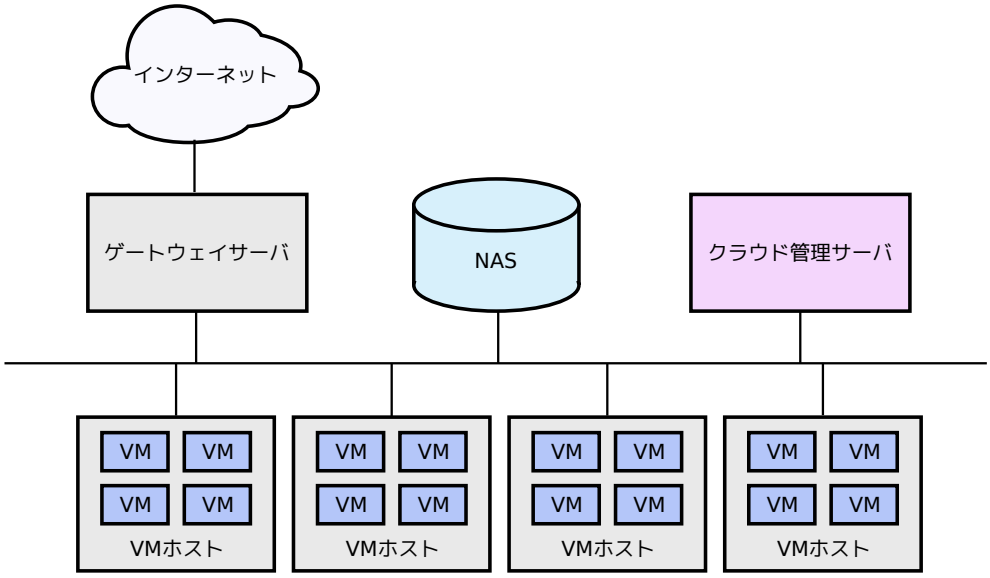

図 A.2.1: 構築する基本構成のクラウド

A.2 CloudStack のインストールの流れ

基本的なインストール方法は CloudStack の公式ドキュメントに従う [23]．本付録では図 A.2.1 に示すように VM ホスト 4 台，クラウド管理サーバ 1 台を対象としてインストールを行う．VM の HDD を格納する NFS 領域を NAS 上に事前に用意しておく．CloudStack は KVM のハイパーバイザノードとして CentOS と Ubuntu にそれぞれ対応しているが，ここでは CentOS 6.5 を用いたクラウド環境を構築する．

インストールするにあたり，すべてのノードで OS のクリーンインストールを行う．その後，クラウド管理サーバと VM ホストに分けて必要なパッケージのインストールと設定作業を行う．これらのインストール作業が終了した後，CloudStack の運用時に使用する「システム VM」と呼ばれる VM のテンプレートの取得および展開作業を行い，CloudStack システムのセットアップを行う．

以下では，クラウド管理サーバの IP アドレスを 10.40.0.1，VM ホストの IP アドレスを 10.40.0.[11-14]，NAS の IP アドレスを 10.40.0.253，ゲートウェイの IP アドレスを 10.40.0.254，作成する VM の IP アドレスを 10.40.1.[1-254] としてクラウドを構成する．また，NAS には/primary と/secondary という NFS マウントポイントが作成し，ネットワーク上のノードから以下のコマンドでマウントできるよう事前に設定しておく．

```
mkdir -p /mnt/primary
mkdir -p /mnt/secondary
mount -t nfs 10.40.0.253:/primary /mnt/primary
mount -t nfs 10.40.0.253:/secondary /mnt/secondary
```

A.3 CentOS のインストールと基本設定

CentOS は minimal 構成でインストールを行い，/etc/hosts を編集して下記コマンドで Fully Qualified Domain Name (FQDN) が出力されるよう設定する．

```
hostname --fqdn
```

次に，下記コマンドを実行して SELinux を無効にしておく．SELinux はセキュリティを高める上で重要な機能であるが，CloudStack は SELinux を有効にした状態では動作しない．このため，CloudStack を構成するノードではこの機能に無効にし，ファイアウォールなど別の機能を用いてセキュリティを確保する必要がある．

```
sed -i -e 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
setenforce 0
```

