Vol.2018-ARC-230 No.27
Vol.2018-SLDM-183 No.27
Vol.2018-EMB-47 No.27
2018/3/8

ROSノード軽量実行環境mROSにおけるデバイス内のノード間通信手法

森 智也[1,a)]　高瀬 英希[1]　高木 一義[1]　高木 直史[1]

概要：我々は，Linuxを搭載できない消費電力の小さな組込みデバイスのためのROSノード軽量実行環境であるmROSの開発に取り組んでいる．本研究では，mROSにおけるデバイス内のノード間通信を高速化する手法を提案する．従来のTCPソケットを使用したデバイス内ノード間通信は効率が悪いため，タスク間で共有メモリを介したデータ通信を実現する．mROS内のノード間通信を効率化することで，エッジデバイスにおける処理の高速化が実現できる．提案手法をmROSに実装し，ノード間通信時間について評価することで，提案手法の有効性を示した．さらに本稿では，分散ロボットシステムの開発事例を示し，mROSの有用性を考察した．

1. はじめに

近年，社会生活を支援することを目的とした様々な場面でモバイルロボットの需要が高まっている．これらのロボットはこれまで開発されてきた産業用ロボットとは異なり，外部電源からのエネルギー供給ではなく，内部電源のエネルギーによって動作する．また，高度かつ多機能なサービスを要求されることが多く，一つのロボットシステムにおいて限られた電力のもとで多くの機能を実現することが求められる．そのため，ロボットが提供するサービス品質の向上には，多機能化および省電力化が必須となる．

ロボット用ソフトウェアの設計生産性の向上に資する開発支援フレームワークとしてROS（Robot Operating System）が注目されている [1]．ROSはロボットシステムのコンポーネント指向開発の実現を目指している．ソフトウェア部品はノードとして表現され，複数のノードを組合せることで所望のロボットシステムが実現される．ROSはノード間における通信層を提供するミドルウェアでもあり，ノード間ではトピックを介して送受信するデータを識別する出版購読モデルによって通信が行われる．ノード間通信は，TCP/IPプロトコルを使用して行われるが，デバイス内のノード間で通信を行う場合，TCPソケットを介した通信は効率が悪い．これまでに広く活用されているROS1はLinux/Ubuntu上での動作を想定した実装のみが提供されている．このため，Linuxが動作する高機能かつ消費電力の大きなデバイスを採用する必要がある．ROSを用いたロボットシステムは省電力化の実現が困難となる．

そこで我々は，組込みデバイス向けROSノード実行環境であるmROSを開発している [2][3]．リアルタイムOSであるTOPPERS/ASPカーネル，および，TCP/IPプロトコルスタックのlwIPを使用して，ROSノードの実行を可能にする組込みデバイス用通信ライブラリを提供している．mROSを活用することで，ホストデバイスおよびエッジデバイスからなる分散型ROSシステムにおいて，エッジデバイスの省電力化を実現できることが期待できる．

本研究では，デバイス内におけるノード間の効率的な通信方式を提案する．3種類の実現方式を検討し，デバイス内のノード間では共有メモリを用いてデータ通信を行う方式を採用してこれをmROSに実装する．高速なノード間通信を実現することで，エッジデバイスで行う処理を効率化することができる．エッジデバイス上で複数のノードを実行してデータ加工を行うことで，ホストデバイスとの通信量を削減することが期待できる．

さらに，mROSの活用事例として，カメラ画像から特徴点抽出を行う分散システムを開発する．既存のパッケージで構成されるシステム内で実行されるROSノードを，mROSを搭載した組込みデバイス上に移植することで，エッジコンピューティングによる処理分散化が実現される．mROS環境上で複数のノードを実行した場合について，提案するデバイス内のノード間通信方式の効果について考察を行う．

