エミュレータを用いた大規模IoTシステムテストのための実験環境構築機能の開発

中村 拓人[1,2,a)]　大畑 誠弥[1,2,b)]　湯村 翼[1,3,c)]

概要：Internet of Things (IoT) 機器を用いたシステムは普及が進み，様々な用途で利用される．IoT システムの安定稼働には試験が肝要であるが，大規模な IoT システムを試験を実施するためには実機と場所を要する．情報通信研究機構の StarBED は， 大規模な実験を目的とし， 多数のコンピュータノードで構成されたネットワークテストベッドである．StarBED のノードの CPU は x86_64 アーキテクチャであるが，エミュレータを活用すれば， ARM 等の組込み機器向け CPU を用いた IoT システムの試験を実施することができる．しかし， StarBED の複数ノード上で多数のエミュレータが稼働する任意の実験環境を構築することは， 非常に煩雑な作業となる．そこで本研究では， エミュレータを用いた大規模な実験環境を簡便に構築するための実験環境構築機能を開発する．エミュレータの実行には Docker によるコンテナ仮想化を利用し，StarBED の複数ノードの管理には Docker Swarm でのクラスタリングを用いた ．構築する環境の情報は 1 つの設定ファイルに集約して記述する．設定ファイルの記述形式を Docker Network ごとに階層化し，エミュレータの適切な分散配置を可能とした．実験環境構築機能の性能評価として環境構築に要する時間を計測し，構築環境のモジュール数が多い場合には想定通り分散処理によって構築時間が短縮されることを示した．

1. はじめに

Internet of Things(IoT) 機器を用いたシステムは様々な場面で利用される．IoT システムはモノやヒトのデータ収集に用いられ，大量のデータを獲得することができる．そのデータを分析し利活用し，新たなサービスを生み出すことが期待される．より大きな価値を生み出すためには，より大規模な IoT システムの構築が求められる．大規模 IoT システムを安定的に稼働させるためには試験が肝要である．しかし，大規模な IoT システムの試験には多くの実機と広い場所を要するため，費用などが障壁となり，容易にできるものではない．そこで我々は，情報通信研究機構のテストベッド StarBED[1][2] を用いて大規模 IoT システムテストを行うことを検討する (図 1)．StarBED は，多数のコンピューターノードで構成されたネットワークテストベッドであり，各コンピュータノードには x86_64 アーキテクチャの CPU である Intel Xeon が搭載される．IoT システムでは ARM 等の組込み機器向け CPU が多く用いられるが，エミュレータを活用すれば StarBED でも IoT システムの試験を行うことができる．しかし，複数の StarBED ノードを用いて試験のための実験環境を構成する際，ノードごとの個別の環境構築には大きな手間がかかり，ネットワーク設定などの複数ノードにわたる設定も非常に煩雑となる．大規模試験を行う場合には，適切な負荷分散も考慮する必要がある．

これらの課題を解決するため，我々はエミュレータを用いた大規模な実験環境を簡便に構築するための実験環境構築機能を開発する．この実験環境構築機能は，1 つの設定ファイルへの記述と 1 つのスクリプトの実行により，複数ノードにわたる実験環境を構築する．本論文は，本研究で想定する IoT システムとその検証のための実験環境，開発した実験環境構築機能の設計と実装，および実験環境構築機能の評価のための計測結果について記述する．

2. IoT システムのエミュレーション

2.1 想定する IoT システム

本研究で対象とする IoT システムは，Sensor，Gateway，HTTP Server の 3 種類のモジュールからなる構成のもの (図 2) を想定する．Sensor は，物理量の計測を行うモジュールである．Sensor には様々な種類があるが，本研究では温度センサを想定する．Sensor は TCP/IP によるインター

[1] 情報通信研究機構
[2] 金沢工業大学
[3] 北陸先端科学技術大学院大学
[a)] hitoron@nict.go.jp
[b)] silmin@nict.go.jp
[c)] yumu@nict.go.jp