その後，以下のコマンドを実行し，CentOS に CloudStack のリポジトリ情報を登録する．CloudStack は CentOS と Ubuntu 向けにパッケージリポジトリが公開されており，これらを登録することで yum コマンドや apt-get コマンドなどのパッケージ管理ツールを使用したインストールが可能になる．パッケージ管理ツールを使用しないインストールも可能であるが，手順が煩雑になるため，リポジトリを使用したインストールを行う．

```
echo "[cloudstack]
name=cloudstack
baseurl=http://cloudstack.apt-get.eu/rhel/4.4/
enabled=1
gpgcheck=0" > /etc/yum.repos.d/CloudStack.repo
```

最後に，NFS 共有領域にアクセスするための NFS デーモンをインストールし，起動する．各ノードは NAS 上に用意された VM の HDD 格納領域に NFS でアクセスできる必要がある．

```
yum install ntp
service ntpd start
chkconfig ntpd on
```

A.4 クラウド管理サーバのインストール

クラウド管理サーバは，クラウド上に存在する VM ホストの計算資源を管理し，ユーザに各ノードの計算資源を容易に利用できる WebUI を提供する．また，VM の情報などを保持しており，必要に応じて HA 機能や保守に関連する機能を提供する．

まず，/etc/sysconfig/network-scripts/ifcfg-eth0 を以下のように編集し，固定 IP を設定する．

```
# /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
TYPE=Ethernet
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
IPADDR=10.40.0.1
NETMASK=255.255.255.0
GATEWAY=10.40.0.254
DNS1=129.250.35.250
DNS2=8.8.8.8
```

次に，yum コマンドを使用して CloudStack の管理サーバとデータベースサーバをインストールする．

```
yum install cloudstack-management mysql-server
```

その後，データベースサーバの設定ファイル/etc/my.cnf を編集し，datadir=の次の行から以下を挿入する．

```
innodb_rollback_on_timeout=1
innodb_lock_wait_timeout=600
max_connections=350
log-bin=mysql-bin
binlog-format = 'ROW'
```

設定ファイルの編集後，データベースサーバを起動しセキュリティの設定を行う．ここではデータベースにアクセスする root のパスワードを dbpassword と設定する．

```
service mysqld start
chkconfig mysqld on
mysql_secure_installation
```

以下のコマンドを実行し，データベースに CloudStack で使用するユーザの作成と基本的なテーブルの作成および初期化を行う．

```
cloudstack-setup-databases cloud:password@localhost\
 --deploy-as=root:dbpassword
```

その後，クラウド管理サーバの初期化を行う．ここでは SELinux の設定確認やファイアウォールの自動設定が行われる．2 行目で OS 起動時に自動的に起動するよう設定している cloudstack-management というサービスが CloudStack の管理を行うサービスの実態である．

```
cloudstack-setup-management
chkconfig cloudstack-management on
```

A.5 VM ホストのインストール

VM の作成と管理に必要なパッケージのインストールを行う．インストールパッケージの中の cloud-agent が CloudStack で提供される VM ホストの管理用エージェントである．このパッケージをインストールすることで，クラウドに VM ホストを登録し，VM が提供できるようになる．

```
yum install qemu-kvm cloud-agent bridge-utils vconfig
```

/etc/libvirt/libvirtd.conf に以下を追記する．この設定は CloudStack の動作に必要であるが，自動で実行されないため，手動で設定ファイルを編集する．

```
listen_tls = 0
listen_tcp = 1
tcp_port = "16509"
auth_tcp = "none"
mdns_adv = 0"
```

以下のコマンドを実行して libvirtd サービスの設定と反映を行う．libvirt は Xen や KVM などのハイパーバイザの違いによるコマンドや書式の差を吸収する抽象化ツールである．

```
sed -i -e 's/#LIBVIRTD_ARGS="--listen"/LIBVIRTD_ARGS="--listen"/g'\
 /etc/sysconfig/libvirtd
service libvirtd restart
```

