Kawaii を探して
KAWAII Quest -ルンバの大冒険-

MA17099 古橋健斗

1. 研究の背景と目的

阪神・淡路大震災をきっかけに，ロボット機器による災害救助ロボットの研究が活発に行われるようになり，レスキューロボットの開発が本格化した．この問題に関しては，災害発生時には倒壊家具や瓦礫により被災者の発見が困難になり，また都市部被災では燃料などの可燃物の漏出や漏電などにより火災などの二次災害が発生し，救援活動を行なう側が被災者となる危険性を含んでいる．被災者の発見は災害救助犬などが存在するが，これには育成に時間とコストがかかり，勘と経験といった要素が含まれるため，一般的な問題となっている．そこで，これらを機械化することで災害による被害者を減らそうと言う思想がレスキューロボットの開発に繋がっている．ロボット技術を瓦礫の除去などに使用する試みがあったが，救助活動を行う人間の救助チームだけではカバーすることができない．したがって，人間探索を行うロボット技術の開発に関する研究開発が進められている．また，福島第一原発事故の県から，危険箇所での作業者の被曝リスクを減らすために原子力災害ロボットの導入や運用の重要性が再認識された．そのため，近年はロボット技術による探索技術が発展している．

一方，ディープラーニングは近年飛躍的な成長を遂げてきている．ディープラーニングとは，多層構造のニューラルネットワークを用いた機械学習である．画像などのデータを入力し，情報がより深くの層に伝達している間に各層で学習を繰り返す．この過程で，使用者が設定した特徴量が自動で計算される．特徴量とは，特定の概念を特徴づける変数のことである．この特徴量を発見できれば，パターン認識精度の向上や，問題の解決により近く．この階層的な特徴量の学習が，ディープラーニングが従来の機械学習と決定的に異なる．判別機の学習には，正解画像が与えられれば 1 を，不正解画像が与えられれば 0 を出力するように学習する．大量のデータを判別機の学習に与えることで，より正確な判定ができるようになる．

2. 提案

これらをふまえて, 我々はディープラーニングを用いて対象物を判別し, 効率的な情報収集を行うことができるロボットシステムの提案をする．日本の誇れる文化であるゲーム・漫画やアニメーションなどのデジタルコンテンツが大きな輸出超過になっている [1]．その主要な要因として，高度な技術力とキャラクタのかわいさが挙げられることから，かわいいを人工物の感性価値として選び，かわいい人工物を対象として選択することにした．しかし，本研究はかわいい人工物を対象物とするが，ディープラーニングの教師データを変えることで，様々な対象にも応用できるようロボットシステムの構築も目的とする．

本システムではロボットとしてルンバ [2] を採用する．ルンバは，iRobot が製造・販売するロボット掃除機であり，部屋の中を単純なアルゴリズムに従って移動し，掃除を行う．本システムでは，ルンバを掃除としてではなく，探索するロボットとして使用する．そこで，本システムを KAWAII QUEST ルンバの大冒険と称し，以下にシステムの概要を述べる．

2.1 システムの概要

本システムではルンバに学習モードと探索モードの 2 つのモードを搭載する．

1. 学習モード
   ディープラーニングには大量の教師データ (画像データ) が必要となる．教師データを多く準備することが精度の向上に直接繋がる．また，個人によってかわいいと感じるものが異なる．そこで，本システム利用者がルンバを操作し，ルンバに搭載しているカメラを用いて教師データを収集する．これにより後述する探索モードで，本システム利用者毎に異なる分類器を作成することが可能になる．

2. 探索モード
   本モードでは，ルンバが掃除を行うときに使用する単純な動作で部屋を徘徊する．そのときに，カメラに映った画像を解析し，物体を検知したときに作成した分類器で画像にかわいいものが映っているかを判別する．かわいいものが映っていれば，それを本システム利用者に表示し，かわいいかどうかを選択してもらう．かわいいと選択されるとその画像が保存され，正解教師データとして分類器を調節し，かわいくないと選択されると，その画像を不正解教師データとして分類器を調節する．

以上の 2 つのモードを繰り返し使用することで，ユーザにあった判別器が作成され，判別の精度が向上する．また，このモードを実現するために，以下の機器を使用する．

1. ルンバ
   本システムの要である探索を行う．ルンバは，地図を作成せずに「らせん状に掃除する」「壁伝いに掃除する」「何かにぶつかったら角度を変えてランダムウォークする」などの単純なアルゴリズムで移動する．また，地図を作成することで同じ場所を探索しなくなり，より効率的な探索が可能になる．

2. Kinect
   Kinect[3] はマイクロソフトが発売したジェスチャー・音声認識によって操作ができるデバイスである．本システムでは Kinect をルンバに接続し，ルンバと連動して画像を撮影するために使用する．Kinect と Robot Operating System(ROS)[4] を連動させることで，Kinect に映っている画像から物体検知を行うことが可能になる．

3. ノート PC(ルンバに搭載)
   この PC で ROS を動作させ，ルンバと Kinect の連動を行う．他にも，撮影した画像をサーバに転送を行う．

4. ノート PC(本システム利用者)
   この PC を使用し，対象物の選択や画像の閲覧を行う．また，学習モードではルンバの操作や画像の撮

表 1: RaspberryPi2 ModelB の性能

CPU	ARM Cortex-A7 クアッドコア 900MHz
メモリ	1GB
ネットワーク	10/100Mbps イーサネット
電源	900mA(4.5 ~5.5W)