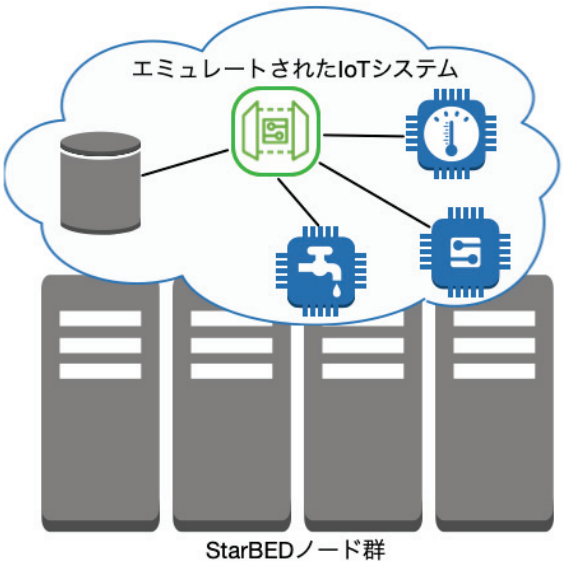

図 1　StarBED で行う大規模 IoT システムテストの概念図

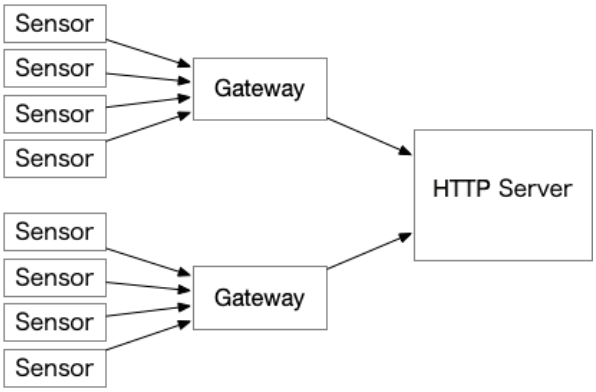

図 2　テスト対象となる IoT システムの構成

ネットへの直接の疎通性は持たず，Gateway を介して通信を行う．Sensor は，温度データを JSON 形式で Gateway に送信する．Sensor-Gateway 間の通信は，Bluetooth や ZigBee 等の近距離無線通信で行われることを想定する．Gateway は，Sensor から受け取ったデータを HTTP Server に送信する．HTTP Server は，データを収集するための，一般的な Web サーバである．受信した情報をログに残す機能を持つ．

2.2　エミュレーション

本研究では，上述の IoT システムを StarBED にてエミュレートして実行する．エミュレータは独自実装したもので，x86_64 アーキテクチャの Linux 上で ARM Cortex-M0 および M4 アーキテクチャの命令セットを実行するソフトウェアである．エミュレータは，1 つのモジュールを 1 つのプロセスとして実行する．Sensor エミュレータおよび Gateway エミュレータは，FreeRTOS[3] を用いて実装したバイナリをエミュレータ上で実行する．エミュレータで用いる温度は，事前に定義された温度テーブルからランダムに選択する．Sensor-Gateway 間の通信は近距離無線通信を想定しているが，エミュレータでは UNIX ドメインソケットによるプロセス間通信を行う HTTP サーバは，IoT デバイスではなく Linux サーバで稼働することを想定するため，エミュレータではなく StarBED のノード上でベアメタルサーバで稼働する．

