卒業論文

論文タイトルは４６文字で２行に収まります１２３４５６７８９０１２３４５６７８９０１２３４５６

公立はこだて未来大学
システム情報科学部　情報アーキテクチャ学科
知能システムコース　1020259

姓 名

指導教員　姓 名

提出日　2024 年 1 月 25 日

BA Thesis

Title in English within two lines: Lorem Ipsum Dolor Sit Amet, Consectetur Adipiscing Elit, Sed Do

by

Firstname Lastname

Information Systems Course, Department of Media Architecture
School of Systems Information Science, Future University Hakodate

Supervisor:　Firstname Lastname

Submitted on January 25th, 2024

Abstract–

(Abstract should be about 150–200 words. Following is a sample text.) Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna. Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci. Aenean nec lorem. In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna. Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci. Aenean nec lorem. In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy.

Keywords: Keyword1, Keyword2, Keyword3

概 要： ロボットソフトウェア開発において，ROS の利用が増加している．ROS を活用したロボットソフトウェアの構築ではクラウドを用いた分散ロボットシステムが注目されているが，ノード配置の柔軟性が課題となっている．特に，クラウドとロボットの CPU アーキテクチャの違いから，動的なノード再配置が難しい．先行研究での動的配置機構はオーバーヘッドの増加が問題とされ，mROS 2-POSIX を用いたアプローチも提案されているが，性能評価はネットワークスループットのマイクロベンチマークに留まり，実アプリケーションにおける有用性が十分に評価されていない．本研究では，mROS 2-POSIX と ROS 2 の性能を比較評価する．実験結果によって得た通信性能やメモリサイズをもとに，mROS 2-POSIX が複雑なアプリケーション上でも期待される性能を発揮できることを示す．

キーワード： クラウドロボティクス,Robot Operating System,WebAssembly

目次

第 1 章

はじめに

さまざまな産業向けやエンターテイメント関連のロボットシステム，自動車の自動運転技術や IoT システムのソフトウェア開発をサポートするフレームワークとして Robot Operating System（以下，ROS）の普及が増加している [1]．ROS のプログラミングモデルは，システムの各機能を独立したプログラムモジュール（ノード）として設計することにより，汎用性と再利用性を向上させて，各機能モジュール間のデータ交換を規定することで，効率的かつ柔軟なシステム構築を可能にしている．たとえば，カメラを操作して周囲の環境を撮影するノード，画像からオブジェクトを識別するノード，オブジェクトのデータをもとに動作制御を実行するノードを連携させることで，自動運転車の基本機能の一部を容易に実装できる．

ROS のプログラミングモデルは，ロボット/IoT とクラウドが協力する分散型システムにおいても有効である．ロボットシステムのソフトウェア処理は，外界の情報を取得する「センサー」，取得した情報を処理する「知能・制御系」，実際に動作するモーターなどの「動力系」の 3 要素に分類できる [2]．クラウドロボティクス [3] において，主に知能・制御系のノードを高い計算能力を持つクラウドに優先して配置することで，高度な知能・制御処理の実現を促進できる．さらに，ロボットが取得した情報や状態などをクラウドに集約・保存することで，複数のロボット間での情報共有と利用を容易にする．一方で，現行の ROS 実装では，各ノードの配置をシステム起動時に静的に設定する必要があり，クラウドとロボット間の最適なノード配置を事前に設計する必要がある．しかし，実際の環境で動作するロボットは，ネットワークの状況やバッテリー残量の変動など，システム運用前に予測することが困難な状況変化に対応する必要があり，設定したクラウドとロボット間のノード配置が最適でなくなる可能性がある．このような状況変化への対応として，ノードを動的に再配置するライブマイグレーション技術があるが，多くの場合でクラウドとロボット間の CPU アーキテクチャが異なり，命令セットがそれぞれ違うため，実行中のノードをシステム運用中にマイグレーションすることは技術的に困難である．

菅らは，WebAssembly（以下，Wasm）を用いることで，クラウドとロボット間での実行状態を含む稼働中ノードの動的なマイグレーションする手法を実現した [4]．Wasm とは