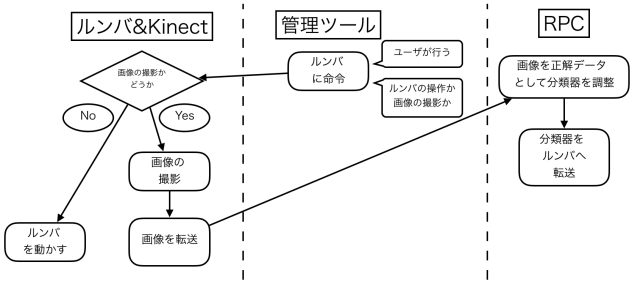

図 1: 学習モードのフロー

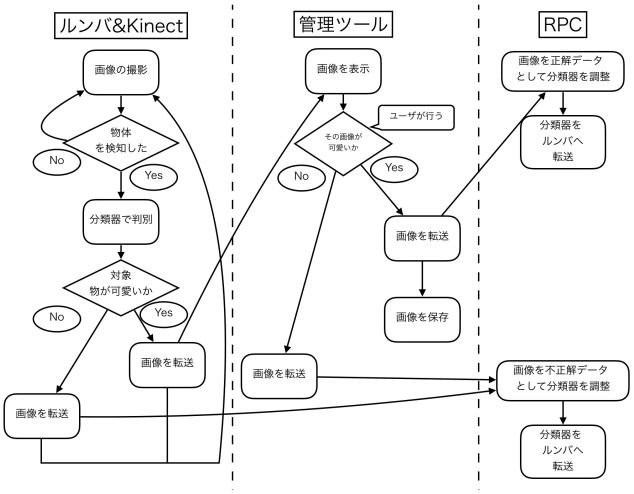

図 2: 探索モードのフロー

影が行える．これらを管理ツールと称し，ブラウザで管理できるような環境を構築する．

5. RaspberryPi
分類器の作成で画像処理を行うためには，高性能の PC が必要であるが，そのような PC を用意するのは非常にコストがかかってしまう．そこで，本システムでは安価である RaspberryPi[5] を大量に使用し，効率よく画像処理を行える環境を構築する．

次に，前述した画像処理を行うための大量の RaspberryPi を使用した環境について述べる．本システムでは，表 1 に示す RaspberryPI2 ModelB を利用し，Linux 系専用 OS である Raspbian Jessie Lite[6] を実行する．表 1 の通り，通常の PC に比べて性能は劣るが，一台約 5000 円と安価であるため少ない費用で多くの数用意することができる．この環境で分散的に処理が行うことで，低コストで高性能なクラスターを作成することができる．RaspberryPi を用いたデータセンタの研究 [7] がすでに行われているため，その有意性は存在する．この環境を RaspberryPi-Cluster(RPC) と称し，詳細を後述する．

次に，本システムの作業フローについて説明する．まず，学習モードの作業フローを図 1 に示す．学習モードでは，本システムの利用者がルンバに対して命令を出す．本システムの利用者はルンバが搭載している Kinect が映している映像を見ながら操作を行う．その命令は前後の移動，左右の旋回，画像の撮影である．命令が前後の移動，左右の旋回だった場合，ルンバは命令に従ってその動作を行う．命令が画像の撮影であった場合，ルンバは現在映っている画像を RPC に転送し，その画像を正解データとして分類器の調整を行う．次に，探索モードの作業フローを図 2 に示す．ルンバは搭載されている単純なアルゴリズムで対象物を探索する．そして，物体が検出されるとその画像を保存し，分類器で判別を行う．かわいいと判別された場合はその画像を管理ツールで表示し，利用者がその画像の判断を行う．利用者がかわいいと判断した場合は，その画像を RPC に転送し，正解データとして分類器の調整を行い，かわいくないと判断した場合は，その画像を RPC に転送し，不正解データとして分類器の調整を行う．分類器でかわいくないと判別された場合はその画像を RPC に転送し，不正解データとして分類器の調節を行う．

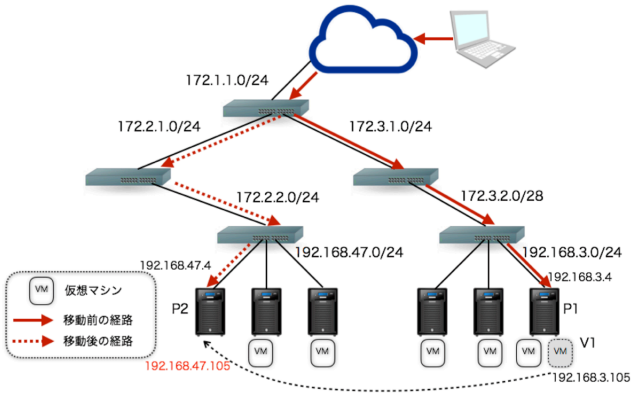

図 3: 想定するクラウド環境

本稿では，RaspberryPi による RPC の構築について説明する．

3. アプローチ

本節では RPC の構築のアプローチを説明する．この環境を構築するには RaspberryPi を複数台接続し，分散的に処理を行う必要がある．そこで，RPC では RaspberryPi で仮想マシンを起動し，仮想マシンを複製や移動によって柔軟な対応を取り，システムの性能や効率の向上を図る．この手法は Amazon Web Service (AWS)[8] や Microsoft Azure[9] のようなクラウドサービスで用いられているため，まずはクラウド環境について説明する．

3.1 クラウド環境

本システムで想定するクラウド環境を図 3 に示す．クラウド環境では，一般に複数のネットワークスイッチとそれらに接続した物理マシンで構成される．そして，サービス提供者からリソース取得の要求があると，インフラ提供者は物理マシンで仮想マシンを起動してそれらを提供している．クライアントのリクエストはスイッチで適切に転送され，対象となる仮想マシンが適切に処理してサービスを運用している．ここで，物理マシンが高負荷になり，仮想マシンが移動する状況を想定する (図 3)．図 3 では，物理マシン P1 で仮想マシン V1 が動作している状況において，V1 を物理マシン P2 に移動することを想定している．P1 と V1 にはそれぞれ IP アドレス (192.168.3.4, 192.168.3.105) が設定されており，ユーザの立場からは区

別できない．一方，P2 にも 192.168.47.4 の IP アドレスが設定されている．仮想マシンの移動では，一般に IP アドレスは変化しないため，V1 の IP アドレスは P2 に移動した後も変わらず 192.168.3.105 となる．V1 の移動は P2 が直接接続しているネットワークスイッチ，および上流に位置するスイッチに影響しないため，従来 V1 に届けられていたパケットは依然として 192.168.3.0/24 を管理するスイッチに届けられることになり，移動後の V1 に届けられることはない．移動後の V1 に正しくパケットを届けるためには，V1 の IP アドレスと移動後のネットワークに合致したものに変更 (e.g., 192.168.47.105) する必要がある．しかしながら，ユーザがこの変更を知る手段がないため，V1 へのリクエストは依然として 192.168.3.105 宛に送信される．したがって，リクエストを正しく移動後の V1 に届けるためには，ネットワークスイッチに変更を加え，宛先 192.168.3.105 へのパケットは変更後のアドレス (192.168.47.105) に書き換える必要がある．

