Vol.2016-SE-192 No.5
Vol.2016-EMB-41 No.5
2016/6/2

Raspberry Pi を用いた小型クラウドシステム

早川 栄一[†1]　斎藤 直生[†2]

概要：ネットワーク帯域が狭い環境において、クラウド連携した組込みシステムの学習やシステムを構築するために Raspberry Pi を複数用いた可搬型のクラウド基盤システムを構築した。特徴は次のとおりである。(1) 省リソースで仮想環境を利用可能なようなコンテナ型のクラウドシステム、(2) 物理リソース状態を可視化可能なようにボードごとの LED を用いたリソース可視化機構、(3) 従来のクラウド管理ソフトウェアと比べて軽量小型のクラウド基盤システムの実現。

キーワード：クラウドコンピューティング, 組込みシステム, Raspberry Pi, 組込みシステム学習支援環境

Tiny Cloud System using Raspberry Pi

Eiichi HAYAKAWA[†1]　Naoki SAITO[†2]

Abstract: We developed a portable cloud infrastructure system using many Raspberry Pis for cloud oriented embedded systems learning and development in narrow network bandwidth environment. The features of the system are following: (1) container based, less resource consumption, (2) physical resource usage visualization using LED at each board and (3) system built on a lightweight and tiny cloud infrastructure system.

Keywords: Cloud computing, Embedded system, Raspberry Pi, Embedded system learning environment

1. はじめに

組込みシステムとネットワークとの連携が進んでいる．Internet of Things(IoT)に代表されるように，組込みシステムをネットワークに接続することによって，強力なサーバリソースの利用や，複数のデバイス連携，Web との協調作業などが可能になりつつある．このようなネットワーク連携の進展により，組込みシステム開発者に対してサーバサイドのプログラミングやシステム管理に関するスキルが必要とされてきている．

その一方で教育現場においてサーバリソースの利用は容易ではない．教材として用いられる LEGO や Raspberry Pi は可搬であり，利用する場所を選ばないが，クラウド利用に必須となる広帯域のインターネット接続のインフラは不十分なことがある．特に教室での利用のように数十台の同時利用を可能にする環境を作り，維持をしていくのは，コストが高くなる．また，クラウドサービスを提供するために OpenStack[1]や CloudStack[2]といったクラウド管理ツールがよく用いられるが，多くの PC に設置して管理するのは大変であり，また PC そのものが可搬ではない．さらに，クラウド管理ツールは設定が複雑であり，利用の手間がかかる．

また，このようなシステムでは Amazon Web Services や Microsoft Azure, Heroku といったクラウドサービスを利用することになるが，クラウドの設計上，物理的な資源は可視化されていない．一方組込みシステムからクラウドサービスを利用する場合は，物理的な CPU の割当てやメモリ，ネットワークの利用状況を意識する必要が出てくるが，この部分をモニタするのは難しい．

このような問題に対して，組込みシステムにおいてクラウド環境を利用する学習環境を対象として，小型・安価なボードコンピュータである Raspberry Pi 2（以下 RPi2）を用いたクラウドシステムの開発を研究の目的とする．近年では小型ボードの性能が向上し，サーバとして利用可能な性能を有していることから，このボードを複数台用いてポータブルなクラウドサーバ群を構築する．また，管理を容易にするためのクラウド管理ソフトウェアを開発し，サービスの容易な起動や終了を可能にする．また，物理リソースについては LED を用いて可視化できるようにして，利用者が資源の利用状況を容易に把握できるようにする．

2. 問題分析

本章では，学習環境において組込みシステムとクラウド

†1 拓殖大学 工学部 情報工学科
Takushoku University
†2 拓殖大学　工学部　情報工学科（現 株式会社ティ・オー・エス）
Takushoku University

とを連携させる場合の問題分析を行う．

2.1 利用環境の問題

第 1 章で述べたように，本システムでは教室や実験室，オープンスペースなどで組込みシステムと併せてクラウドシステムを利用することを想定している．このような環境では，次のような問題が生じる．