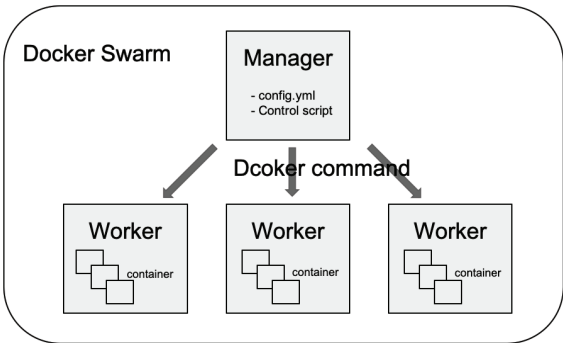

図 3　Docker Swarm によるクラスタリングの構成

2.3　StarBED

IoT システムのエミュレータは，StarBED で稼働することを想定する．StarBED は，多数のコンピュータノードで構成されるネットワークテストベッドである．それぞれのノードは，x86_64 アーキテクチャの CPU である Intel Xeon を搭載する市販のコンピュータであり，通常は Linux や Windows などの OS をインストールして使う．実験環境を複数台のノードを用いて構築することも想定され，各ノードはスイッチを経由してイーサネットで接続される．StarBED は汎用ネットワークテストベッドであり，ネットワークに関する様々な実験に用いられる．エミュレータを活用した実験も行われる [4][5]．過去の研究では StarBED での環境構築を支援するツール Alfons[6] も開発されるが，Alfons では本研究で扱うコンテナ仮想化は対象ではない．

3.　実験環境構築機能の設計

本研究では，Docker[7] によるコンテナ仮想化を実験環境構築に用いる．Sensor または Gateway の 1 つのエミュレータを 1 つのコンテナとして構築する．複数の SterBED ノードを用いて環境構築するために，Docker Swarm によるクラスタリングを行う．Docker Swarm では，クラスタリングされたコンピュータは管理ノードと実行ノードに分けられ，それぞれ Manager，Worker と呼ばれる (図 3)．Manager は Worker に対して docker コンテナの起動命令を送信することができる．送信された docker コンテナの起動命令は Worker で実行される．

本研究では，実験環境構築を行うための実験環境設定ファイルとコントロールスクリプトを Manager に配置し，環境構築を実行する．実験環境設定ファイルは，実験環境の構築に必要な情報が YAML 形式で記述される．コント

ロールスクリプトは，実験環境設定ファイルを読み込み，記述された情報を元に docker command を生成し，送信する．

docker コンテナは，コンテナ外との通信を行うために docker network に所属する．本研究での IoT システムエミュレーションでは，HTTP Server-Gateway 間および Gateway-Sensor 間で通信がある．本研究では，docker network を各 Worker 内で完結させることを原則とし，実験環境設定ファイルはノードをトップレベルとした階層形式で記述とした．また，各 Worker でのサブネットを記載するようにし，簡潔な IP アドレス設定を可能とした．他の記述方法として，各コンテナを並列に配列形式で記述する方法も考えうる．Docker Swarm では，Worker をまたいだ docker network を作成することもでき，コンテナを任意のノードに配置することも可能である．しかし，配置するエミュレータごとに稼働する Worker を指定する必要があり，設定ファイルの記述が煩雑となる．また，コンテナ実行状態の監視が困難になる可能性がある．そのため，本研究では，ノードをトップレベルとした階層形式で記述する方法をとる．

実験環境設定ファイルの記載例を図 4 に示す．各項目の概要は，以下の通りである．

- Node_name： 実行するノード名．作成する docker network の名前を兼ねる．
- subnet：docker network の IP サブネットワークアドレス．
- http_nodes：HTTP Server の IP アドレス．
- gateway_nodes：Gateway の情報．
- volume：Gateway-Sensor 間で UNIX ドメインソケット通信を行うための共有ボリューム名．
- ip：Gateway の IP アドレス．
- sensor_nodes :Sensor の情報．この階層下にグループ名をキー，Sensor 数をバリューとして配置 Sensor 数を記述する．

```
p118:
  subnet: "192.168.0.0/16"
  http_nodes:
    server: "192.168.0.2"
  gateway_nodes:
    GW-1:
      ip: "192.168.0.100"
      volume: "p118sock1"
      sensor_nodes:
        GroupA: 3
        GroupB: 2
    GW-2:
      ip: "192.168.1.100"
      volume: "p118sock2"
      sensor_nodes:
        GroupC: 3
        GroupD: 1
p120:
  subnet: "192.169.0.0/16"
  http_nodes:
    server: "192.169.0.2"
  gateway_nodes:
    GW-3:
      ip: "192.169.0.100"
      volume: "p120sock"
      sensor_nodes:
        GroupE: 1
        GroupF: 2
```