このように，クラウド環境で円滑にサービスを運用するためには，仮想マシンの移動や複製に伴い，関連するネットワークスイッチも連動して変更する必要がある．

3.2 RPC の要件

3.1 節で述べたように，クラウド環境では仮想マシンの移動だけでなく，IP アドレスの変更など，移動に伴う変更や，移動の契機となる状況の把握，多数のマシンを制御する必要がある．このことを踏まえて，本システムでは RPC を実現するため，以下の項目を必要要件と考える．

負荷状況の把握 仮想マシンの移動や複製は，一般に高負荷状態にある物理マシンの負荷を分散し，仮想マシンで提供するサービスの性能維持や，低負荷状態の物理マシンに散財する仮想マシンを集約し，リソースの効率的な使用のために行われる．本環境でもこれらの負荷状態を測定するため，負荷状態の判定に一般的に用いられる CPU とメモリ使用率を使用する．

IP アドレスとルーティング管理 仮想マシンが移動する時，移動先物理マシンが同一ネットワークに所属している場合には IP アドレスの変更は必要ない．しかし，3.1 節でも述べたように，異なるネットワークに所属する物理マシンに移動する場合には移動後に IP アドレスの変更が必要になる．同一ネットワーク内だけの移動/複製は，本システムの効率的な運用の妨げになるため，本システムではネットワーク間を跨いだ仮想マシンの移動/複製も想定する．その場合，複数の物理/仮想マシンに同一の IP アドレスを指定することはできないため，仮想マシン移動時には IP アドレスを開放し，移動後には未使用の IP アドレスを設定する必要がある．さらに，ネットワーク間を跨いだ仮想マシンの移動/複製には，3.1 節で述べたように，IP アドレスの変更に伴い，関連するネットワークスイッチを変更し，適切にクライアントからのリクエストを 転送する必要がある．

複数マシンの管理 本システムでは，多数の物理マシン，ネットワークスイッチで構成することになり，それぞれの状態管理 (e.g, 負荷の測定) や制御 (e.g, 設定変更や仮想マシンの移動/複製) を個別に行うことは現実的に難しい．そこで，本環境を構成する物理マシンやネットワークスイッチを一元管理し，設定変更や制御を自動で行う必要がある．

表 2: QEMU の機能

qemu-img	仮想マシンのイメージファイルの作成
qemu-system-arm	仮想マシンの起動
migrate	仮想マシンの移動

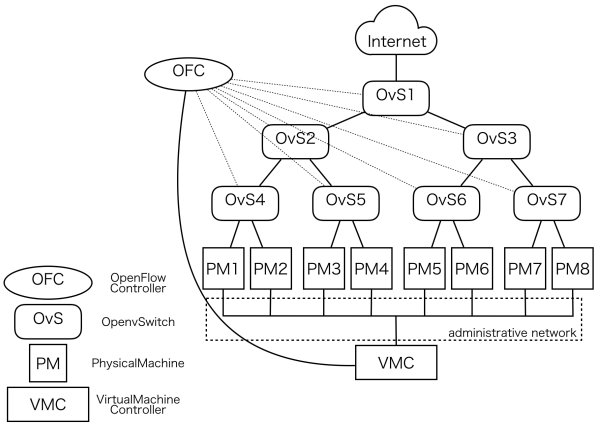

図 4: テスト環境のトポロジ

4. 設計

本節では，RaspberryPi で仮想環境を実現する方法について述べ，それに際して必要となるネットワークを構築する OpenFlow について述べる．最後に，本システムの分散システムとしての要件をまとめる．

4.1 RaspberryPI での仮想化環境

本環境では 2.1 節で述べたように，RaspberryPI2 ModelB を利用する．CPU である ARM Cortex-A7 は仮想化拡張機能を備えているため，Linux カーネルが備えるハイパーバイザ，KVM を実行することができる．一般の PC では，KVM はパッケージ管理ソフトウェア (e.g., apt-get) などを利用してインストールできるが，RaspberryPi ではパッケージでは提供されていない．そのため，カーネルの組み込み機能として実行する．また，KVM での仮想マシンの管理には virt[10] を利用することが一般的であるが，RaspberryPi では virt を利用することができない[1]．そのため，本システムでは KVM と連携して動作する qemu[11] が提供する機能 (表 2) を利用する．

一方，3.2 節でも述べたように，本システムの実現には仮想マシンの移動/複製に同期したネットワークの変更が必要になる．近年，仮想化技術の進歩に伴い，ネットワークを柔軟に制御するために Software Defined Network (SDN)[12] が注目を集めており，利用されている．そこで本システムでも SDN を利用してネットワークを制御する．本研究では SDN の一種であり，広く利用されている OpenFlow[13] を利用するため，OpenFlow 対応のソフトウェアスイッチである Open vSwitch(OvS) を RaspberryPi にインストールして利用する．そのため，本環境はすべて RaspberryPi だけで構成できる．

4.2 OpenFlow の概要

SDN では，経路制御を行うコントロールプレーンとデータの転送を行うデータプレーンが共存する既存のネットワーク機器とは異なり，コントロールプレーンとデータプレーンを分離したアーキテクチャを採用している．OpenFlow は，データプレーンを OpenFlow スイッチと呼び，OpenFlow スイッチを制御するコントロールプレー

[1] インストールは可能であるが，実行できない

ンを OpenFlowController(OFC) と呼ぶ．OpenFlow スイッチではデータの転送のみを行い，経路制御は OFC によって行われる．OpenFlow スイッチは，パケットが内包する情報 (送信元/送信先 IP アドレスなど) を条件に対象となるパケットを抽出し，それらに対して適切なアクション (パケット転送など) を実行する．この，条件とアクションの組み合わせをフローと呼び，それらをまとめたテーブルをフローテーブルと呼ぶ．OpenFlow スイッチはフローテーブルを保持し，自身がもつフローに従ってパケットを処理する．そして，条件に合致しないパケットを OpenFlow が受信した場合，OpenFlow スイッチは OFC に問い合わせ (パケットイン)，適切なフローを受信してパケットを処理すると伴に，フローテーブルに記憶する．

