ロボット制御システムにおけるOSS機能モジュール向けサンドボックスの実現

b1017197 瀧本恒平
指導教員：松原克弥

Implementation of a Sandbox for an OSS Function Module in Robot Control Systems

Kouhei Takimoto

概要： Open Source Software（OSS）ノードを組み合わせることによるロボットアプリケーション開発が一般化している．Robot Operating System（ROS）は，ロボットアプリケーション開発用のフレームワークであり，ロボット開発に必要な基本的な機能を，膨大な数の OSS ノードとして提供している．これらの既存のノードを利用することで，ロボットアプリケーションの開発速度や品質が向上する．一方，OSS ノードが消費する各計算資源量は，あらかじめ不明な場合が多い．消費資源量が明らかでないノードをシステムに組み込むことで，それらがシステムに想定外の負荷を与える可能性が考えられる．本研究では，OSS ノードの消費資源量を事前に見積もり，ノードが消費可能な計算資源量に適切な上限を設けることで，OSS ノードのサンドボックスを実現する．これにより，OSS ノードによる開発効率向上と，ロボット制御システムの安定性の両立を目指す．

キーワード： ROS, OSS, サンドボックス

Abstract: Robotic application development by combining Open Source Software (OSS) nodes is becoming more common. The Robot Operating System (ROS) is a framework for developing robot applications and provides the basic functions necessary for robot development many OSS nodes. By using these existing nodes, the development speed and quality of robot applications can be improved. On the other hand, the amount of computational resources consumed by each OSS node is often unknown in advance. The inclusion of nodes whose resource consumption is not known may cause unexpected load on the system. In this study, we estimate the amount of resources consumed by OSS nodes in advance and set an appropriate upper limit to the amount of computational resources that can be consumed by the nodes to achieve a sandbox of OSS nodes. This will improve both the development efficiency of OSS nodes and the stability of the robot control system.

Keywords: ROS, OSS, sandbox

1 背景

近年，様々な分野においてロボットの活用が拡がっている [1][2]．このロボットアプリケーション開発において，Robot Operating System（ROS）を用いる機会が増えている．ROS とは，ロボット開発を効率化するアプリケーションフレームワークのことであり，ノードを複数組み合わせることでシステムを構築する．図 1 で提示されているシステムのように，ノードはロボットを構成する 1 つの機能である．この ROS のコミュニティにおいて，様々なロボットに必要な共通の機能（画像処理 [3]，顔検出 [4] など）は，オープンソースソフトウェア（OSS）のノードとして提供されている．そのため，フルスクラッチで機能を実装していた従来のロボット開発に比べて，OSS ノードを用いた開発は効率的であることから，ROS は現在のロボットアプリケーション開発において重要なフレームワークのひとつとなっている．ROS には，ROS 1 と ROS 2 の 2 種類が存在している．先にリリースされた ROS 1 では，サポート OS は Ubuntu のみであったが，ROS 2 では，Ubuntu に加え Windows や OS X をサポートしている．また，ブリッジと呼ばれる ROS 2 のパッケージを用いることで，開発途中の ROS 2 においても ROS 1 で開発されていた膨大なソフトウェア資産を使用できるようになるため，今後は ROS 2 の利用拡大が予想される．そのため，本研究では ROS 2 を対象とする．

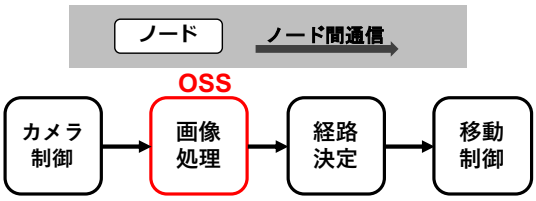

Fig. 1: ノードを用いたシステムの例

2 課題

ROS では，OSS ノードの動作がシステム内の他のノードの動作に悪影響を与える可能性がある．OSS ノードのような第 3 者が作成したノードをシステムに組み込む際，開発効率を考慮すると，ノードの動作や計算資源消費量の詳細について確認することは少ない．図 2 は，実際に OSS ノードを取り入れたロボット制御システム（ドローン）の例であり，フルスクラッチで実装した移動制御ノード，位置情報取得ノードに加え，OSS である画像処理ノードを取り入れたシステムの一例である．ここでは，OSS 画像処理ノードがシステム内の計算資源を専有してしまい，その影響を受けた移動制御ノードは必要分の計算資源を利用できない．そのため，ロボット制御システムの動作が不安定になっている．