図 4　実験環境設定ファイルの記述例．

4. 実験環境構築機能の実装

前節で記述したとおり，コンピュータのクラスタリングの実装には Docker Swarm を利用する．Docker Swarm では，Manager から Worker に対してネットワーク経由でサービスの起動命令を送信できる．Worker は，クラスタのメンバーに参入する際，Manager から送られてくる起動命令を受け付けるためのリッスンアドレスおよびポートを指定する．Manager は，このアドレスおよびポートを用い，クラスタに参加した複数の Worker に対して起動・停止命令を送信する．

Manager に配置するコントロールスクリプトは Python で実装した．コントロールスクリプトは，実行時に YAML 形式の実験環境設定ファイルを読み込む．ファイル読み込みには Python のライブラリである PyYAML ライブラリ [8] を用い，コントロールスクリプト内では dictionary 型の変数として扱う．読み取った実験環境設定ファイルの情報を元に，実験環境構築のための docker command を作成する．作成した docker command は，Python の Subprocess 関数によって Manager ノードのシェルが実行する．各 Worker は docker command に応じてコンテナを起動する．起動コンテナが多い場合，単一ノードで全てのコンテナを実行すると処理仕切れない可能性がある．そのため，設定ファイルベースでの分散処理を行った．実験環境設定ファイルの Network_name が実行 Worker の指定も兼ねており，実行ノードを決定している．Manager から Worker に対しての docker command は逐次処理され，Worker の task 完了を監視する update 処理も実行される．

5. 性能評価

実験環境構築機能の性能を評価するため，実験環境構築にかかる時間を計測し，実験環境規模に応じた構築時間の変化と分散実行の効果を調べた．コントロールスクリプトで実行する docker command の実行開始から

表 1　計測を実施した StarBED ノードの仕様

筐体	Dell PowerEdge R430
CPU	Intel Xeon E5-2683 v4 (2.1GHz，16core) × 2
メモリ	384GB
OS	Ubuntu 18.04 Server AMD64

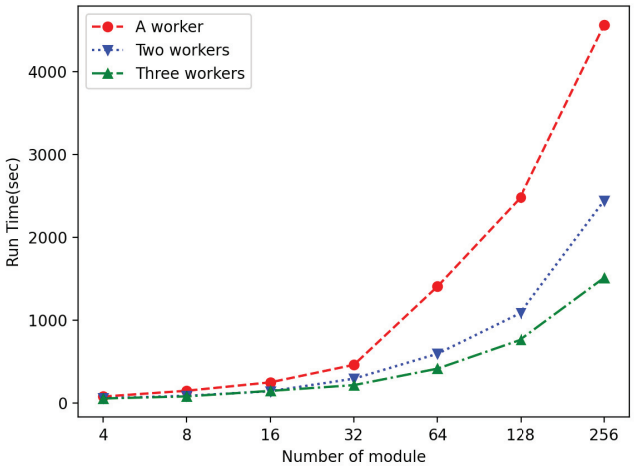

図 5　実験環境構築時間の計測結果

終了までの時間を構築時間とみなして計測した．HTTP Server:Gateway:Sensor の数の比を 1:n:4n とし，Sensor 数は 4，8，16，32，64，128，256 とした．Worker 数は 1，2，3 とした．計測には，表 1 に示す StarBED ノードを 4 台用い，1 台が Manage，3 台が Worker を担った．