4.3 分散システムの要件

本稿で説明した RPC において，分散システムであることはシステムの効率を上げるために必要である．RPC を実装するにあたって，必要となる分散システムの要件を以下に記述する．

まずは透過性について説明する．

位置透過性 本システムでは，OpenFlow によるパケットの振り分けを行うため，ルンバは 1 つの仮想マシンの IP アドレスを知っていれば，各仮想マシンにパケットが振り分けられ処理を行うことができる．

移動透過性 仮想マシンの起動，複製，移動が行なわれるが，OpenFlow がパケットを書き換え，データの転送を行うので，ルンバは仮想マシンの状況を知る必要がない．

複製透過性 仮想マシンが (データごと) 複製されるが，ルンバはそのことを考慮する必　　要がない．

並行透過性 リソース (機械学習の教師データなど) を複数の仮想マシンやユーザ間で共有することが可能である．

障害透過性 物理マシンの障害が発生しても，OpenFlow が検知し，その物理マシンを避けてデータの転送を行うので，ユーザには障害が発生していないように見える．

次にスケーラビリティについて述べる．

サービス 複数仮想マシンでデータの処理を行うので，ユーザが増加してもシステムの運営には影響を与えない．

データ 新たに RaspberryPi を接続することで，物理的にリソースを増やすことが可能である．

アルゴリズム OpenFlow で負荷状況に応じて負荷を分散することが可能である．

5. 実装

本節では，3.2 節で挙げた要件に対応する本システムでの実現方法を述べる．

5.1 テスト環境のアーキテクチャ

3.2 節で述べたように，多数の物理マシンや仮想マシンの状態管理，および制御を個別に行うことは困難である．そこで，本システムでは，仮想マシンや仮想マシンを起動する物理マシンを一元管理する仮想マシンコントローラ (VMC) を実現する．また，KVM を用いた仮想マシンの移動には NFS を利用する必要があり，一般に NFS の利用はローカルネットワークであることが望ましい．そこで本システムでは，図 4 に示すように，ユーザがサービス利用のために利用するネットワークとは異なる管理ネットワークを設け，すべての物理マシンを管理ネットワークにも接続する．そして，この管理ネットワークに VMC を接続することで，後述する負荷状況や IP アドレス，およびルーティングの管理を容易にする．

表 3: 統計情報 [13]

カウンター	Bits
フローテーブルごと	
有効エントリー数	32
パケットルックアップ数	64
パケットマッチ数	64
フローごと	
受信パケット数	64
受信バイト数	64
フローが作られてからの経過時間 (秒)	32
フローが作られてからの経過時間 (ナノ秒)	32
ポートごと	
受信パケット数	64
転送パケット数	64
受信バイト数	64
転送バイト数	64
受信ドロップ数	64
転送ドロップ数	64
受信エラー数	64
受信フレームアライメントエラー数	64
受信オーバーランエラー数	64
受信 CRC エラー数	64
コリジョン数	64
キューごと	
転送パケット数	64
転送バイト数	64
転送オーバーランエラー数	64

5.2 負荷状況の把握

負荷状況の指標となる各物理マシンの CPU とメモリ使用率，各スイッチのトラフィック量を取得する．CPU とメモリ使用率の取得には，各物理マシンの/proc ディレクトリ以下に格納される情報 (meminfo) を利用し，一定間隔 (現在は 10 秒) で継続的に取得する．また，トラフィック量の取得には，表 3 で示す OvS が保持する統計情報を利用し，各 OvS 単位，および個々の OvS のポート単位でのトラフィック量を計測する．これらの情報を OFC に集約し，VMC と連携する．

5.3 IP アドレスとルーティング管理

3.1 節で述べたように，クラウド環境の実現には仮想マシンの移動や複製だけでなく，関連するネットワークスイッチも連動して変更する必要がある．そのため，本システムでは仮想マシンに使用する IP アドレスは DHCP

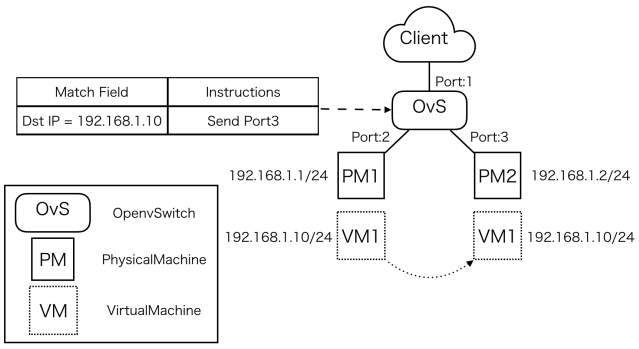

図 5: 同一ネットワークへの仮想マシンの移動

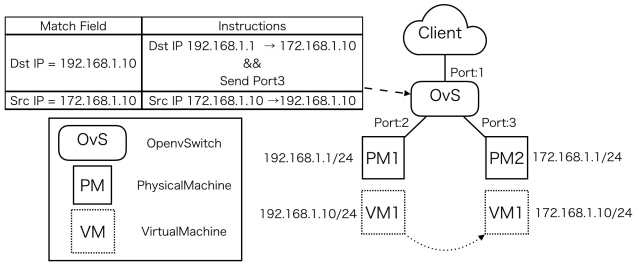

図 6: 外部ネットワークへの仮想マシンの移動

を用いることなく VMC が管理し，仮想マシンの移動に同期した変更を可能にする．本システムでは，仮想マシンに付与する IP アドレスのネットワークアドレスは，物理マシン所属するネットワークアドレスと同一である必要がある．そのため，移動や複製に伴い変更の必要がある．本システムでは，使用中の IP アドレス，ネットワークアドレスを VMC で集中管理しており，仮想マシンの移動や複製先のネットワークに適した IP アドレスを付与する．仮想マシンへの具体的な設定は，ifconfig や route コマンドを使用したスクリプトを用意している．