2. mROS

mROSは組込みデバイス向けROSノード軽量実行環境

1 京都大学
a) emb@lab3.kuis.kyoto-u.ac.jp

© 2018 Information Processing Society of Japan

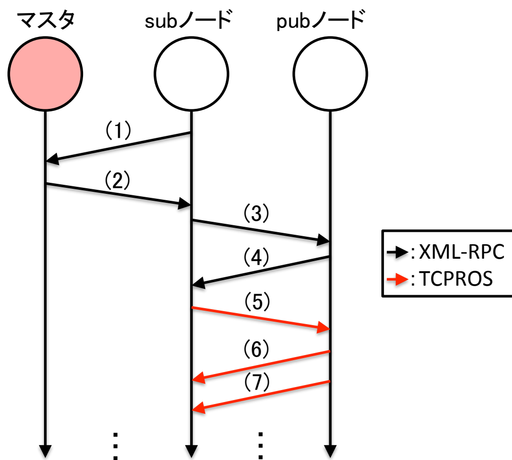


図 1: ROS における出版購読通信フロー

である．Linux を搭載できないミッドレンジクラスの組込みデバイスを対象として，ロボットシステムの省電力化，およびリアルタイム性を保証することを目標としている．ROS マスタが実行されているホストデバイスと ROS ノードが実行されているエッジデバイスからなる分散 ROS システムを想定し，組込みデバイスをエッジデバイスとして採用することで，システム全体の省電力化が実現できる．

mROS では，組込みシステム向けの TCP/IP プロトコルスタックおよびリアルタイム OS を使用して，mROS 通信ライブラリを提供している．アプリケーションは，mROS 通信ライブラリが提供する API を使用することで，**図 1** に示した通信フローで ROS システムとの通信が可能になる．これにより，組込みデバイス上で実行されるプログラムに ROS ノードとしての振舞いを可能にする．

mROS では，リアルタイム OS として μITRON 仕様の TOPPERS/ASP カーネルを使用し，TCP/IP プロトコルスタックには ARM mbed ライブラリに含まれる lwIP を使用している．TOPPERS/ASP カーネルは，実行単位であるタスクや排他制御資源であるセマフォおよびデータキューなどの資源を静的に生成するため，組込みシステムに適している．lwIP は組込みデバイス向けに設計された軽量 TCP/IP プロトコルスタックであり，オープンソースとして広く使用されている．mbed ライブラリは ARM プロセッサを搭載する組込みデバイスにおける，周辺ペリフェラルを含む様々なデバイスの統一的なデバイスドライバが含まれる．また，様々なライブラリがオープンソースとして利用できる．これらのソフトウェアを使用して実装を行った mROS の環境サイズは約 600KB であり，軽量となっている．

mROS 通信ライブラリが提供する API は ROS プログラミングモデルで使用されている関数と同名のものとなっている．そのため，mROS で実行するアプリケーションを ROS プログラミングモデルで記述することで，オープンソースとして公開されている ROS パッケージを活用することが可能になる．また，mbed ライブラリと TOPPERS/ASP カーネルを活用したアプリケーション設計も可能である．TOPPERS/ASP カーネルの機能を活用することで，リアルタイムシステムの設計が容易になる．