Web 上で高速にプログラムを実行するために設計された仮想命令セットアーキテクチャのことで，1 つのバイナリが複数のアーキテクチャで動作するため，異種デバイス間でのマイグレーションに適しているといえる．課題として，ROS 2 を Wasm 化したことでライブマイグレーション後のファイルサイズのオーバーヘッドが増大し，ノードの実行時間が大幅に増えてしまう問題が残った．柿本ら [5] は，組込みデバイス向けの軽量な ROS 2 ランタイム実装である mROS 2-POSIX を採用し，ROS ランタイムの Wasm 化にともなうオーバヘッド増加に対処した．しかし，採用された mROS 2-POSIX 上で指定されたメッセージを往復させるシンプルなアプリケーション上でしか評価実験はされていない [6]．そのため，ライブマイグレーション後のオーバーヘッド増加を解決するロボットソフトウェア基盤として，アプリケーションが複雑化した場合の動作が明らかでない．

本研究では，クラウドとロボット間での実行状態を含む稼働中ノードの動的マイグレーションの実現に向けた mROS 2-POSIX の性能評価を行い，mROS 2 が ROS 2 と比べて動的配置機構ロボットソフトウェア基盤としてどの点で優位性があるのか明らかにすることを目指す．

第 2 章

mROS 2-POSIX と ROS 2

2.1　ROS 2

ROS 2 は，ROS の後継であり，ROS 2 は ROS 1 と比べて，分散型のロボットシステムに対応している．ROS 1 では，主に UDP を使用したメッセージ型の通信を行っていたが，ROS 2 では，DDS と呼ばれる通信ミドルウェアを採用しており，RTPS（Real Time Publish Subscribe）プロトコルを用いたメッセージ型通信を行っている．これによって，個々のロボットが独立して動作するだけでなく，複数のロボットが協調して動作することが可能になった．

2.2　mROS 2-POSIX

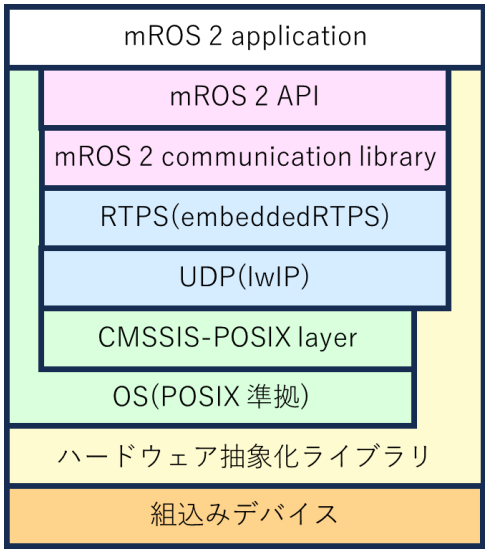

図 2.1: mROS 2-POSIX の内部構成

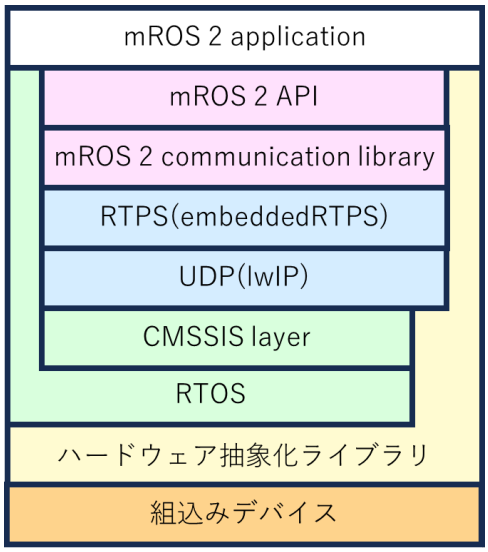

図 2.2: mROS 2 の内部構成

mROS 2 は，ROS 2 ノードの軽量実行環境である．このロボットソフトウェア基盤によって，分散型のロボットシステムへの組込み技術の導入ができる．組込みデバイスは計算資源が限定的であるが，mROS 2 を導入することによって組込みデバイスのリアルタイム性の向上および消費電力の削減ができる．そして，mROS 2 が POSIX[7] に対応したのが mROS 2-POSIX である．

