IoT アーキテクチャとしてのリアルワールド OS アーキテクチャデザイン

○竹林康太　出口弘 （東京工業大学大学院） 倉田正 石塚康成 木寺重樹 (株式会社パイケーク)

The architecture design of the Real World OS as an IoT architecture

∗K. Takebayashi, H. Deguchi (Tokyo Institute of Technology), T. Kurata, Y. Ishizuka and S.Kidera (Piecake Incorporated)

1　はじめに

近年，IoT というキーワードが世間を賑わすと同時に，それに伴う形で様々な IoT プラットフォームが各社から発表された．これらの IoT プラットフォームを活用することで, 社会問題の解決, 産業の活性化などが期待される. しかし, 数多くある IoT プラットフォームはそれぞれのメリット・デメリットが明確にされておらず, 開発者が必要とする要件を満たすためのプラットフォームを選ぶ術がないという問題点がある.

本研究ではまず, これら IoT プラットフォームについて計算処理を実現する階層に着目することで 3 つのアーキテクチャに分類し，各 IoT プラットフォームの特徴を説明する．更に，それらアーキテクチャの一つである当研究室で開発中のリアルワールド OS について，他 IoT プラットフォームとの処理能力の違いを実験を通して比較した上でリアルワールド OS がどのようなソリューションに適しているかを考える.

2　IoT アーキテクチャの比較とリアルワールド OS の特徴

IoT アーキテクチャとは，センサーや製造装置，情報端末など多様なデバイス同士による多対多の通信を想定して作られたシステムアーキテクチャである. ここでは, 各種センサーなどの情報を扱う「データ収集レイヤ」，各データの計算処理を行うための「計算処理レイヤ」，利用者や開発者に情報を提供するための「アプリケーションサービスレイヤ」の 3 つのレイヤを基本とするアーキテクチャと定義する.

Cisco は，ネットワークに接続されたデバイスは 2020 年までに 500 億台に到達すると見込んでおり [2]，今後 10 年間で IoT の経済価値は全世界において約 14 兆ドルにのぼるとしている [3]. クラウドコンピューティングなどに代表される従来のような集中管理型 (1 対多型のモデル) のアーキテクチャでは，このような膨大な数の通信プロセスは処理しきれない．そこで，多対多型のモデルによる次世代のアーキテクチャを規格化すべく，多くの企業や団体が IoT アーキテクチャを提案するようになった.

各アーキテクチャ毎の特徴を更に詳しく見ていくため, IoT アーキテクチャの代表としてここでは Amazon の Amazon Web Services IoT[4](以下,AWS IoT) と Cisco IOx，そしてリアルワールド OS のコンセプトを紹介する.

2.1　AWS IoT

AWS IoT は, 接続されたデバイスが簡単かつ安全にクラウドアプリケーションやその他のデバイスとやり取りできるマネージド型クラウドプラットフォーム [7] として設計されている. 従来のクラウドコンピューティング技術を使用したものだが，センサデータの収集方法に MQTT を取り入れており, より IoT に適応しやすいプラットフォームとなっている.AWS IoT では, 標準で X.509 証明書をベースとした認証がサポートされており, あらゆる場所にあるデバイスを安全に接続できることを強みとしている.

AWS IoT の階層モデルを次の Fig. 1 に示す.

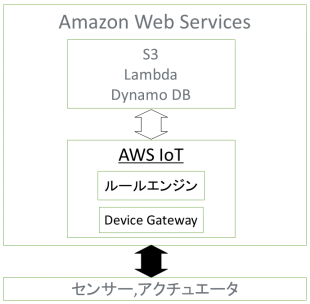

Fig. 1: AWS IoT の階層モデル

AWS IoT は,IoT に対してクラウドサーバ上の “アプリケーション層” に着目したプラットフォームといえる. これは,AWS が持つ強大なデータセンター基盤により可能としたものである. センサデータの値をリアルタイムに計算し, グラフによって可視化する, といったシステムであれば素早い構築が可能である.