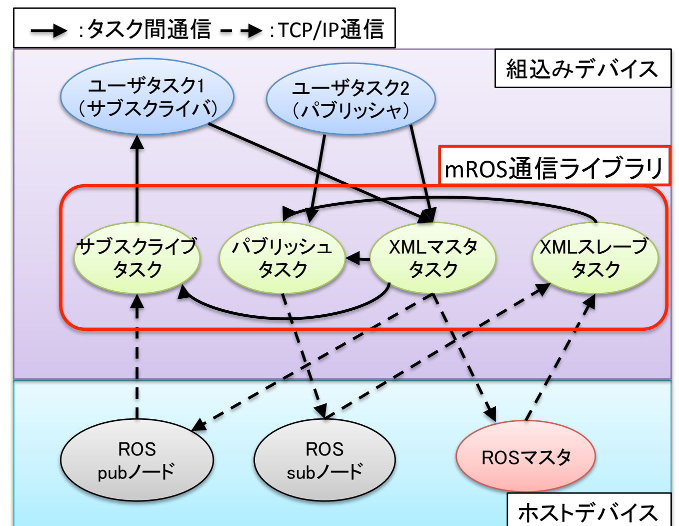


図 2: mROS におけるタスク構成とデータフロー

図 2 に mROS のタスク構成とデータフローを示す．mROS 通信ライブラリは 4 つのタスクを自動生成し，ホストデバイス上の ROS マスタおよび ROS ノードとの通信機能を実現する．大域変数としてノードリストをもち，組込みデバイス上で実行されているノードを管理する．また，サブスクライブタスクは周期的に実行されてデータを購読し，ノードリストに登録されているコールバック関数を実行する．

3. デバイス内のノード間通信方式

3.1 ROS におけるデバイス内のノード間通信方式

ROS では，ノード間のデータ通信は TCP/IP プロトコルを使用して行われる．デバイス内で実行されるノード間のデータ通信においても，TCP ソケットを介して実行されるため，通信オーバヘッドが大きくなる．この問題を解決するために，ROS では Nodelet という機能が提供されている．Nodelet は，ノード間におけるデータ通信に共有メモリを使用する．そのため，シリアライズ/デシリアライズ処理を行うことなく，メモリコピーのみによる高速なデータ通信を実現できる．

[4] で公開されている Nodelet 評価用パッケージを使用し，ROS におけるデバイス内のノード間通信時間を測定した．測定環境として，NEC の LAVIE Hybryd ZERO を使用した．CPU は Intel Core-i7 2.4GHz，メモリは 16GB を搭載している．ホストデバイスの OS は Ubuntu 14.04 LTS, ROS のディストリビューションは indigo を使用した．

2 つのノード間において，TCP ソケットを用いた通信，

© 2018 Information Processing Society of Japan

および，Nodelet を使用した通信について測定を行った．また，ROS ではスマートポインタ（`boost::shared_ptr`）を使用することで，ノード内でデータを共有する機能を提供している．それぞれの方式について，出版購読するデータを `boost::shared_ptr` を用いた参照渡しとして共有する場合と，コピーすることでデータを渡す場合について測定を行った．**表 1** にそれぞれの通信方式を使用して，3MB のデータについて 1000 回出版購読通信を行ったときの平均通信時間，および，最悪通信時間を示す．

通常の TCP ソケットを使用した通信と比べて，Nodelet を使用した通信方式は，約 4 倍高速なノード間通信を実現できている．また，データを参照渡しにする方式と組合わせると 800 倍の高速化が実現できる．このことから，デバイス内におけるノード間通信を高速化することは効果が高いと考えられる．

3.2 mROS における実現方式の検討

mROS が対象としているミッドレンジの組込みデバイスでは，搭載するマイコンの性能，および，メモリ資源は複数ノードを実行するには十分であると考えられる．例えば，センサからデータを取得するノードに加えて，そのデータを購読して処理した後にホストデバイスに出版するノードを実行することが想定される．このとき，取得したセンサ値をそのままホストデバイスに出版するよりも，データ処理によってその通信量が小さくできることがある．そのため，mROS においてデバイス内のノード間通信を効率化するデータ通信方式を提供する価値は高いと考えられる．

本節では，**図 3** に示す 3 種類の mROS におけるデバイス内のノード間通信方式を検討する．黒色の矢印は転送を行うデータのフローを示し，青色の矢印はタスク間における通知のフローを示す．メモリ管理とノードの実行を担う TOPPERS/ASP カーネルが提供する機能を使用し，共有メモリを介したノード間のデータ通信を実現する．