次に，3.1 節で述べたように，クライアントのリクエストを正しく移動，または複製後の仮想マシンに届けるためには，リクエストを経由するスイッチを適切に変更する必要がある．VMC は仮想マシンの移動または複製を行った後，移動・複製前，および移動・複製後の IP アドレス，実行の種類 (移動または複製) を OFC に通知する．OFC は関連する OvS を特定し，適切に設定変更する必要がある．以降，同一ネットワーク，およびネットワークを跨いで仮想マシンが移動または複製されるときの設定変更について述べ，最後に関連する OvS の特定について述べる．

1. 同一ネットワークへの仮想マシンの移動
   同一ネットワーク内の仮想マシンの移動では IP アドレスの変更は行われない．したがって，図 5 に示すように，特定した OvS において，転送先アドレスが VM1(192.168.1.10) にマッチしたパケットの送出ポート番号を変更する (Port2 → Port3)．

2. 外部ネットワークへの仮想マシンの移動
   外部ネットワークへの仮想マシンの移動では，IP アドレスの変更が伴う．図 6 では，異なるネットワークに所属する PM1 と PM2 間で仮想マシンが移動する．そこで，OvS では転送先アドレス 192.168.1.10 にマッチしたパケットの送出ポートを

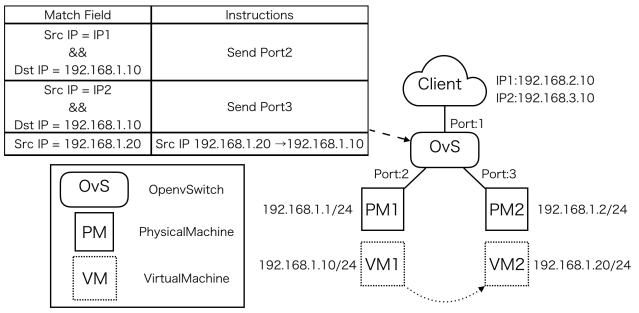

図 7: 仮想マシンの複製

変更し (Port2 → Port3)，かつパケットの DST アドレスを変更する (192.168.1.10 → 172.168.1.10)．さらに，172.168.1.10 からの送信元に戻されるパケットについては，OvS で SRC アドレスを 192.168.1.10 に書き換える．その結果，クライアントが仮想マシンの移動に気づくことはない．

3. 仮想マシンの複製
   仮想マシンが複製される場合，同一ネットワークか否かに関わらず，複製した仮想マシンには新たな IP アドレスが割り当てられ，仮想マシン間でリクエストを分散して処理する．このとき，複製前に既にリクエストを処理したクライアントに対しては複製前の仮想マシンがリクエストを処理し，新たにリクエストを送信したクライアントに関してはラウンドロビンで処理を分散する．

   図 7 は 2 台のクライアント (IP1，IP2) のうち IP1 が VM1(192.168.1.10) に既にアクセスしており，VM2(192.168.1.20) を生成後，IP2 が VM1 に対してリクエストを送信したときの様子を示している．IP1 から VM1 へのリクエストは既に実行されているため，IP1 から VM1 へのリクエストは OvS のフローテーブルに従って引き続き Port2 に送出される．一方，IP2 から VM1 へのリクエストは OvS のフローテーブルにマッチしないため，OvS は OFC に処理の問い合わせを行う (パケットイン)．この時，OvS は VMC によって複製された VM2 の IP アドレスを認識しているため，パケットの DST アドレスを変更する (192.168.1.10 → 192.168.1.20) して Port3 に送出する．そして，学部ネットワークへ仮想マシンの移動と同様，戻りパケットの SRC アドレスを VM1 のものに書き換える (192.168.1.10)．その結果，処理の分散はクライアントには透過であり，既存のクライアント (IP1) のセッションも維持される．

5.4 複数マシンの管理

5.1 で述べた通り，VMC を使用することで，多数の物理マシンや仮想マシンの状態管理や制御を個別に行うことが可能になる．本システムの利用者はそのサーバにアクセスすることで本環境の設定を行うことができる．各物理マシン，仮想マシンの ID やリソース状況は hostname/index で確認することができる．この情報を元に，リソースの閾値や仮想マシンの管理を行い，システムを自動で効率よく運用できるとともに，システム利用者が予測していなかった状態でも管理することができる．

6. 評価・考察

本節では，RaspberryPi 17 台 (OvS に 7 台，PM に 8 台，OFC と VMC にそれぞれ 1 台) を使用して図 4 に示

表 4: サーバを通信する時の経由する OvS の数

route	通信間	経由する OvS 数
1	PM1 ⟷ PM2	1
2	Internet ⟷ PM1	3
3	PM1 ⟷ PM3	3
4	PM1 ⟷ PM8	5

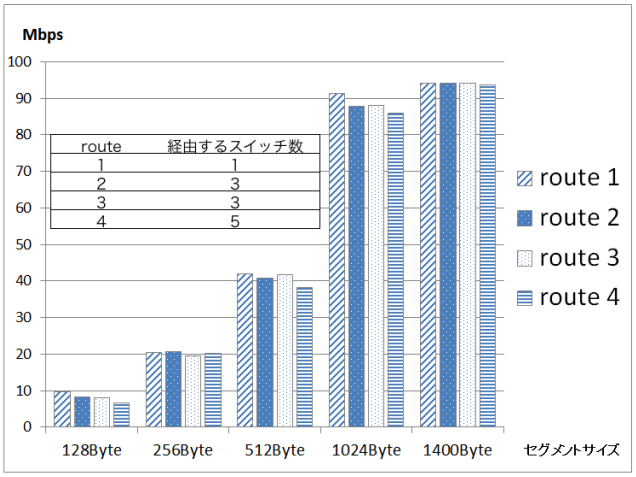

図 8: ネットワーク速度

すトポロジを構成し，本システムを評価する．

まず，RaspberryPi での OpenFlow 環境が十分な性能を有することを示すため，複数の OvS を経由するときのトラフィック量を計測し，スループットを示す．次に，仮想マシンの移動，および複製時の実行時間を測定して本テスト環境の基本性能を示す．次に，仮想マシンの移動，複製時，時間経過に伴うの CPU とメモリ使用率を測定し，考察する．最後に，RaspberryPi で分類器の作成を行なった時にどのようなリソース推移になるかの結果を提示し，考察する．