2.2　Cisco IOx

Cisco IOx とは, ネットワーキングソフトウェアである Cisco IOS にオープンソースソフトウェアの Linux OS を組み込んだものであり, フォグコンピューティングを実現するための IoT プラットフォームである. フォグコンピューティング型のアーキテクチャは Cisco が提唱するもので，集中管理用のサーバは持つものの，大半の計算処理は各事業所に設置されたサーバで行うことで負担を軽減する．Cisco は, クラウドコンピューティングの概念を拡張したものがフォグコンピューティングであるとし [6], 実世界に近い位置付けがなされているのがフォグコンピューティングの特徴であるとしている.

Cisco IOx を以下に Fig. 2 に示す [6].

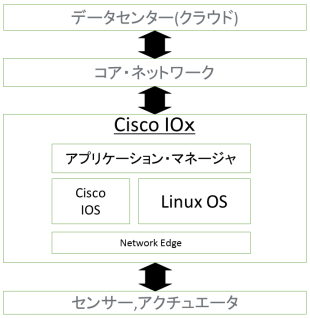

Fig. 2: Cisco IOx n 階層モデル

Cisco IOx は,“ネットワーク層”の計算処理を分散化することに着目したプラットフォームといえる. これにより, 従来のデータセンターでは対応しきれない量のセンサデータを, フォグコンピューティング技術により各事業所ごとに処理をすることで大規模なシステムの運用を可能とする.

2.3 リアルワールド OS

リアルワールド OS は出口研究室で開発された IoT プラットフォームである. リアルワールド OS の “OS”は Linux OS のような CPU やメモリなどのマシンリソースを仮想化するための OS ではなく,IoT 時代における, 人や端末, タスクなどから構成されるワークフローとしての CPS(サイバーフィジカル空間) 上のアプリを表現し動作させるための OS である [8].

リアルワールド OS の階層モデルを，以下に Fig. 3 として示す.

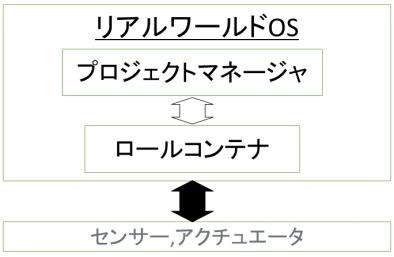

Fig. 3: リアルワールド OS の階層モデル

リアルワールド OS は, ネットワーク機器やデータセンター基盤の技術の延長ではなく,CPS という新たな枠組みのもとに作られたプラットフォームである. そのため,“ビジネスソリューション層”という, アプリケーション層よりもさらに上位の層に着目した IoT プラットフォームといえるだろう.

最後に,IoT 時代における CPS という観点のもと, これら 3 つの IoT プラットフォームを一つの図として表現することを考える.CPS とは, 一般社団法人 電子情報技術産業協会の定義によると「実世界（フィジカル空間）にある多様なデータをセンサーネットワーク等で収集し, サイバー空間で大規模データ処理技術等を駆使して分析・知識化を行い, そこで創出した情報・価値によって, 産業の活性化や社会問題の解決を図っていくもの」[5] となっている.

CPS と IoT の関係についてまとめたものを Fig. 4 に示す.

Fig. 4: CPS と IoT の関係

すなわち, これからの IoT,IoE の時代におけるシステム構築の基本的な考え方になるものである. これを踏まえ,3 つの IoT プラットフォームがそれぞれ重視する階層に注目し, 対応付けたものが次の Fig. 5 である.

Fig. 5: 各 IoT プラットフォームの CPS 上の位置付け

Cisco IOx や AWS IoT が従来のコンピュータアーキテクチャにあるような ICT レイヤーに重きを置いているのに対し, リアルワールド OS はビジネスソリューションという,IoT 時代を見据えたコンセプトのもとにアーキテクチャ設計を目指している.

3 リアルワールド OS の実装検討

3.1 リアルワールド OS の特徴

前項では，リアルワールド OS と他 IoT プラットフォームを比較するため,CPS という概念のもとにそれぞれの位置付けを考え, リアルワールド OS がビジネスソリューションという上位の層に存在することが分かった．次に, このようなリアルワールド OS がもたらすメリットと問題点について紹介する.