3.2.1 方式 1：データ購読用メモリ領域への書き込み

図 3(a) に示す方式 1 では，パブリッシュタスクとサブスクライブタスクの間に専用のメモリ領域を確保し，そこに出版データを書き込む．

(1) ユーザタスクで `publish()` が実行された時，データ出版用メモリ領域に出版データを書き込む．
(2) ユーザタスクからパブリッシュタスクへと出版通知を発行する．
(3) 出版通知を受け取ったパブリッシュタスクは，メモリからデータの読み込みを行い，TCPROS プロトコルによりデータ変換を行う．
(4) デバイス内のノード間通信である場合，パブリッシュタスクはデータ購読用に用意されたメモリにデータを書き込むことでデータ出版を行う．
(5) データを購読するサブスクライブタスクは，メモリからデータの読み込みを行う．

表 1: ROS におけるデバイス内のノード間通信時間 [us]

通信方式	平均時間	最悪時間
TCP	4661.59	9785.13
TCP + `shared_ptr`	2090.04	4421.55
Nodelet	1006.20	6088.76
Nodelet + `shared_ptr`	50.35	507.54

専用の共有メモリによって，ノード間通信を実現しており，TCPROS にエンコードされたデータを購読することとなる．データ出版があった場合にサブスクライブタスクがデータを購読する前にデータの更新が行われる可能性が低い．しかし，データ購読を実行する周期がユーザタスクによるデータ出版の周期よりも短い場合，すでに購読を行ったデータを再度購読してしまう．また，専用の共有メモリを使用するため，実行環境のサイズが大きくなる．

3.2.2 方式 2：パブリッシュタスクによるデータ購読通知

図 3(b) に示す方式 2 では，ユーザタスクからパブリッシュタスクにデータを渡すために使用する共有メモリのアドレスを，パブリッシュタスクがサブスクライブタスクに通知する方式である．

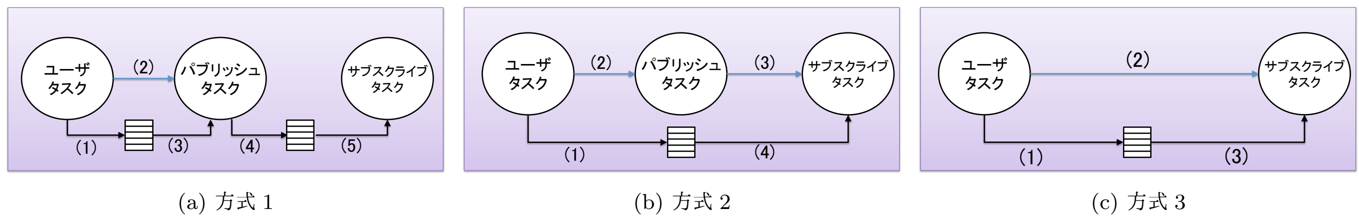
(1) ユーザタスクで `publish()` が実行された時，データ出版用メモリ領域に出版データを書き込む．
(2) ユーザタスクからパブリッシュタスクへとデータ出版通知を発行する．
(3) パブリッシュタスクは，サブスクライブタスクへデータ購読通知を発行する．このとき，データが書き込まれているアドレスを通知する．
(4) 共有メモリからデータを読み込み，コールバック関数を実行する．

前述の方式 1 と比較して，メモリコピーが 2 回に抑えられ，通信時間を小さくできる．また，使用する共有メモリ領域は，他デバイスへのデータ出版に使用される領域と共用するため，実行環境サイズが大きくならない．パブリッシュタスクによる購読通知がトリガーとなって共有メモリからのデータ購読が実行されるため，同一データを再購読することがない．しかし，購読するメモリ領域が別の出版処理によって，サブスクライブタスクが購読を行う前に更新される可能性がある．この可能性を排除するためには，メモリ資源への排他制御などを実装する必要があり，実装が複雑化する．

3.2.3 方式 3：ユーザタスクによるデータ購読通知

図 3(c) に示す方式 3 では，ユーザタスクのコンテキストにおいて，デバイス内のサブスクライバノードに対するデータ出版かどうかを判断する．

(1) ユーザタスクで `publish()` が実行された時，データ出版用メモリ領域に出版データを書き込む．
(2) ユーザタスクから，直接サブスクライブタスクへと購