(1) **多くのユーザで利用可能なネットワーク環境の設営が難しい**

クラウド環境ではネットワークの利用が不可欠である．現在のネットワーク・インフラでは，単体のクライアントに対して，クラウド利用に必要となるネットワークリソースを用意することは難しくはない．その一方で，数十台規模の場合には，適切なネットワークを用意できる環境を準備することは未だ難しい問題である．セキュリティを確保しつつ, 適切なネットワーク環境を構築することは難しい．

(2) **物理的なマシン状態を把握しにくい**

組込みシステムと連携したクラウドシステムを利用する場合，物理的なマシンの状態を把握する必要がある．CPU やメモリ，ネットワークの負荷による応答性の違いが，組込みシステムのサービスにどのように影響を与えるかを分かりやすく表示できなければならない．マシン管理のためのモニタリングには多くのツールが存在するが[6,7]，複数台のマシンの状態を分かりやすく示すことは難しい．

2.2 クラウド環境の問題

クラウドでは，ユーザに提供されるリソースはすべて仮想化されたリソースであり，またその仮想リソースが実際に動作している物理マシンについての情報は，ユーザに対しては隠されているのが一般的である．この特徴はクラウドサービスを利用する際には有効だが，クラウドを運用する技術者としての知識を深めるには障害となってしまう．

RPi2 で既存のクラウドシステムを動作させる場合, 次に述べる問題があった．

(1) **KVM の ARM サポートが不十分**

クラウドシステムでは仮想化機構が重要な役割を果たす. Linux の仮想化機構としては KVM などを用いた完全仮想化や Xen などの準仮想化などが存在する. これらの機構は, PC で用いられている x86 系ではサポートが充実していて, 物理マシンに近い性能で利用することが可能である．

その一方で，RPi2 で用いられている ARM Cortex-A7 は仮想化支援機構は提供されているものの，Linux カーネルでの支援は不十分であり，KVM などでの利用には問題がある．ARM での KVM サポートは存在する[3]が，CloudStack や OpenStack からの利用は性能面からも難しかった．

(2) **インスタンス起動の際のオーバヘッドが大きく，インスタンスの生成ができない**

稼働するソフトウェアに対し RPi2 のマシン性能は実用的に十分ではないことから，実用上の問題が生じる可能性がある．本システム構築のための調査として，既存のクラウド環境である CloudStack や OpenStack を RPi2 で実行して確認したが, RPi2 ではインスタンス起動に多くの時間を要することが明らかになった. RPi2 ではメモリや入出力の性能が高くなく，カーネル起動のためのメモリイメージの処理に時間がかかっている．このことから，実用的な起動速度を確保することが難しい．

(3) **クラウド管理ソフトウェアのオーバヘッドが高い**

OpenStack や CloudStack は，いずれも複数のコンポーネントからなる大きなパッケージである．ARM への対応もコード内にあるが, RPi2 のような省サイズのコンピュータでは，インストールやインスタンス起動時の処理の負荷がかなり高い. 例えば, OpenStack のコントローラノードやコンピュートノードは 2GB のメモリが必要であり，RPi2 のような小型のコンピュータでの利用が難しい．また，現在はインストールスクリプトによってインストールが可能だが，I/O の速度が速くない RPi2 ではインストールに長時間を要してしまう．

3. 設計方針

本システムにおける設計方針を次に示す．

(1) サーバ側に物理リソース表示のためのインタフェースを備える
(2) あらかじめ構築されたクラウドサーバを用意する
(3) 安価かつ移動が可能な程度の小型のクラウドサーバとする．
(4) 軽量なクラウド管理ソフトウェアの提供

これらを踏まえ，次に示すような機能を備えた学習用クラウド環境を開発する．次にこれらについて説明する．

(1) **物理リソース表示のためのインタフェース**

これについては，学習者がクラウド側の構造や処理状態を把握しやすくするためである．

本システムでは，クラウドについての学習の観点から，仮想リソースと物理リソースの関係を明確に示す為に，各物理マシンで使用しているリソースを可視化して表示する機能を備える．方法として，どのマシンのリソースなのかが明確になる，リアルタイム性が向上するといった理由から，LED 等を用いた専用のインタフェースを用意する．