3.2 メリット : ワークフローモデルによるプロジェクトの統括的な管理

ワークフローモデルについて, 出口は「あらゆる『ひと』『もの』（センサーや機械など）『ソフトウェア』（計算ノードや人工知能のプログラムなど）からなる『(IoE) ノード』によって実行される『仕事の単位プロセス（タスク）』が, インターネット（サーバーフィジカル空間 : CPS）上で相互に結びつくこと」と定義している [8].

たとえば,「農場の作物管理をリアルワールド OS で実装する」という事例を考える. 物理上の各ノードの位置関係とデータフローを表したのが次の Fig.6 の上側であり, これをリアルワールド OS においてワークフローとして記述したものが下側のグラフである.

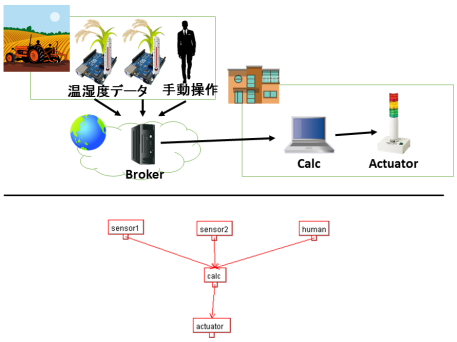

Fig. 6: ワークフローによる記述

このように, 各ノードの端末が農場, データセンタ, 自宅というように分散されて置かれたとしても, ワークフロー上では一つのサイバーフィジカル空間として統括的に管理することができる. これは, 他の IoT プラットフォームには備わっていない, リアルワールド OS の大きな特徴である.

3.3 メリット：開発工数の削減

もう一つの大きなメリットとして，開発工数の削減が挙げられる．従来のソフトウェア開発ではワークフローを前提としてモジュールの構成を静的にデザインする必要があり, ワークフローに変更が生じる度にモジュールの構成を修正しなければならず, 多大な労力を要することとなる.

リアルワールド OS では, 各ノード上のアプリ同士がデータをやり取りする際, ロールコンテナを介することで動的な結合が可能となる. これにより, ワークフローに変更が生じたとしても機能モジュールに修正を加える必要はなく, 迅速なソフトウェア開発が見込める.

3.4 リアルワールド OS の問題点とその検証

リアルワールド OS ではワークフローと各ノード間の通信において MQTT と呼ばれる通信プロトコルを使用する. そのため, リアルワールド OS では MQTT による通信で発生するオーバヘッドがボトルネックとなりうる. そこで, 今回はリアルワールド OS の基本パターンとして使用する「3 ステージモデル」を用いた検証を行った.

3.5 3 ステージモデル

3 ステージモデルとは，プロジェクトプログラミングのシンプルな運用モデルの一つであり，Fig.7 で示すように「情報収集ステージ (Sensor)」「計算ステージ (Calc)」「機器制御ステージ (Actuator)」の 3 ステージで構成される.

Fig. 7: 3 ステージモデルの運用ケースの比較

3 ステージモデルはセンサーデータを活用するあらゆるビジネスソリューションに適応可能であり, 例えばオフィスにおける照明システム管理, 農場における作物管理, 工場における機器の自動制御といった課題を解決できる.

4 メッセージング機能の設計と評価

次に, この 3 ステージモデルを動作する際に発生するオーバヘッドを検証するため,3 つの方法による処理時間の測定を行う. まず, リアルワールド OS で使用する MQTT ブローカーについて, ここでは Mosquitto と MQTT for AWS IoT の 2 つによる比較実験を行う.

4.1 MQTT，MQTT ブローカーとは

MQTT は，TCP/IP ネットワークで利用できる通信プロトコルの一つで，多数の主体の間で短いメッセージを頻繁に送受信する用途に向いた軽量なプロトコルである [9]．MQTT では,Publish/Subscribe モデルと呼ばれる, 一対多による非同期通信を行うための仕組みがある.MQTT　メッセージングモデルを Fig. 8 に示す.