(a) 方式 1 (b) 方式 2 (c) 方式 3

図 3: mROS におけるデバイス内のノード間通信方式

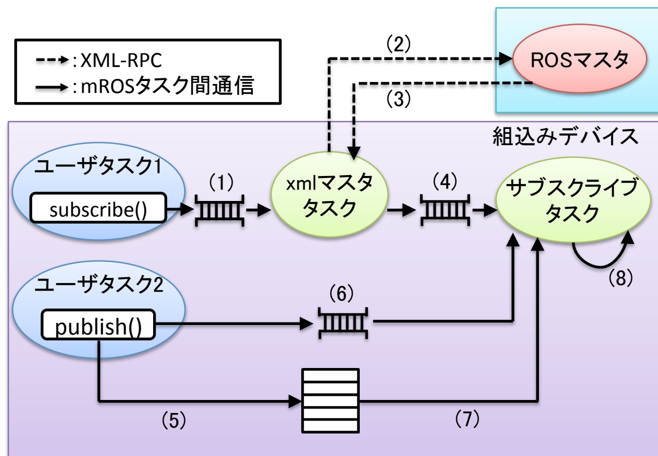


図 4: デバイス内のノード間通信実行フロー

読通知を発行する．

(3) 共有メモリからデータを読み込み，コールバック関数を実行する．

この方式では，ユーザタスクから直接サブスクライブタスクへ通知を発行するため，パブリッシュタスクを実行する時間が削減される．そのため，方式 2 と比較してより短い周期でサブスクライブタスクが実行可能になり，デバイス内のノード間通信の高速化が実現できる．方式 2 と同様に，共有メモリに書き込まれているデータが，購読前に更新される可能性がある．

3.3 採用した方式の実装

前節で検討した通信方式のうち，方式 3 が最も高速にデータ通信を実現でき，要求されるメモリ領域も小さくなると考えられる．欠点として挙げた購読前にデータが更新される可能性のある問題は，サブスクライブタスクおよびデータ出版の実行の周期を適切に設定することで解消できる．以上の理由から，本研究では方式 3 を採用して mROS に実装する．

図 4 に，デバイス内における，ノード間通信の実行フローを示す．ユーザタスク 1 がサブスクライバノードを実行し，ユーザタスク 2 がパブリッシャノードを実行しているとする．

サブスクライブタスクへのデータ購読通知はデータキューを使用して実現した．

(1) `subscribe()` が実行されると，サブスクライバノード初期登録通知が xml マスタタスクへ発行する．

(2) xml マスタタスクが XML-RPC 通信によって，ROS マスタへ登録手続きを行う．

(3) xml マスタタスクにて，サブスクライバノード登録手続きのレスポンスに URI として含まれているパブリッシャノードの IP アドレスとポート番号を取得する．

(4) パブリッシャノードの IP アドレス情報を含むサブスクライバノード初期化通知を発行する．サブスクライブタスクでは，通知された IP アドレスと自デバイスの IP アドレスとを比較することで，デバイス内通信か判断する．ノードの情報を mROS が管理するノードリストへと追加する．

(5) パブリッシャノードを実行しているユーザタスクにおいて，`publish()` が実行されると，データをメモリに書き込む．

(6) 出版するトピック名をキーとしてノードリストを検索し，トピックを購読するサブスクライバノードが存在する場合，データ購読通知を発行する．

(7) 周期的に実行され，データ購読通知を受け取ることで，共有メモリからデータの読み込みを行う．

(8) 登録されているコールバック関数を実行する．

同一のトピックを購読する外部サブスクライバノードが存在する場合は，パブリッシュタスクに対しても出版通知を発行する．

また，これらの処理は mROS 通信ライブラリが提供する API を通してバックエンドで実行される．そのため，ユーザタスクに記述されるプログラムは同一の関数を使用することが可能になっている．mROS 環境を搭載するデバイス間でのノードの移植を行うことが容易になっている．