6.1 RaspberryPi での OpenFlow 環境

RaspberryPi を用いた OvS のスループットを測定するため，図 4 に示すトポロジにおいて，表 4 に示す経路でネットワークに負荷をかけ，スループットを計測する．この計測には iperf[14] を利用し，最大 TCP セグメントを，128/256/512/1024/1400[Byte] としたときの値を測定した．まず，設定した最大 TCP セグメントに依存せず，route1 と route4 を比較すると分かるように，経由する OvS の数によって僅かに転送速度が低下する．これは，パケット転送に関連する OvS の個数の影響であり，最大でも 5Mbps 程度の性能低下であることから (最大 TCP セグメントサイズ 1024[Byte] の場合)，極端な速度低下は見られない．また，最大 TCP セグメントサイズを 2 倍にするごとににスループットも倍増していることから，パケット転送に関しては十分な性能が得られていることがわかる．なお，最大 TCP セグメントサイズが 1400[Byte] の場合，1024[Byte] の場合と比較して大幅なスループットの向上は見られなれなかった．これは，RaspberryPi のイーサネットポートが 10/100Mbps 対応であり，100Mbps 以上のスループットを得られないためである．一方，各 OvS では表 3 で示した情報を取得できるため，関連する OvS のトラフィック量を計測した．その結果，計測したトラフィック量の合計が iperf で転送したトラフィック量と一致することを確認した．

これらの結果から，RaspberryPi での OpenFlow 環境は十分な性能が得られると判断することができる．

表 5: 仮想マシンの操作に要する時間

仮想マシンの操作	起動 (秒)	移動 (秒)	複製 (秒)
実行時間	47	32	49
ネットワーク構成変更		3	4

表 6: 表 5 の実行時間内訳

仮想マシンの操作	起動 (秒)	移動 (秒)	複製 (秒)
Qemu の機能	32	23	32
スナップショットの転送	-	2	2
ネットワークの設定	8	-	8
リソースの取得	7	7	7
合計	47	32	49

6.2 仮想マシンの移動と複製

本テスト環境では，仮想マシンの起動に qemu のスナップショット機能 [15] を用いている．また，この機能は仮想マシンの複製時にも利用する．そのため，仮想マシンの移動と複製に加え，スナップショットを用いた起動時間を表 5 に示し，その内訳を表 6 に示す．

表 6 に示すように，仮想マシンの起動は，a) スナップショットファイルを指定した qemu-system-arm コマンドの実行，c)ifconfig や route コマンドを利用した，ネットワーク設定用スクリプトの実行 (ネットワークの設定)，d) 仮想マシンのリソース使用量を測定するスクリプトの実行 (リソースの取得)，で構成される．表 6 より，仮想マシンの起動には a) に 32 秒，c) に 8 秒，d) に 7 秒要し，起動して使用可能になるまでに約 47 秒かかった．一方，仮想マシンの移動には，a')migrate コマンドの実行，b) スナップショットの転送，d) リソースの取得，で構成され，a') に 23 秒，b) に 2 秒，d) に 7 秒要し，移動全体で 32 秒要した．なお，この実験では仮想マシンは同一ネットワーク内を移動したため，c) ネットワークの設定は必要ない．

最後に，仮想マシンの転送は，仮想マシンの起動に加えスナップショットの転送が加わるため，a), c), d) の実行に加え，b) スナップショットの転送に 2 秒要し，全体で 49 秒要した．

これら 3 つの結果から，起動や移動で用いるコマンドの違いによって a) と a') では必要な時間が異なるが，b), c), d) に要する時間はどの場合でも同じ値を示した．これらのことから，本テスト環境で実現した機能 (b), c), d)) は，安定して動作すると考えることができる[2]．

一方，5.3 節で述べたように，仮想マシンの移動と複製には，VMC がそれらの完了を検知した後，OFC に依頼して関連する OvS を変更する，というネットワーク構成変更を伴う．したがって，表 5 に示すように，仮想マシン移動時にはネットワーク構成変更に伴いクライアント

[2]VMC や OFC と各 PM 間の通信，およびスナップショット転送は管理ネットワークを使用するため，安定したスループットを得られる．

のリクエストが 3 秒間遮断されたが，複製では遮断されることなく通信することができた．これは，仮想マシン移動時は構成変更が終わるまでパケットを受け取る仮想マシンが存在しなくなるのに対し，仮想マシンの複製では複製後の仮想マシンに関するネットワークの変更が行われるまでは複製元の仮想マシンにリクエストが届けられていることを意味している．

この結果から，本テスト環境で提供するネットワーク構成変更は正しく機能していると推察される．

6.3 CPU とメモリ使用率の変化

仮想マシンの起動，移動，および複製することによる物理マシンのリソース使用率の変化を確認するため，物理マシンで仮想マシンを起動した後，2 台の物理マシン間で仮想マシンの移動，および複製するときのリソース使用率の変化を確認する．また，仮想マシンの移動では，ネットワーク構成変更とリソース使用率の変化も確認するため，仮想マシンで Apache を起動し，Internet からその Apache に HTTP リクエストを送り続けながら仮想マシンを移動した．6.2 節で述べたように，移動や複製には管理ネットワークを使用するため，移動や複製がネットワークを跨ぐ場合でも結果に影響しない．そこで，本実験では図 4 の PM1 から PM2 に仮想マシンを移動，および複製した．仮想マシンを移動，および複製したときのリソース使用率の変化を図 9, 図 10 にそれぞれ示す．

まず，仮想マシンの移動では，PM1 で仮想マシンが実行され，約 40 秒かけて起動が完了する (図 9, 20 から 60 秒付近)．その間，CPU とメモリ使用率は上昇し，起動完了後に CPU 使用率は再び仮想マシン起動前と同等の値 (10%弱) を示している．この起動時間は表 6 とほぼ一致しており，リソース使用率の変化を正しく計測できていることを示唆している．その後，HTTP アクセスを開始後，仮想マシンの移動が完了するまで CPU 使用率は平均 20%弱を示しており，HTTP リクエストの処理や migrate コマンドの実行に割り当てられていることがわかる．その後，仮想マシンの移動が完了 (図 9，110 秒付近) すると CPU 使用率は再び仮想マシン起動前と同等の値となった．また，メモリ使用率も仮想マシン移動後は仮想マシン起動前と同等の値となった．