(2) **あらかじめ構築されたクラウドサーバの提供**

これについては，プライベートクラウドを構築するのは手間がかかるためである．あらかじめ整えられたクラウド環境を用意することで，学習をはじめる際の煩雑さを軽減

Vol.2016-SE-192 No.5
Vol.2016-EMB-41 No.5
2016/6/2

し，学習者がスムーズに実験を始められるようにする．

また，ハードウェア環境があらかじめ構築されていれば，環境構築に伴うマシンの選定なども不要になり，学習者はソフトウェアについての学習に集中することができる．

(3) **安価かつ移動が可能な程度の小型のクラウドサーバ**

運用コストや設置場所の観点から，必要に応じてサーバを移動させることができれば，サーバ室のようなサーバを配置するためのスペースが不要になる．

また，学習の際に，物理リソースの状態確認やマシン構成の組換えなどの実験を行うためには，学習者から離れた位置にサーバを配置してあるよりも，学習者の近くにクラウドサーバが配置されていることが望ましい．さらに使用されているマシンが安価であれば，学習者自身がマシンを購入して発展的な実験を行うことも容易になる．よって本システムは，学習の際に教室間等の移動や，手元に置いて実験のための作業を行いやすくするために，システム全体を自由に移動できる程度の大きさで構成する．

(4) **軽量なクラウド管理ソフトウェアの提供**

学習者が学習を進めるうえで必要となるのは，システムが実際に運用されるクラウドサーバと同様に複数のマシンで分担して動作することと，学習の妨げとならないよう軽快に動作することである．そこで，クラウドとして最小限の機能を備えたクラウド基盤システムを開発する．

4. 設計

本章ではシステムの設計について述べる．

4.1 全体構成

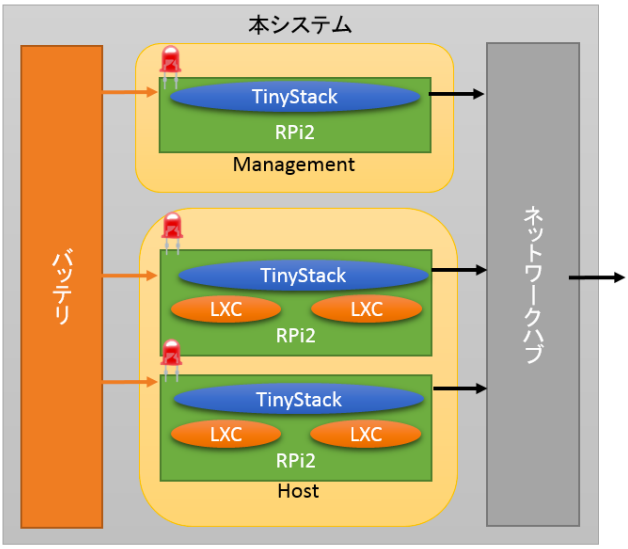

図 1 全体構成

Figure 1 Overall Structure.

システムの全体構成を図 1 に示す．システムは管理ノードとホストから構成している．管理ノードでは，システム全体のモニタリングおよび各ホストの動作の管理および仮想マシンの起動，終了のための Web フロントエンドを提供している．コンピュートノードを複数動かし，サーバを提供する．

ボードにはすべて RPi2 を用いる．RPi2 は，CPU が ARM Cortex-A7 クアッドコア・900MHz，メモリも 1GB と従来の Raspberry Pi と比較して性能が向上している．本体サイズも 85.60mm × 56．5mm と小型であり，小型化を目指す本システムにおける要件を満たしている．RPi2 上では Raspbian をはじめとする Linux 環境が動作することから，クラウドのノードとして十分であると判断した．ストレージについては，起動イメージは SD カードを用い，クラウドサーバで提供するストレージについては，USB2.0 で接続された SSD を管理ノードで用意して，それを NFS で共有している．各ノードはネットワークハブを介して接続している．

システム全体はバッテリで動作できるコンポーネントだけで構築されているので，設置場所を選ばずに動作させることができる．教室のようなクローズドな環境でもネットワークを用意してやれば利用可能である．