図 1（a）は，mROS 2-POSIX のソフトウェア構成を示す．mROS 2-POSIX アプリケーション層は，ユーザが実装する ROS 2 ノードに相当する．mROS 2-POSIX API 層および通信ライブラリ層は，メッセージを非同期にパブリッシュやサブスクライブするためのコミュニケーションチャネルである ROS 2 の Topic に相当する API および通信機能を提供する階層である．本階層は，ROS 2 のネイティブなクライアント通信ライブラリである rclcpp と互換性を保つように設計されている．mROS 2 通信ライブラリでは，rclcpp のうち pub/sub 通信の基本的な機能のみ実装されている．利用可能な機能は制限されているものの，組込み技術を導入する ROS 2 開発者は，汎用 OS 向けのプログラミングスタイルを踏襲しながら C++ によって mROS 2 のアプリケーションを実装できる．

Real Time Publish-Subscribe（以下，RTPS）プロトコルスタックには UDP でパブリッシャとサブスクライバ C++ 実装の embeddedRTPS[8] が採用されている．UDP については組込み向けの C 実装である lwIP が採用されている．通信層の embeddedRTPS および lwIP は CMSAIS-POSIX に依存しており，図 1（b）に示す mROS 2 の CMSIS-RTOS を互換した層になっている．最下層にはハードウェアを抽象化したライブラリがある．

mROS 2-POSIX は図 2 に示す実行方式を採用している．リアルタイム OS では，組込みマイコンを実行資源の管理対象として，タスク単位でアプリケーションが実行される．POSIX においてはタスクに相当する概念はプロセスであり，そこから生成されるスレッドを実行単位として処理が進行している．しかし，mROS 2-POSIX は実行単位であるノードに POSIX のスレッドを対応づけ，組込みマイコンでの通信処理におけるイベント割込みについては，POSIX 準拠 OS におけるブロッキング API の発行に相当させて処理している．

第 3 章

評価方針

mROS 2-POSIX と ROS 2 の性能を比較評価するにあたって，mROS 2-POSIX に実装できるアプリケーションと同様のアプリケーションを ROS 2 に実装し，比較評価する．本章では，比較評価に用いるアプリケーションの選定基準と，比較評価に用いるアプリケーションの詳細について述べる．

3.1 アプリケーションの選定基準

本研究では，mROS 2-POSIX と ROS 2 の性能を比較評価するにあたって，以下の基準を設けた．

- Pub-Sub 通信のみを使用したアプリケーションであること
- 組み込みデバイス上で動作できるアプリケーションであること
- エンドツーエンドのアプリケーションであること（ユーザー入力からロボットの動作まで）

3.1.1 Pub-Sub 通信のみを使用したアプリケーションであること

mROS 2-POSIX は，embeddedRTPS を利用して RTPS を実装している都合上，ROS 2 の Pub-Sub 通信のみの実装となっている．そのため，評価に用いるアプリケーションは Pub-Sub 通信のみを使用したアプリケーションである必要があり，この条件を満たすことで，mROS 2-POSIX と ROS 2 の通信性能を比較評価できる．

3.1.2 組み込みデバイス上で動作できるアプリケーションであること

先行研究での動的配置機構はオーバーヘッドの増加が問題とされ，mROS 2-POSIX を用いたアプローチも提案されているが，性能評価はネットワークスループットのマイクロベン

チマークに留まり，実アプリケーションにおける有用性が十分に評価されていない．そのため，評価アプリケーションの実装条件として，動的配置機構が実現される組み込みデバイス上で動作するアプリケーションでなくてはならない

3.1.3 エンドツーエンドのアプリケーションであること

ユーザー入力によってロボットが動作するアプリケーションを評価に用いることで，より実アプリケーションに近い状況での評価を行うことができるため，エンドツーエンドを条件の一つとした．

第 4 章

実装

本章では，mROS 2-POSIX と ROS 2 の性能を比較評価するにあたって，mROS 2-POSIX と ROS 2 に実装するアプリケーションの概要と，これまで実装予定だったものについて説明する．