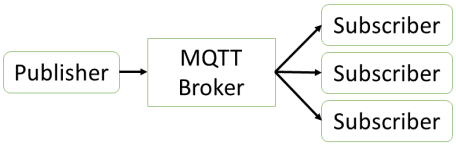

Fig. 8: MQTT メッセージングモデル

メッセージの送り側を “Publisher(配信者)”, 受け取り側を “Subscriber(購読者)”, メッセージの仲介サーバを “Broker” と呼ぶ. メッセージには Topic と呼ばれるタグがセットとなっており, 受け取るメッセージの選別を行うことが出来るワイルドカードを使用することでブロードキャストも可能となる. 従来の TCP/IP 通信と大きく違い, 情報の送り先の IP アドレスや同期の必要性がないため,Publisher と Subscriber のプログラムはお互いに依存することなく書くことができる.

MQTT の問題点として, キューイングバッファを持たないということが挙げられる. この問題に対しては, 一般的には外側にバッファ機能を持たせることで解決を図る. たとえば Amazon Web Services には,KVS の仕組みとして DynamoDB が用意されている. リアルワールド OS では, 各ロールコンテナにキューイングバッファを保つ仕組みがあり, これを利用することが出来る. Pub/Sub メッセージを仲介するためのサーバ (従来の TCP/IP 通信におけるホストサーバにあたる) をブローカーと呼ぶ. 2016 年 2 月現在，Mosquitto をはじめとする様々なブローカー用サーバが存在する．以下の Table 1 は, 主な MQTT ブローカーの特徴や公開形態などをまとめたものである.

Table 1: 主な MQTT ブローカーの一覧

ブローカー名	開発元, メリット, 公開形態
Mosquitto	Eclipse 財団, 無料利用可能, オープンソースソフトウェア
MQTT for AWS IoT	Amazon Web Services, AWS 各サービスとの連携可能, 従量課金制
Akane[10]	時雨堂, クラスター利用可能, 年間ライセンス制

4.2 実験方法

まず 1 つ目の実験方法では, リアルワールド OS で使用するブローカーの性能評価である. このブローカーに対し様々なサイズのメッセージを Publish し，Subscribe として返ってくるまでの時間を測る．なお,MQTT にはメッセージの配達保証を行うための QoS という仕組みがあるが, 今回この機能は使用しない (QoS0 とする) こととした.

AWS IoT ではブローカーとして “Message Broker for AWS IoT”が用意されている．リアルワールド OS では専用のブローカーは特に設けられいないため，ここでは前項で説明した Mosquito を用いた場合として実験を行う．以下，この実験を “実験 1”とする．実験 1 の構成を以下に Fig. 9 として示す.

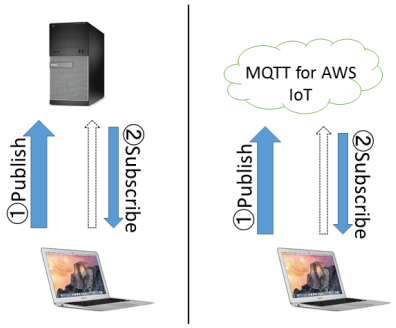

Fig. 9: 実験 1 の構成

2 つ目の実験では，3 ステージモデルを仮定した場合でのリアルワールド OS の処理能力を調べる．ブローカー単体による処理内容とは違い，リアルワールド OS 全体の処理時間を調べることとなる．以下，この実験を “実験 2”とする．実験 2 の構成を以下に Fig. 10 として示す.

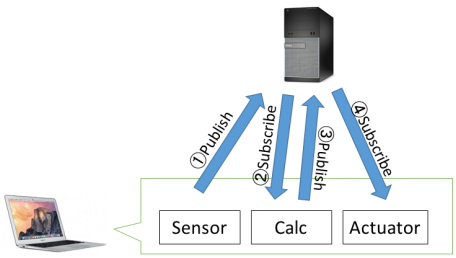

Fig. 10: 実験 2 の構成

最後に，実際のシステムを考慮し, センサーの台数を 3 個,10 個の 2 通りにした場合での処理能力を調べる．以下，この実験を “実験 3”とする．実験 3 の構成を，以下に Fig. 11 として示す.