4.2 TinyStack

前章で述べたとおり，小型で安価なコンピュータでは，PC を対象としたクラウド基盤ソフトウェアを動作させるのは困難である．

そこで，本システムでは，RPi2 用にクラウドとして最小限の機能を備えたクラウド基盤システムを作成し実装する．このシステムを TinyStack と呼ぶ．

TinyStack は，RPi2 上で動作することを前提とし，学習のための最低限の機能を備えた軽量のクラウド基盤システムである．主な特徴を次に示す．

(1) **軽量な仮想化機能である LXC のサポート**

小型ボードでの仮想化機能には Linux Container を用いた仮想化機構を用いた．具体的には，Linux Container の機能を，LXC を用いて提供する．Linux Container は，ホストの OS から提供されるリソースを隔離することで，仮想的な環境を並列で動作させる．このため，KVM などのようにホスト OS と異なる OS を動作させることはできないが，ハードウェアのエミュレートの必要がないためオーバヘッドが少なく，リソースを隔離しているだけなので仮想環境の起動や停止も高速である．ネットワークサービスを提供する場合，クラウド上ではサーバが起動すればよく，OS の違いが問題になることは少ない．これらの特徴が通常のコンピュータよりも非力な RPi2 上での動作に適しているので，本システムでは LXC を利用してクラウドを構成する．

(2) **管理ノードとコンピュートノードの二つを用意する**

サーバのスケーラビリティを保証するために，従来のク

Vol.2016-SE-192 No.5
Vol.2016-EMB-41 No.5
2016/6/2

ラウド基盤ソフトウェアと同様に，システム全体を管理する管理ノードと，サービスを提供するコンピュートノードの二つから構成する．コンピュートノードは，サーバの負荷に応じて台数を増やして負荷分散を行うことができる．コンピュートノード内のサービスは，前述の LXC によって複数起動することが可能である．

(3) 起動イメージ管理はしない

OS の起動イメージはクラウドの管理対象から外し，ローカルなマシン上に個別のコピーを置くことで，オーバヘッドを低減させた．通常のクラウド基盤ソフトウェアでは，起動イメージの管理は必須の機能となっている．しかし，クラウド利用において重要なことはネットワークサービスの提供であり，その点で OS のイメージを変えていくという利用状況はあまり発生しない．起動イメージはサイズが大きく，メモリやストレージの性能が高くない RPi2 では，性能を維持して扱うことは難しいと判断して，起動イメージ管理は行わないこととした．

(4) 物理リソースの可視化機能を持たせる

物理リソースの使用状態を，物理的に表示できる機構を用意した．多くのクラウド基盤ソフトウェアでは，ブラウザ内でリソースの利用状況を表示する機能を提供しているが，本システムではあえて LED による物理的な表示機構を用意した．これは，クラウドという仮想化されて分かりにくい環境において物理的な表示デバイスを用いることで，サーバという「モノ」を意識させることを狙っているからである．また，クラウド全体の仮想的な状況については，管理ノードへ情報を集約させてブラウザで表示させることで，仮想環境でのサーバ資源の状況と，物理環境での状況とを対比させて表示することを可能にする．

(5) Web サービスによって拡張を容易にする

拡張性が高く，機能を追加することができるという観点から，本システムは Python で記述し，HTTP+REST ベースで管理マシンとの通信を行う．

管理マシンから，コンピュートノードの IP アドレスにコマンド名を加えて指定することで，各種機能を利用する．これらの機能によって，ノードの情報表示やインスタンス管理などの操作を管理マシンから行うことが可能となる．通常のクラウド基盤ソフトウェアでは，ノード間の通信には RabbitMQ などのメッセージキューイングを利用しているが，本システムでは実装をシンプルにするために HTTP ベースでの通信を基本とした．

TinyStack では，接続されているコンピュートノードの一覧表示やインスタンスの作成等を行うことができ，各コンピュートノードではインスタンスとして LXC コンテナが起動する．１台の管理マシンと複数のコンピュートノードから構成し，それらを同一のネットワーク下に配置し運用する．