一方，仮想マシンの移動先である PM2 は，PM1 で migrate コマンドが実行された時刻付近からメモリ使用率が上昇し始め，移動が完了すると CPU 使用率が上昇した．これは，PM2 に仮想マシンが移動し，実行を開始したことを示唆している．なお，PM2 では仮想マシン移動完了後 CPU 使用率が急上昇した後，一度下降しているが，これはネットワーク構成変更によって HTTP リクエストが届かず，CPU はリクエストを処理していないことを示唆している．この振る舞いは表 5 の結果とほぼ一致しているため，リソース使用率の変化を正しく計測できていると考えることができる．

次に仮想マシンの複製による負荷分散では，移動と同様に，PM1 で仮想マシンが実行され，約 30 秒かけて起動が完了する．その後，スナップショットを転送する時に，CPU 使用率が上昇する (図 10, 65 秒付近)．スナップショットの転送が完了すると，PM2 で転送されたスナップショットを指定して仮想マシンの起動が行われる．複製完了後，PM1 へ HTTP アクセスを開始する (図 10, 破線 PM1)．通常はクライアント毎にアクセスを分散させるが，今回はリソースの変化を見るために，25 秒ごとにアクセスを分散させた．PM1 にアクセスが届いている間，PM1 では CPU 使用率が 10%程度上昇し，PM2 ではリソースに変化がなかった．次に，リクエストが PM2 へ転送が行われ

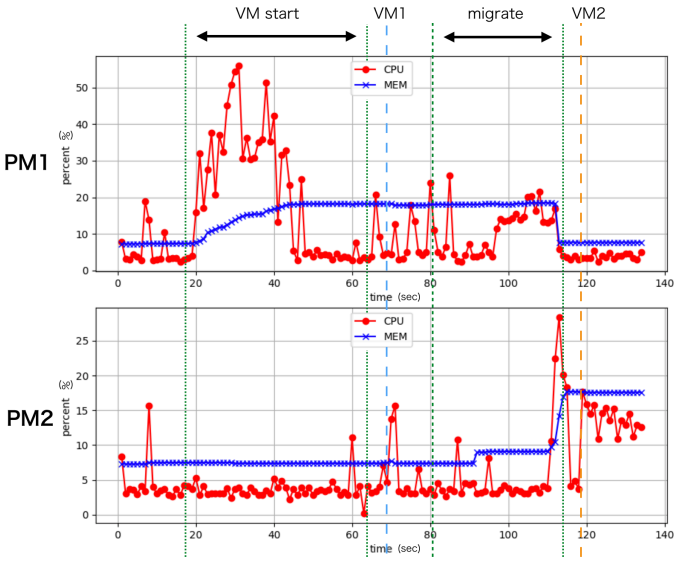

図 9: 仮想マシンの移動によるリソース推移

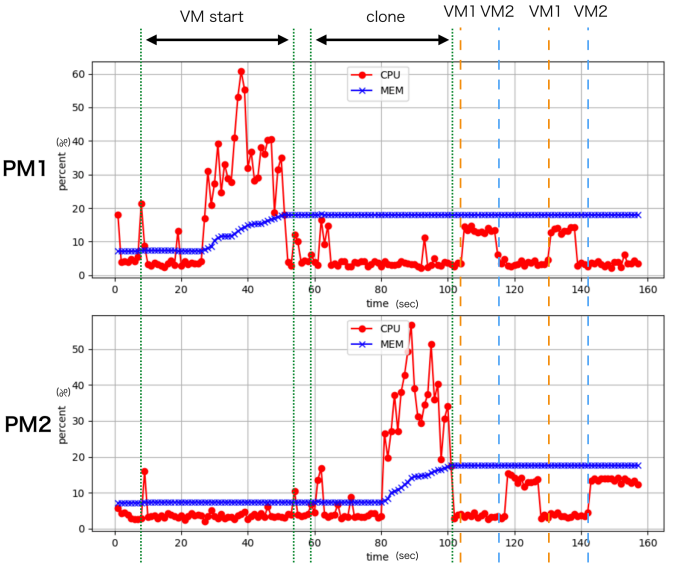

図 10: 負荷分散によるリソース推移

るようにネットワーク構成の変更を行うと，(図 10, 破線 PM2)，PM1 では CPU 使用率が下がり，PM2 では CPU 使用率が 10%程度上昇した．その後も PM1 へ転送が行われるようにネットワーク構成の変更を行うと，PM2 では CPU 使用率が下がり，PM1 では CPU 使用率が 10%程度上昇した．この振る舞いは表 5 の結果とほぼ一致しているため，リソース使用率の変化を正しく計測できていると考えることができる．また，負荷の分散が行われていることが確認できたので，本機能が想定通りに動作していると言える．

最後に一台の物理マシンで仮想マシンを複数起動した状態において，一台の仮想マシンの移動と複製を行なった場合のリソース使用率の変化を図 11, 12 にそれぞれ示す．これまでの結果同様，仮想マシンの実行が開始されると CPU，メモリ使用率が増加し，起動が完了すると CPU 使用率は実行前と同等の値となる．一方，メモリ使用率は起動している仮想マシンの個数に応じて増加している (一台あたり約 10%使用)．そして．仮想マシンの移動や