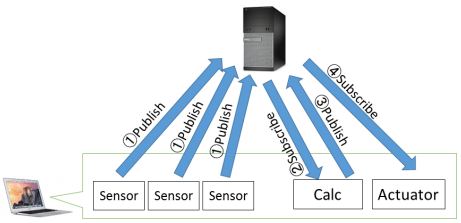

Fig. 11: 実験 3 の構成

4.3 実験環境

今回の実験環境について，リアルワールド OS のネットワーク構成図について以下の Fig. 12 に示す．また，今回使用した MQTT サーバ，プログラム実行端末のスペックについて，Table 2 に示す．そして，サーバ間の距離について，予め Ping コマンドを実行させた際の動作時間について,Table 3 に示す. なお,AWS ではサーバの拠点として世界中のリージョンを選択できるようになっているが, 今回は東京リージョンを選択した.

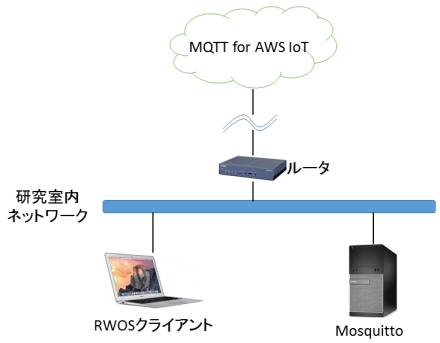

Fig. 12: 実験環境のネットワーク構成図

Table 2: 実験で使用した端末のスペック

端末	スペック
MQTT サーバ	OS:Debian 64bit, CPU:intel Core i5-4590 3.30GHz, RAN:4GB
RWOS クライアント	OS:Mac OSX, CPU:intel Core i5-4250U 1.3GHz, RAM:4GB

Table 3: Ping の実行時間

MQTT サーバ	速度 (msec)
Mosquitto	2.428
MQTT for AWS Iot	6.348

さらに，実験 1 の実験方法についてまとめたものを Table 4，実験 2 の実験方法についてまとめたものを Table 5，実験 3 の実験方法についてまとめたものを

Table 6 として，それぞれ示す.

Table 4: 実験 1 の実験方法

項目	説明
実験対象	Mosquito， Message Broker for AWS IoT
実験方法	MQTT ブローカーの通信 (2 端末間の Publish-Subscribe の通信を 1 サイクルとする) に掛かる時間を測定する
扱うデータ	3 種類用意する (センサデータを想定した, それぞれ 1 バイト・1 キロバイト・100 キロバイト・1 メガバイトから成るバイナリデータ)
比較対象	1 サイクルに掛かる時間 (10 回行ったうちの平均とする)

Table 5: 実験 2 の実験方法

項目	説明
実験対象	リアルワールド OS
実験方法	提示した 3 ステージに沿ったシステムをそれぞれのプラットフォーム下において作成し，これを実行する.
扱うデータ	1 つのセンサノード・計算ノードから Publish されるデータは 10 バイト程度のものを，実行ノードから Publish されるデータは 1KB のバイナリデータとする
測定対象	1 サイクルに掛かる時間 (10 回行ったうちの平均，最短処理時間，最長処理時間を記録する)

Table 6: 実験 3 の実験方法

項目	説明
実験対象	リアルワールド OS
実験方法	実験 2 で使用した 3 ステージのモデルについて，センサノードをそれぞれ 3 個,10 個とした際の速度を測定する.
測定対象	1 サイクルに掛かる時間 (10 回行ったうちの平均，最短処理時間，最長処理時間を記録する)

4.4 実験結果・考察

実験 1 について，結果は以下の Table 7 のとおりとなった.

Table 7: 実験 1 の結果

サイズ	Mosquitto	AWS
1Byte	0.039s	0.026s
1KB	0.022s	0.028s
100KB	0.080s	0.144s
1MB	0.817s	−

上記結果より,1KB 程度のメッセージであれば,Mosquitto と MQTT for AWS IoT の速度はそれほど変わらないものであることが分かる.100KB 以上から成るメッセージやバイナリデータの場合,Mosquitto の方が約 2 倍の速さでメッセージを処理できる上,AWS では対応ができなかった 1MB 以上のバイナリデータを扱うことが出来るという利点も判明した.