4. 評価

4.1 評価環境

マイコンとして RZ/A1H が搭載されている GR-PEACH を対象に提案する通信方式の実装を行った．10MB の内蔵 RAM を持ち，最高動作周波数 400MHz で，5V の MicroUSB 給電で動作することが可能となっている．

評価環境の分散型 ROS システムのエッジ端末として，mROS を搭載した GR-PEACH を使用し，ホストデバイス

© 2018 Information Processing Society of Japan

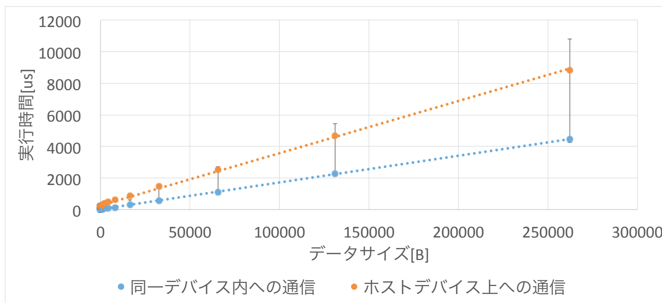


図 5: ノード間通信の実行時間

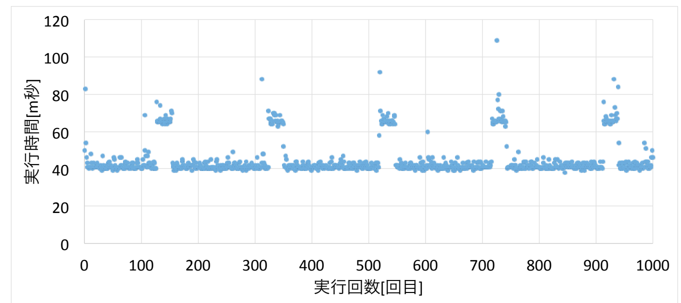


図 6: 1KB データに関するノード間通信実行時間の分散

として，NEC の LAVIE Hybryd ZERO を使用した．CPU は Intel Core-i7 2.4GHz，メモリは 16GB を搭載している．ホストデバイスの OS は Ubuntu 14.04 LTS，ROS のディストリビューションは indigo を使用した．ホストデバイスとエッジ端末は他のデバイスが存在しない同一ローカルネットワーク内に存在し，有線 LAN ケーブルでそれぞれルータと接続されている．

TOPPERS/ASP カーネルで提供される時間評価用の `get_utm()` サービスコールを使用して，データ出版処理にかかる実行時間を計測した．`publish()` の前後で `get_utm()` を実行してそれらの差を取ることでデータ通信時間とした．

4.2 デバイス内のノード間通信の評価

図 5 に 1B から 256KB までの 2 のべき乗ごとにデータサイズを変えて 200 回データ通信を行った場合の，平均実行時間，最悪実行時間および最良実行時間をエラーバーとして示す．データ出版周期は 100Hz，サブスクライブタスク実行周期を 10m 秒とした．データを購読した際に実行するコールバック関数としては，カウンタをインクリメントさせる処理を行い，実行されていることを確認した．

外部ノードに対する実行時間と比較すると，データサイズにかかわらず，データ出版にかかる実行時間が小さい．4KB 以下のデータを出版するためにかかる実行時間は 100μ 秒以下であり，外部デバイスに対する通信時間の約 5 倍高速にデータ通信が可能になっている．より大きなサイズのデータに関しても，外部デバイスに対する通信時間の半分以下の時間でデータの出版が可能になっている．また，実行時間とデータサイズには線形の相関があるため，実行時間の見積もりが容易である．しかし，データサイズによらず，最大の実行時間は平均値のおよそ 2 倍となることがわかった．そこで，データサイズを固定して，データ出版ごとの実行時間を計測することで，要因の特定を行った．