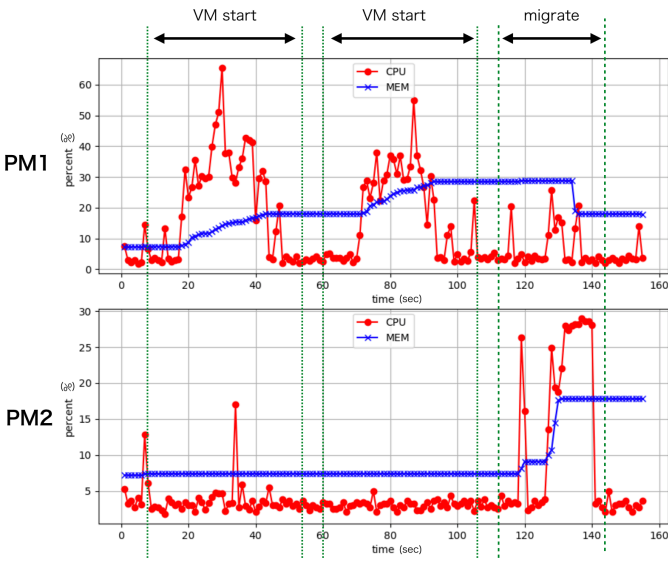

図 11: 仮想マシンを複数起動した状態でのリソース推移: 移動

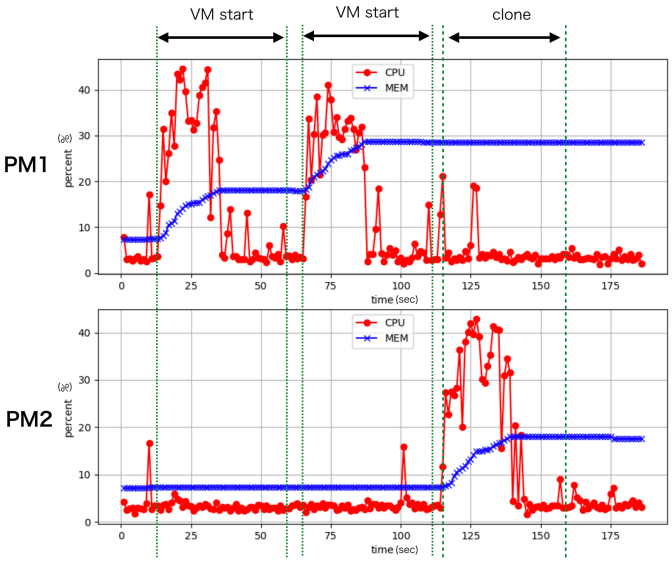

図 12: 仮想マシンを複数起動した状態でのリソース推移: 複製

複製が行われると，移動した仮想マシンの個数に応じて移動先物理マシン (PM2) のメモリ使用率は増加し，反対に移動元物理マシン (PM1) のメモリ使用率は減少する．

これらのことから，一台の物理マシンで複数の仮想マシンが実行される場合でも，本テスト環境では正しくリソース使用率を計測していると考えることができる．

6.4 分類器の作成

最後に，RaspberryPi で分類器の作成を行なった時のリソース推移を図 13 に示す．分類器の作成は OpenCV2[16] のカスケード分類器 [17] の作成を行なった．今回は正解データ 60 枚，不正解データ 30 枚で実験を行なった．その結果，CPU 使用率は 25 35%，メモリ使用率は 20%使用された．このことから，1 台の RaspberryPi で 2 つの分類器の作成を行えることがわかった．しかし，図 13 に示すように，1 時間の時間を要した．これは，分類器の作成を行なってから 1 時間後にその結果が反映されることになるので，RPC としては機能していないことがわかる．CPU, メモリの使用率が 100%になっていないことから，単純に RaspberryPi の性能によるものだと推測される．

7. まとめ

本研究では，ロボットの探索能力とディープラーニングの判別能力を活かしかわいいものを探索するシステムを開発した．これにより，対象物の情報を収集することに成功した．しかし，本稿で説明した RPC システムは本研究で成果を残すことができなかったため，今後より効率的なシステムの運営が行えるように改良する必要があることがわかった．

7.1 今後の課題

今後の課題として，分類器の作成に時間がかかってしまう問題は，分類器の作成を分散処理させ，高速で分類器の作成を行うように修正する必要がある．また，対象物を拡張し，様々な情報収集を行えるようにするなどが挙げられる．

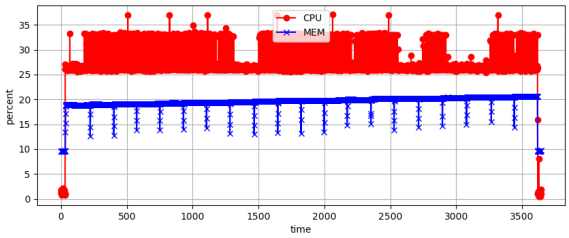

図 13: 分類器の作成を行なった時のリソース推移

参考文献

[1] 大倉典子, 青砥哲朗 ”かわいい人工物の系統的研究” 第 2 回横幹連合コンファレンス, 2007

[2] ロボット掃除機 ルンバ—アイロボット公式サイト-iRobot https://www.irobot-jp.com/roomba/ 2017/07/24

[3] Xbox One 用 Kinect—Xbox http://www.xbox.com/ja-JP/xbox-one/accessories/kinect 2017/07/24

[4] Ros Wiki http://wiki.ros.org/ 2017/07/24

[5] Raspberry Pi - Teach, Learn, and Make with Raspberry Pi https://www.raspberrypi.org/ 2017/07/24

[6] Raspbian Jessie Lite https://www.raspberrypi.org/downloads/raspbian/ 2017/07/14

[7] Fung Po Tso, David R. White, Simon Jouet, Jeremy Singer, Dimitrios P. Pezaros School of Computing Science, University of Glasgow, G12 8QQ UK, ”The Glasgow Raspberry Pi Cloud: A Scale Model for Cloud Computing Infrastructures”, 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops

[8] Amazon Web Services website. https://aws.amazon.com/jp/ 2017/04/19

[9] Azure website. https://azure.microsoft.com/ja-jp/ 2017/07/14

[10] virt-manager website. https://virt-manager.org/ 2017/04/19

[11] qemu website. http://www.qemu.org/ 2017/07/14

[12] Open Networking Foundation website. https://www.opennetworking.org/ 2017/01/11

[13] Open Networking Foundation OpenFlow website. https://www.opennetworking.org/sdn-resources/openflow 2017-01-11

[14] iperf website. https://iperf.fr/ 2017/04/19

[15] http://wiki.qemu.org/Documentation/CreateSnapshot 2017/07/20

[16] OpenCV2 http://opencv.jp/ 2017/07/25

[17] カスケード型分類器 — opencv 2.2 (r4295) documentation - OpenCV.jp http://opencv.jp/opencv-2.2/c/objdetect_cascade_classification.html 2017/07/25