以上の設定が終了したら，ネットワークの設定を行う．/etc/sysconfig/network-scripts 以下の ifcfg-eth0 と ifcfg-cloudbr0 を以下のように編集する．この設定により，cloudbr0 という名前のブリッジインタフェースが作成され，eth0 がブリッジに追加される．CloudStack は cloudbr0 を対象にネットワークの自動設定を行うため，このインタフェース名は変更できない．

システムによっては eth0 ではなく em0 や p2p1 などの名前でネットワークインタフェースが認識されていることがある．しかし，eth0 以外の名前で認識されている場合は正しく設定が行われない可能性がある．このため，/etc/udev/rules.d/70-persistent-net.conf を編集し，インタフェースを認識した際に設定される識別子を変更しておく必要がある．

```
# /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
TYPE=Ethernet
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
BRIDGE=cloudbr0
```

```
# /etc/sysconfig/network-scripts/ifcfg-cloudbr0
DEVICE=cloudbr0
TYPE=Bridge
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=static
IPADDR=10.40.0.11
NETMASK=255.255.255.0
GATEWAY=10.40.0.254
DNS1=129.250.35.250
DNS2=8.8.8.8
```

設定後，ネットワークを再起動し，変更を適用する．

```
service network restart
```

A.6 システム VM テンプレートのセットアップ

CloudStack の管理は上記手順でインストールしたクラウド管理サーバの他に，VM のテンプレートや ISO イメージが格納される NFS 領域へのアクセスを提供するセカンダリストレージ VM，ユーザに WebUI 経由で VM の画面アクセスを提供するコンソールプロキシ VM，ユーザの VM にネットワークアクセスを提供する仮想ルータなどの VM が利用される．これらの VM はシステム VM と呼ばれ，CloudStack で構成するクラウドの運用上必須な VM である．システム VM は VM のテンプレートとして提供されており，CloudStack のセットアップ前に取得して NFS 共有領域に展開しておく必要がある．

システム VM を格納する NFS 領域にアクセスするため，管理サーバで以下のコマンドを実行する．

```
service rpcbind start
chkconfig rpcbind on
service nfs start
chkconfig nfs on
mkdir -p /mnt/primary
mkdir -p /mnt/secondary
mount -t nfs 10.40.0.253:/primary /mnt/primary
mount -t nfs 10.40.0.253:/secondary /mnt/secondary
```

その後，システム VM テンプレートのインストールを行う．テンプレートの取得ツールは /usr/share/cloudstack-common/scripts/storage/secondary/ 以下に配置されている．また，KVM 以外のハイパーバイザを使用する場合は下記コマンドで設定する引数が変わるため，注意が必要である．

```
cd /usr/share/cloudstack-common/scripts/storage/secondary/
./cloud-install-sys-tmplt \
-m /mnt/secondary \
-u http://cloudstack.apt-get.eu/systemvm/4.4/\
systemvm64template-4.4.0-6-kvm.qcow2.bz2 \
-h kvm -F
```

システム VM テンプレートの取得と展開が終了したら，NFS 領域のアンマウントを行う．これで CloudStack のインストール作業は終了である．

```
umount /mnt/primary
umount /mnt/secondary
rmdir /mnt/primary
rmdir /mnt/secondary
```

A.7 CloudStack のセットアップ

管理サーバの URL にアクセスすると，図 A.7.1 に示すようなログイン画面が表示される．ユーザ名に admin，パスワードに password と入力すると初回ログインできる．この際，ドメインは空のままで良い．初回ログインに成功すると，クラウドのセットアップウィザードが表示される．ウィザードに沿って設定を行えば，クラウドのセットアップが完了し，図 A.1.1 のようなダッシュボード画面が表示される．この画面を基点として，VM，ユーザ，ネットワーク，テンプレートなどクラウドを構成する様々なモジュールや機能の管理を行うことができる．VM の作成時には図 A.7.2 のような画面が表示され，クリック操作のみで VM の作成が可能である．