4.1　実装アプリケーションの概要

章で述べたように，実装するアプリケーションの条件として，Pub-Sub 通信のみを使用したアプリケーションであること，組み込みデバイス上で動作できるアプリケーションであること，エンドツーエンドのアプリケーションであることを設けた．これらの条件を満たすアプリケーションとして，Wii Fit Board と Raspimouse を用いたロボットの動作を制御するアプリケーションを実装する．Raspimouse とは，ロボット開発キットである Raspberry Pi Mouse のことで，Raspberry Pi 上で動作する ROS 2 のノードで制御することができる．Wii Fit Board は，任天堂が発売した Wii の周辺機器で，体重移動を検知することができる．このアプリケーションは，Wii Fit Board からのセンサデータを Raspimouse に Pub-Sub 通信を用いて送信し，Raspimouse は受信したセンサデータをもとに動作を制御する．ユーザの体重移動によってロボットが動作するため，エンドツーエンドのアプリケーションである．また，Raspimouse 上でノードを立ち上げ動作しているため，組み込みデバイス上で動作するアプリケーションという条件も満たしている．

4.2　実装予定だったアプリケーション

評価実験に用いるアプリケーションとして，Wii Fit Board と Raspimouse を用いたロボットの動作を制御するアプリケーションを実装するが，本来は，Wii Fit Board からのセンサデータを Sphero sprk + に Pub-Sub 通信を用いて送信し，Sphero sprk + は受信したセンサデータをもとに動作を制御するアプリケーションを実装する予定だった．Sphero sprk

+ は，Sphero 社が発売した球体のロボットで，ユーザーがスマートフォンから操作することができる．ROS 2 側では Wii Fit Board からのセンサデータを受け取り，Sphero sprk + に送信する 2 つのノードを実装することに成功したが，mROS 2-POSIX 側では Wii Fit Board からセンサデータを送信するノードを完成させることができたものの，Sphero sprk + を受信したセンサデータをもとに動作させるノードの作成が難しく，実装を断念した．実装を断念した理由として，ROS 2 側の実装では Sphero sprk + は Python のライブラリを用いて制御しており，mROS 2-POSIX は Python の Support はないため，動作させるには，Python のライブラリを C++ 言語に変換する必要があった．しかし，Python のライブラリを C++ 言語に変換するためには，bluez を用いて Sphero sprk + を BLE サーバーと見立てたクライアントのスクリプトを実装する必要があり，その実装が間に合わないと判断したため，実装を断念した．また，Sphero sprk + を動作させるライブラリは go 言語にもあったため，cgo と呼ばれる go 言語から C 言語の関数を呼び出すための仕組みを用いて，go 言語から Sphero sprk + を動作させることも試みたが，mROS 2-POSIX の cgo 移植になってしまい，実装範囲が広くなってしまったため，実装を断念した．

4.3 実装アプリケーションの詳細

実装アプリケーションは ROS 2 と mROS 2-POSIX の 2 つの環境で動作する．ROS 2 側のアプリケーションは, C++ 言語で Wii Fit Board のセンサの値をパブリッシュするノードを実装した．センサの値をパブリッシュするためには，Wii Fit Board を端末と Bluetooth 接続する必要がある．接続後は/dev にある/event ○○を開くことで Wii Fit Board のセンサ値を取得できる．しかし，/event はネットワーク環境によってランダムな/event 番号に振り分けられるため，実行するネットワーク環境に変更があるたびに修正しなくてはならない．また，ノード実行前に/dev/uinput に chmod で a+rw の権限を与えておく必要がある．Wii Fit Board はそれぞれの角に 4 つのセンサがあり，センサの値を/event から 4 つの値が取得できる．この値をこのままパブリッシュすると，サブスクライブする側で 4 つの値を処理しなくてはならないので，この 4 つの値を x,y の 2 つの値に変換する．x,y はユーザの重心点を表しており，Board に乗っているユーザの体重に応じてその大きさが変化する．次に，x,y の値をサブスクライブするノードではパブリッシュされた x,y の値を受け取り，x,y の値に応じて Raspimouse の動作を制御する処理を実装した．x,y の値はそれぞれユーザの体重によって大きく増減するため，適切な閾値の値はユーザーによって変化する．今回の場合は，評価実験するユーザーとして開発者のみになるため，開発者の体重に合わせた閾値の値を設定した．ロボットが動作する処理は，x,y の値が閾値を超えた場合に Raspimouse が動作するように実装した．ユーザーがロボットを前進させようとした時，y の値がマイナスに大きく傾くため，y の値がマイナスの閾値を超えた場合に Raspimouse が前進するように実装した．同様に，y の値がプラスに傾くとき，ロボットを後退させ，x の値がマイナスに