図 6 に，1KB のデータについて 1000 回データ通信を行い，それぞれに要した実行時間を示す．データ出版周期は 100Hz，サブスクライブタスク実行周期は 10m 秒とした．図 6 では，周期的に実行時間が増大している部分がみられた．実行時間が増大する周期が何に起因しているのかを調べるために，データサイズ，データ出版周期，および，タスク実行周期について比較実験を行った．その結果，タスクの実行周期に依存してデータ実行時間が増大する部分が発生していることが分かった．しかし，現時点では根本的な原因を究明するには至っていない．

また，データ出版周期をサブスクライブタスク実行周期よりも短く設定した場合のデータ出版に必要な実行時間の計測を行った．このとき，`publish()` の実行にかかる時間は，毎回 10m 秒程度の時間がかかることがわかった．そのため，デバイス内においてノード間通信が発生するシステムを構築する場合は，データ出版周期をサブスクライブタスク実行周期よりも長く設定する必要がある．

5. 活用事例

本章では，mROS を使用して分散システムを構築し，提案した通信方式の有用性について考察を行う．

5.1 活用対象の分散システム

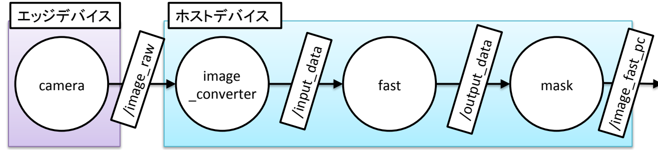
今回対象とするシステムとして，カメラから取得した画像をもとに，特徴点抽出を行う機能を実現する ROS パッケージを使用した．対象システムを構成するノードについて説明する．

- camera ノードは，カメラのデバイスドライバノードであり，画像データを取得し，出版する．
- image_converter ノードは，画像データを圧縮加工する．
- fast ノードは，画像から特徴点抽出を行う．特徴点を表示するグレイスケールの画像データ，または，元の画像データを出版する．
- mask ノードは，特徴点を示す画像と元の画像データについてマスク処理を行う．

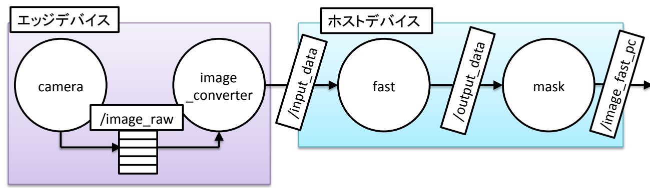
4.1 節で使用したホストデバイスとエッジデバイスを使用して分散システムを構成した．**図 7** に構築した分散システムを示す．**図 7(a)** に camera ノードのみをエッジデバイスで実行する分散システムを示す．**図 7(b)** に camera ノードおよび image_converter ノードをエッジデバイスで実行する分散システムを示す．また，エッジデバイス内のノード間通信は本研究で提案した通信方式を使用する．