4.3 LED によるリソース表示機構

本システムでは，8×8 の 64 個の LED で構成される LED マトリクスを使用し，物理リソースの表示を行うようにした．LED を用いることにより，直観的な資源状況を把握することができ，動作を実感させることが可能になる．またマトリクスを利用することにより，複数の資源を一度に表示することが可能になることから，資源の相互関係も同時に理解することができる．

クラウドサービスにおいて重要となる次の資源の利用状況を可視化することにした．

- コンテナ数
- CPU 利用率
- メモリ利用率
- ネットワークトラフィック（Send/Receive それぞれ）
- IOPS(1 秒あたりの Input, Output 数)

これらを LED マトリックスの 1 行にそれぞれ対応させて表示する．

表示例を図 3 に示す．

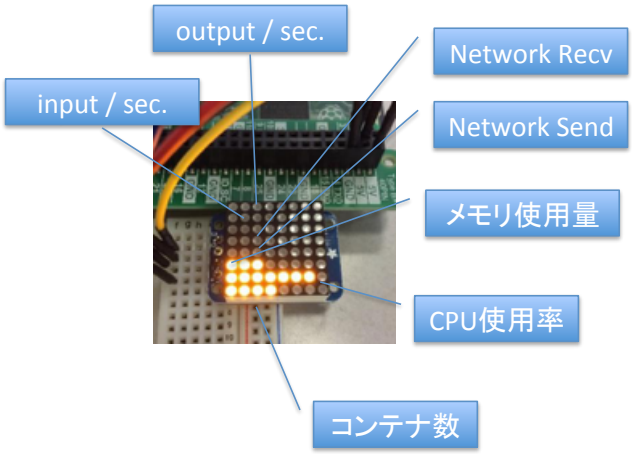

図 2 状態表示例
Figure 2 Example of Status Display on LED matrix.

この例では，四つのコンテナが起動し，CPU の負荷が高く，メモリの使用量は高くなく，ネットワークや IO は利用していないことが分かる．

システムの利用情報は，TinyStack が Linux のプロセスファイルシステムから情報を取得することで行っている．これらの情報は，管理ノードにも送信されて Web ブラウザ上からも確認することができる．

4.4 Web ページ

管理ノードが提供する Web ページを図 4 に示す．

Vol.2016-SE-192 No.5
Vol.2016-EMB-41 No.5
2016/6/2

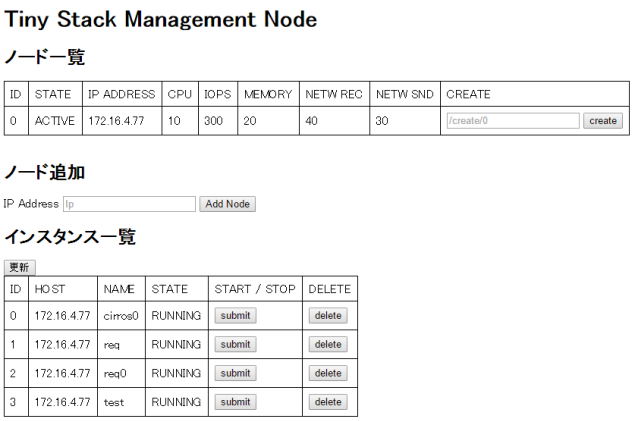

図 4 管理ノードの実行画面
Figure 4 Screenshot of Management Node.

Web ブラウザから，ノードの一覧やインスタンス生成，起動，停止，削除が可能である．また，スケーラビリティを上げるためにコンピュートノードを追加した場合に，それらを管理対象として追加することも可能である．ノードの状態については，LED 表示に加えてブラウザ上からも確認することができる．

コンピュートノードと管理ノードとは Web インタフェースによってだけ関連づけられているので，管理オーバヘッドを低く抑えている．

4.5 WebAPI

管理ノードおよびコンピュートノードが提供する WebAPI を表 1 に示す．コンピュートノードおよび管理ノードはすべて Web API を通してアクセスすることが可能である．機能は，CRUD（Create-Read-Update-Delete）にしたがって割り当てられていて，それらを HTTP のメソッドへ対応づけている．