計測結果を図 5 に示す．Worker を増やし環境を分散することで，構築時間が短縮されることを予想した．Sensor 数 64 以上では，Worker 数と構築時間は反比例し，Worker の数に応じて起動時間が短縮していることがわかる．一方，Sensor 数 16 以下では Worker 数 2 と 3 の場合であまり変わらなかった．Sensor 数 32 でも Worker 数 3 の実行時間が短縮したとは言い難い．これは起動した Sensor 数が少ないためである．特定の Worker が負担する処理が多くなり，Worker 数 3 における Worker の最長の実行時間が，Worker 数 1 とあまり変わらなかったと考えられる．本評価では，Sensor 数を StarBED ノードの CPU スレッド数より多い場合の計測も行ったが，その場合にも構築時間の大幅な増加は見られなかった．より大規模な実験環境を構築する場合に Sensor 数がさらに増え 1 スレッドの処理がビジーとなることが予想される．その際には，分散実行の効果がより顕著になると考えられる．

6. まとめ

本研究では，エミュレータを用いた大規模 IoT システムテストのための実験環境構築機能を開発した．各エミュレータは，Docker コンテナ上で起動する．複数の StarBED ノードを活用して実験環境を構築するために，Docker Swarm を用いてクラスタリングを実現した．実験環境の構成は，実験環境設定ファイルに記述する．この実験環境構築機能によって，実験実施者は煩雑な設定をすること無く実験環境を構築できる．また，複数ノードで処理を分散する環境構築も容易に行うことができ，単一ノードで実験を行うよりも効率的な実験が実施できる．

実験環境構築機能の性能評価のため，実験環境構築にかかる時間を計測し，実験環境規模に応じた構築時間の変化と分散実行の効果を調べた．計測結果は，モジュール数が少ない場合は分散処理の効果が小さく，モジュール数が多い場合には想定通りの分散処理の効果が現れることを示した．

本論文の計測では，StarBED ノード 4 台を用いた．2020 年現在，StarBED は合計 464 台のノードを有す．これらのすべてを独占して使用することはできないが，より多くのノードを利用することにより，より大規模な IoT システムの実験環境を構築することも可能である．

本研究の今後の展望として，実験実行時のログ取得機能の追加や，実験後の環境管理の支援機能の追加などが挙げられる．これらは，本論文での課題と同様に，煩雑な作業である．また，別の研究として，エミュレータを実行するための統合開発環境の開発も進められており [9]，本研究との統合も検討している．今後，これらの実施を進め，より高度な実験支援機能の実現を目指す．

参考文献

[1] 北陸 StarBED 技術センター：StarBED4 プロジェクトウェブサイト，情報通信研究機構（オンライン），入手先 ⟨http://starbed.nict.go.jp/⟩（参照 2020-09-19）．
[2] Miyachi, T., Nakagawa, T., Chinen, K.-i., Miwa, S. and Shinoda, Y.: StarBED and SpringOS architectures and their performance, *International Conference on Testbeds and Research Infrastructures*, Springer, pp. 43–58 (2011).
[3] Barry, R. et al.: FreeRTOS, *Internet, Oct* (2008).
[4] Nakata, J., Miyachi, T., Beuran, R., Chinen, K.-i., Uda, S., Masui, K., Tan, Y. and Shinoda, Y.: Starbed2: Large-scale, realistic and real-time testbed for ubiquitous networks, *2007 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, IEEE, pp. 1–7 (2007).
[5] Akashi, K., Inoue, T., Yasuda, S., Takano, Y. and Shinoda, Y.: NETorium: high-fidelity scalable wireless network emulator, *Proceedings of the 12th Asian Internet Engineering Conference*, pp. 25–32 (2016).
[6] Yasuda, S., Miura, R., Ohta, S., Takano, Y. and Miyachi, T.: Alfons: A mimetic network environment construction system, *International Conference on Testbeds and Research Infrastructures*, Springer, pp. 59–69 (2016).
[7] Merkel, D.: Docker: lightweight linux containers for consistent development and deployment, *Linux journal*, Vol. 2014, No. 239, p. 2 (2014).
[8] Simonov, K.: PyYAML (2014).
[9] 竹村太一，湯村翼：Tsumikiot: IoT デバイスエミュレータのための統合開発環境，電子情報通信学会 2020 年総合大会 (2020).