エッジデバイスで camera ノードを実行するために，GR-PEACH AUDIO CAMERA シールドを使用し，mbed ラ

© 2018 Information Processing Society of Japan



(a) 単一ノードをエッジデバイス上で実行するシステム



(b) 複数ノードをエッジデバイスで実行するシステム

図 7: mROS 搭載エッジデバイスを使用した分散システム

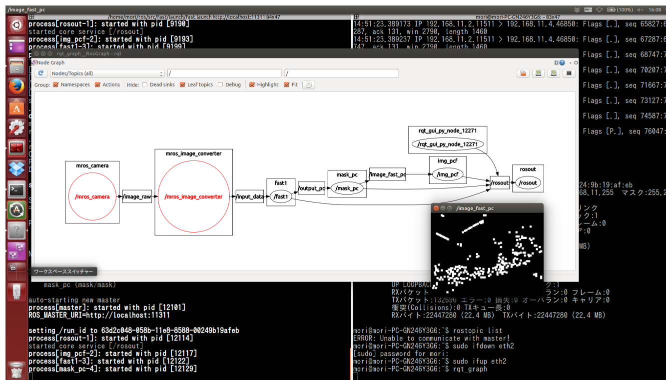


図 8: mROS を活用した分散システムの実行の様子

イブラリで提供される API を使用して実装を行った．

image_converter ノードを実装するために，GR-PEACH 向けに移植された OpenCV ライブラリを利用した．元の ROS パッケージでは，OpenCV ライブラリで扱う画像オブジェクトと ROS で扱う画像オブジェクトを相互に変換する cv_bridge ライブラリを使用している．しかし，cv_bridge ライブラリではスレッドを動的に生成する機能を使用しているため，mROS 環境に移植することができなかった．そのため，画像オブジェクトを変換するプログラムを実装することで OpenCV ライブラリによる画像圧縮を可能とした．

実際にシステムを動作させ，ROS パッケージと同様の機能が実現できていることを確認した．図 **8** に ROS のツールである rqt_graph によりシステムを可視化したグラフ，および，特徴点抽出した画像を示す．赤い丸で示されたノードが mROS で実行されているノードとなっている．

実装を行った図 7(b) のシステムのうち，エッジデバイスに書き込むアプリケーションのバイナリサイズは 3,416KB となった．内蔵 RAM を 10MB 持つ GR-PEACH に搭載するには十分軽量であるといえる．

5.2 考察

図 7(a) のシステムの場合，約 300KB のデータをホストデバイスへ出版する必要がある．図 7(b) のシステムでは，画像の圧縮処理を行うことで，約 75KB のデータを出版することになる．つまり，データ通信量を約 75%削減することができ，通信オーバヘッドが小さくなる．提案した通信方式により，ノード間の通信は高速に行える．

mROS を搭載したエッジデバイスで取得したデータを加工することで，通信オーバヘッドを小さくすることが可能になるとわかった．fast ノードをエッジデバイスで実行することにより，さらに通信オーバヘッドを削減することが期待できる．

今回のシステム構築において，既存の OpenCV ライブラリ，および，使用した image_converter のソースコードの大部分を mROS 環境で利用することができた．利用できなかった部分については，組込み用の記述に変更することでそのライブラリが提供する機能を実現することができた．このことから，一部のソースコードおよびライブラリの修正は必要であるものの，既存の ROS パッケージ資産を mROS を採用した組込みデバイス上で実行可能であることが示された．

6. まとめ

本研究では，mROS におけるデバイス内のノード間通信方式を提案した．TOPPERS/ASP カーネルの提供する機能を活用し，タスク間でデータの出版購読を通知することで，共有メモリを介するデータ転送を実現した．ノード間の通信時間を評価し，共有メモリを使用した通信方式では，TCP ソケットを介した従来のノード間の通信に比べて高速なデータ転送が実現できることを示した．

mROS の活用事例として，カメラから画像データを取得し，特徴点抽出を行う ROS パッケージを分散システムとして実装できることを示した．提案したデバイス内のノード間通信方式を使用することで，エッジコンピューティングの効率化を実現することができた．データ通信量を削減し，通信オーバヘッドを小さくすることができた．

謝辞　本研究の一部は JSPS 科研費 16H02795 の助成による．本研究を進めるにあたりご意見をいただいた宇都宮大学の大川猛助教ならびに名古屋大学の松原豊助教に深く感謝いたします．

参考文献

[1] Quigley, M.,et al.: ROS: an open-source Robot Operating System, *ICRA workshop on open source software*, No. 3.2, p. 5 (2009).

[2] 森智也，高瀬英希，高本一義，高木直史：mROS: 組込みデバイス向け ROS ノード軽量実行環境，電子情報通信学会技術研究報告， Vol. 117, No. 273, pp. 221–226 (2017).

[3] 京都大学高木研究室：mROS，`https://github.com/tlk-emb/mROS`.

[4] yoneken: measure_message_passing, `https://github.com/yoneken/measure_message_passing`.

© 2018 Information Processing Society of Japan