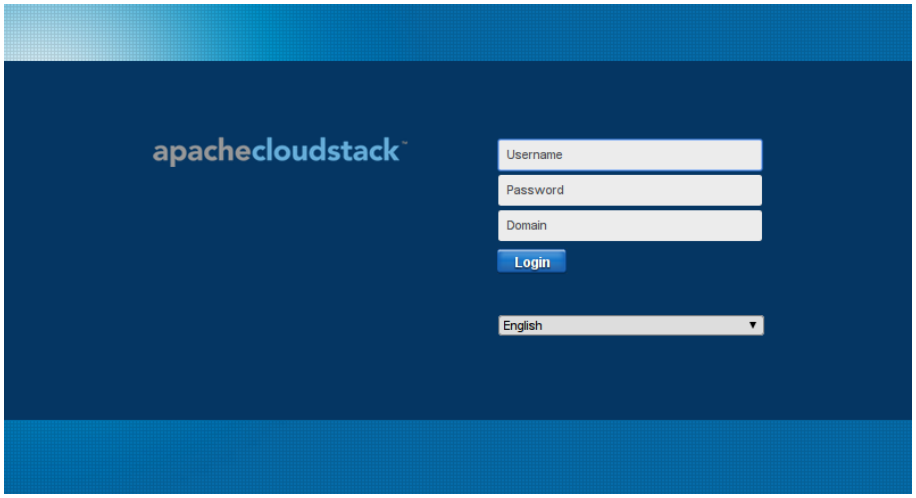

図 A.7.1: CloudStack のログイン画面

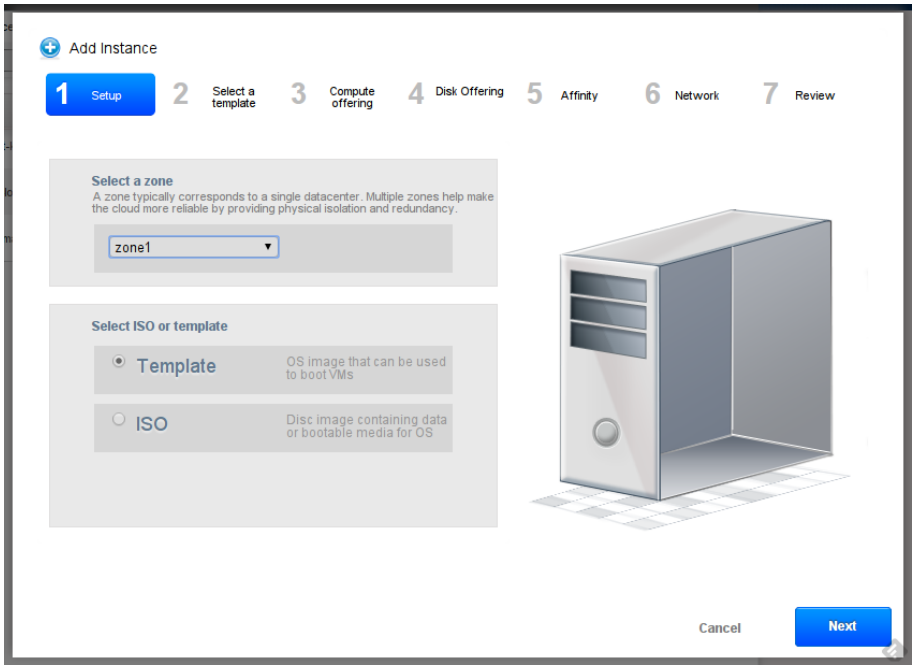

図 A.7.2: VM の作成ウィザード画面

発表論文

[1] 中島 拓真, 吉見 真聡, 入江 英嗣, 吉永 努. “クラウド環境における透過的データアクセスと計算リソースの動的共有手法,” 信学技報, Vol. 113, No. 282, pp.91-96, Nov. 2013.

[2] Takuma Nakajima, Masato Yoshimi, Hidetsugu Irie, Tsutomu Yoshinaga. “Sharing Computing Resources with Virtual Machines by Transparent Data Access.” 1st International Workshop on Computer Systems and Architectures (CSA), pp.359-365, December, 2013.