続いて，実験 2 の結果について以下の Table 8 として示す.

Table 8: 実験 2 の結果

平均時間	最短時間	最長時間
0.394s	0.210s	0.576s

計算処理に掛かる時間に影響されるが, シンプルな設計であれば, センサデータの送信から 1 秒もかからないうちに全ての処理を終えることが出来ることが分かる.

最後に，実験 3 の結果について以下の Table 9 として示す.

Table 9: 実験 3 の結果

ノード数	平均時間	最短時間	最長時間
3	0.486s	0.274s	0.716s
10	1.350s	0.912s	1.777s

センサーの数が多い場合においても, リアルワールド OS は各ノードに対して並列処理を行うことが出来る. しかし,MQTT による通信がボトルネックとなり, 実行時間が比例して伸びてしまう傾向にあることが分かる.

リアルタイム性を重視するシステムであればノードの数は 5 個程度に抑えておくのが良いだろう. 一方で, 多少の遅延を気にしないシステムであれば, 接続されるノード数が数十個程度であっても十分に実務として利用可能である.

5 まとめ

今回の研究では，各企業から相次いで発表された IoT プラットフォームの位置付けと，その中でのリアルワールド OS の立ち位置について見ることができた．また, リアルワールド OS 上のシステム開発でのデザイン・パターンとして,3 ステージモデルのシナリオで検証を行った.

リアルワールド OS は，小規模〜中規模タイプの IoT 向けシステム構築において，迅速なシステム開発および管理を行う上では欠かせない IoT プラットフォームとなるであろう．特に，工場における機器の制御，ビル

の照明・空調機器の制御などといった事例には最適である．リアルワールド OS は今後, 様々なソリューションに柔軟に対応できるよう, 汎用性のある多くのモジュールを実装していく予定である. これにより, 開発者は GUI 上でモジュールを選択し, 幾つかのパラメータを変更するだけで要件に沿ったシステムを素早く構築できるようになるだろう.

リアルワールド OS の適用を本格化するためにはセキュリティへの対策が重要である, 今後はセキュリティ機能を備えていく必要がある. また, より多くのノードを扱う大規模な要件にも対応できるよう, 処理速度を向上させるための改善をしていく必要があるだろう.

リアルワールド OS を利用したシステムを開発する際には費用対効果の見積りも重要であり, このためのシミュレーションシステムを開発する予定である．このシステムは室内環境の改善という具体的なソリューションを対象とした上で，電力利用の効率的な改善のために必要となるセンサのコストと，それによる効果に関するシミュレートを行うことのできるツールが含まれている．また，リアルワールド OS を活用した事例として, ワイヤレス環境下での自律移動を可能とする汎用的な小型アクチュエータの開発を進めている (Fig.13).

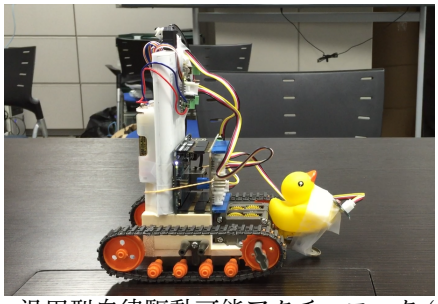

Fig. 13: 汎用型自律駆動可能アクチュエータ (プロトタイプ)

このアクチュエータは MQTT による無線通信, 温湿度センサーや照度センサーなどのセンシング機能を自由に取り付けることができ, 室内外を問わず動作可能な安価なアクチュエータとして利用されることを想定している.

参考文献

1) IT 用語辞典 e-words, http://e-words.jp/w/
2) http://www.cisco.com/web/JP/news/pr/2014/012.html
3) シスコシステムズ合同会社, IoT インキュベーションラボ:Internet of Everything の衝撃, 26, インプレス R&D (2013)
4) https://aws.amazon.com/jp/iot/
5) http://www.jeita.or.jp/cps/about/
6) https://communities.cisco.com/docs/DOC-57363
7) https://aws.amazon.com/jp/iot/
8) https://www.esd21.jp/news/20151212
9) http://e-words.jp/w/MQTT.html
10) http://akane.shiguredo.jp/