傾くときにロボットを右に移動させ，プラスに傾くときにロボットを左に移動させるように実装を行った．ロボットを移動させる処理はロボットのデバイスドライバを直接 write 命令を用いてたたくことで実装を行っている．mROS 2-POSIX 側のアプリケーションは，ROS 2 側のアプリケーションと大きく変更されていない．ROS 2 側のアプリケーションと同様に Wii Fit Board のセンサの値をパブリッシュするノードを実装し，その値をサブスクライブするノードを実装した．パブリッシュするノードでは ROS 2 側と同様の実装になっている．異なる点は ROS 2 側では rclcpp と呼ばれる ROS 2 のノードを作成するためのインターフェースを使用しているが，mROS 2 ではオリジナルのインターフェースが用意されている．

4.4 実装に際しての課題と解決方法

実装に際しての課題として Wii Fit Board の値をパブリッシュするノードとその値をサブスクライブし，Raspimouse を動作させるノードそれぞれであった．Wii Fit Board の値をパブリッシュするノードでは，mROS 2-POSIX 側のノードで実行後に Segment Fault が発生した．mROS 2-POSIX を利用していると，ビルドが通って実行時に Pub-Sub 通信の準備が完了した後，Segment Fault が発生することがある．これによって，ノードが突然動作しなくなる問題があったが，再度ビルドすることで修正することができた．Wii Fit Board のセンサの値をパブリッシュするノードからサブスクライブするノードでは，Raspimouse を動作させる部分で課題があった．Raspimouse を制御する方法として ROS 2 を利用する方法が一般的である．方法としては，Raspimouse の制御ノードを立ち上げ，制御ノードがパブリッシュしているトピックを別のノードでサブスクライブしたり，制御ノードがサブスクライブしているトピック宛に値をパブリッシュすることで Raspimouse を操作する．すでにキーボードやゲームコントローラを使って Raspimouse を操作できるノードも配布されている．この Raspimouse の操作だが ROS 2 側では操作できるノードが配布されているため，比較的容易に動作させることができた．しかし，mROS 2-POSIX 側で Raspimouse の制御ノードである ROS 2 のトピックに対してパブリッシュを動作させることができなかった．制御ノードがパブリッシュしている値のサブスクライブは問題なく mROS 2-POSIX 側で可能であった．制御ノードに対して mROS 2-POSIX からのパブリッシュが通らない理由として，QoS 設定の問題がある．

4.5 実装アプリケーション変更に伴って生じる影響

第 5 章

関連研究

第 6 章

おわりに

本研究では，WasmEdge と WAMR 間での，異種ランタイム間ライブマイグレーションの手法について提案した．Wasm の仕様からランタイム構造や実行状態を整理し，実行状態の保存機構，復元機構を作成した．また，ランタイムごとの内部構造が異なる部分は，変換機構を実現することで解決した．

本手法の課題として，シグナルを受け取ってから，命令が終了して保存処理が始まるまでにタイムラグがあることである．とくにホスト関数呼び出しは，ユーザがホスト関数として任意の関数を定義できるため，非常に長い時間のタイムラグが発生する可能性がある．これを解決するために，Wasm 外の実行状態のマイグレーションの実現を行い，ホスト関数実行中でのマイグレーションを実現を目指す．

今後の課題では，より多くの要求に対応するために，すべての Wasm ランタイムで，ライブマイグレーション機構を実現する手法を検討する．現実装では，ランタイムの実装に手を加えることで，ライブマイグレーション機構を実現したが，すべてのランタイムに対応させるには，この手法は現実的に困難である．したがって，ランタイムの実装に依存しないような手法で，すべてのランタイム間でのライブマイグレーション機構の実現を目指す．

謝辞

謝辞を記入する.

参考文献

[1] Reference1
[2] Reference2