5. 実現

本章では，実現および実行例について述べる．

5.1 実現

開発したマシンのイメージを図 3 に示す．本システムは 4 台の RPi2 で構成した．1 台の管理ノードと 3 台のコンピュートノードから構成している．すべてのノードに LED マトリックスを用いたリソース表示機構をつけた．

TinyStack および LED 表示については Python で記述している．Python の Web フレームワークである flask を用いて，Web サーバとして実現している．TinyStack 自体は管理ノードおよびコンピュートノードのいずれでも動作する．コンピュートノードでは，LXC へのフロントエンドおよび LED 表示機構を提供する．起動イメージとして，小型の Linux イメージである cirrOS を各ノード上に置き，Linux が起動可能であることを確認した．

表 1 Web API 一覧
Table 1 List of Web API

URL	method	機能
/usage	GET	ノードの使用状況を得る
/instances	GET	インスタンス一覧を得る
/instances	POST	インスタンスを生成する
/instances/<id>	GET	<id>で指定したインスタンスの情報を得る
/instances/<id>	POST	<id>で指定したインスタンスの起動と停止
/instances/<id>	PUT	<id>で指定したインスタンスの名前を変更
/instances/<id>	DELETE	<id>で指定したインスタンスを削除

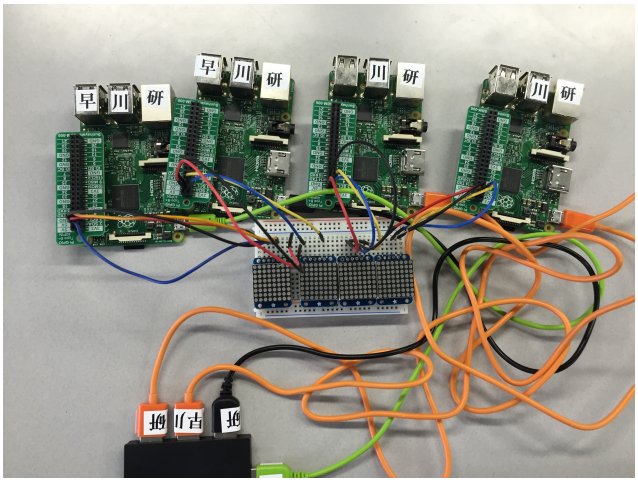

図 3 開発したクラウドサーバ
Figure 3 Developed Cloud Servers

5.2 実行例

本システムによって，ローカルクラウド環境を容易に準備できるようになった．さらにシステムから提供される Web インタフェースと実マシンの LED 表示をあわせて，管理マシンからインスタンスの起動を指示すると，直後に

Vol.2016-SE-192 No.5
Vol.2016-EMB-41 No.5
2016/6/2

対応したコンピュートノードの LED が点灯するなど，仮想マシンと実マシンの状況を比較することができるようになった．これによって，クラウド開発や管理を行う際に起きうる問題のテストを容易に行うことができる．

例として，図 4 に示す中央のマシンに CPU 負荷をかけた場合，図 5 に示す通り，CPU 利用率を表す左から 2 列目の LED の点灯数が増えている．このように，LED の点灯状態によって過剰に使用されているリソースが一目でわかるため，トラブルの発生に気づきやすく，学習中に起こるトラブルの原因特定にも役立つ．

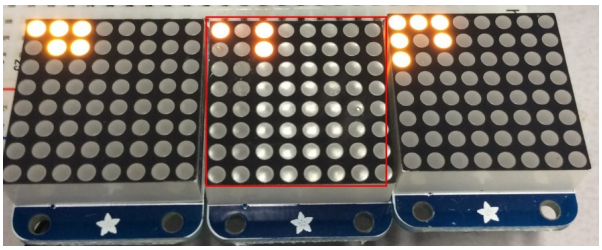

図 4 負荷をかける前の状態
Figure 4 LED Display before increasing CPU load

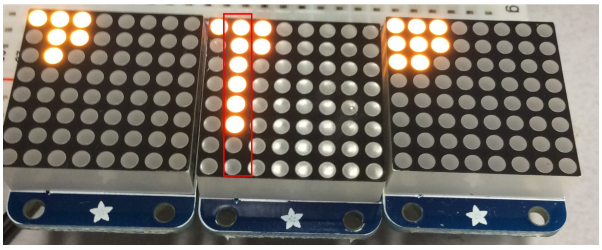

図 5 負荷をかけた後の状態
Figure 5 LED Display after increased CPU load

6. 関連研究

Raspberry Pi は性能／コスト比が高いことから，これを用いてクラスタを構築するシステムは数多く存在する[4,5]．Idris-pi では，64 台の Raspberry Pi を接続し，MPI のようなライブラリをサポートしてクラスタを構成し，計算ノードとして利用可能である．一方，[4]では，マシン管理コストの問題があるが，ほぼ個人利用のシステムでは問題にならないとしている．一方，本システムが対象とするクラウドシステムでは，マシン管理は重要な課題となる．クラウド基盤ソフトウェアを提供することにより，マシン管理で生じる問題を解決している．

クラウド基盤ソフトウェアとしては，OpenStack[1]や CloudStack[2]が用いられる．いずれも PC が主体であり，PC を中心に機能が実装されている．2.2 で述べたようにインストールおよび実行時のオーバヘッドが高すぎて，RPi2 のようなマシンでの利用は困難である．本システムでは，軽量な基盤ソフトウェアを用意することでこれらの問題を解決している．

クラウド監視については，Nagios[6]や Munin[7],Zabbix[8]など数多くのシステムが存在する．規模の大きな Web システムを主眼としていて，本システムが対象とするようなスケールに対しては，サイズおよび機能が大きすぎるという問題がある．

7. おわりに

本報告では，小型ボードである Raspberry Pi を用いたクラウド基盤システムの構築について述べた．教室のようなネットワーク帯域が不十分な環境において，組込みシステムからクラウド環境を使用するために，可搬サイズの Raspberry Pi による環境を構築した．Raspberry Pi は CPU の性能面でクラウドサーバとして十分機能するものの，既存の基盤システムでは資源の消費量やサイズが大きいことから，省サイズの基盤システムを開発した．また，物理資源の使用量を容易に把握できるように，LED を用いた資源使用量の表示機構を提供した．

現在のシステムは，クラウドサーバとして最低限の機能を提供しているだけであり，実用化するにはより機能を追加する必要がある．特に，組込みシステムとの連携を想定した場合は，MQTT[9]などのメッセージングプロトコルのサポート，低オーバヘッドの起動イメージ管理機構，サーバのレイテンシのモニタリングが必要になる．システムのモニタリングやログストレージについては，我々の研究室で開発している BlueSky[10]がすでに提供していることから，これらのシステムとの連携も検討する．

参考文献

[1]OpenStack：http://www.openstack.org/（参照 2016/05/10）
[2]CloudStack： http://cloudstack.apache.org/（参照 2016/05/10）
[3]KVM/ARM https://systems.cs.columbia.edu/projects/kvm-arm/ (参照 2016/05/10)
[4]Simon J. Cox, James T. Cox, Richard P. Boardman, Steven J. Johnston, Mark Scott, Neil S. O'Brien, Iridis-pi: a low-cost, compact demonstration cluster, Cluster Computing, DOI: 10.1007/s10586-013-0282-7, June 2013
[5]信田 圭哉， 長谷川 明生, 安価なコンピュータを用いた実験・教育用並列計算機環境の構築, 情報処理学会インターネットと運用技術研究会研究報告，2015-IOT-30, 7, pp.1-7, 2015
[6]Nagios, https://www.nagios.org/ (参照 2016/05/10)
[7]Munin, http://munin-monitoring.org/ (参照 2016/05/10)
[8]Zabbix, https://www.zabbix.org/wiki/Main_Page (参照 2016/05/10)
[9]MQTT, http://mqtt.org/ (参照 2016/05/10)
[10]プラウィーンアモーンタマウット，早川 栄一，分散組込みシステム向き Web ベース開発環境，組込みシステムシンポジウム 2015 論文集，pp.34-39, 2015