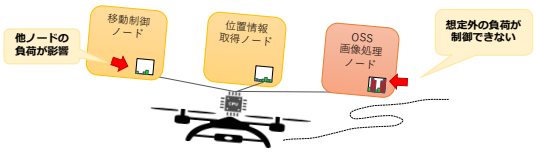

Fig. 2: ロボット制御システムの例

3 提案する手法：サンドボックスの作成

前章までで述べたとおり，ROS を用いたロボット制御システムの開発において OSS ノードを用いることには，同一システム内の他ノードの動作に悪影響を与える可能性があるという課題がある．本研究では，OSS ノードによるシステムへの想定外の負荷を防ぐことを目的として，各ノードを対象としたサンドボックスの作成を提案する．ここでのサンドボックスとは計算資源の最大消費可能量に制限をかける機構のことを指す．はじめに，サンドボックスを作成する前に，サンドボックス作成に必要な情報を得るため，シミュレータを用いてノードの計算資源消費量について見積もりを行う．次に，見積もり結果から，サンドボックスで制限する計算資源量の上限を設定し，各ノードに対応したサンドボックスを作成する．これにより，図 3 のように OSS 画像処理ノードの使用可能な計算資源量の上限がコントロールされ，移動制御ノードは必要とする分の計算資源を使用できる．そのため，不安定であったロボット制御システムの動作が安定するようになると考えられる．

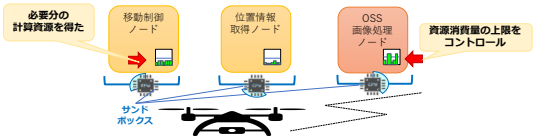

Fig. 3: サンドボックス導入後のロボット制御システムの例

3.1 手順 1：ノードの計算資源消費量見積もり手法

はじめに，ロボットを実環境で動作させる前に，サンドボックスがどの計算資源をどれほどの数値に制限するかについて決定する．そのために，各ノードが消費するおおよその計算資源量を見積もる必要がある．この見積もりは，Gazebo[5] によってロボットを仮想環境で動作させることで行う．Gazebo とは，ROS で一般的に用いられるオープンソースの 3D ロボットシミュレータである．これにより，各ノードを擬似的に動作させ，その動作中の各計算資源量を一定間隔で計測・記録する．この記録をもとに各ノードが消費する計算資源量の見積もりを行う．この計算資源消費量の計測には，Linux の procfs と，帯域幅監視ツールの一つである NetHogs を使用する．procfs とは，Linux からプロセス情報を取得するための仕組みである [6]．また，NetHogs は，プロセスごとのトラフィック量を取得するものであり，ここではネットワーク帯域幅の取得に使用する．

3.2 手順 2：Linux 機能によるノードの消費資源量制限

本研究におけるサンドボックスは，cgroup と tc コマンドによって各ノードの消費可能な計算資源量に上限を設定することで作成する．cgroup とは，ROS の動作プラットフォームである Linux 上で動作するコンテナ型仮想化環境を実現する機能の一つであり，プロセスの計算資源の利用を制限する Linux カーネルの機能である [7]．cgroup

には, 旧バージョンの cgroup v1 とその改良版の cgroup v2 が存在する．cgroup v1 では設定した計算資源量の上限を超過するとプロセスが停止してしまうが，cgroup v2 であればプロセスを停止させずに計算資源消費量を抑えることができる．そのため, 本研究では主に cgroup v2 を使用する．しかし，cgroup v2 は比較的新しい機能であるため，cgroup v1 で使用されていた機能全てを使えるわけではない．そのため, cgroup v1 と v2 を共存させる形で使用する．cgroup の操作には, 基本的に cgroupfs という擬似的なファイルシステムを用いる．新たに cgroup を作成する際も，通常のファイルシステムを扱うように mkdir コマンドを用いることが可能である．具体的に cgroup を用いてノードの計算資源消費量の制限を行うには, はじめに sys/fs/cgroup 以下に cgroup を作成し, ノードの PID を sys/fs/cgroup/cgroup.procs に登録する．その後，計算資源消費量の上限値を sys/fs/cgroup
/cpu.max 等に書き込むことで，CPU 使用率とメモリ使用率の制限を完了できる．また，tc コマンドは, 作成したネットワークインターフェースに対してネットワーク帯域制限を設定する機能である．ネットワーク帯域幅の制限についても，cgroup の net_cls を用いて，パラメータとネットワークインタフェースを設定することで制限を完了できる．これらの設定を行うことで, 本研究におけるサンドボックスを作成する．

4 関連技術

本研究の関連技術として，Docker が挙げられる．Docker とは，コンテナ型仮想化を用いてアプリケーションを実行するためのソフトウェアである．コンテナ型仮想化とは, アプリケーションの利用可能な計算資源を制限・隔離することで，仮想的な環境を低コストで実現する技術である．コンテナ型仮想化により，異なるマシン上でもアプリケーションの実行環境を統一し, 消費計算資源量を制限できる．Docker は，計算資源の制限と隔離に cgroup を用いるという点で, 本研究の手法と関係性が深い．一方，Docker には, 仮想環境を配布する機能や, 複数の仮想環境を管理する機能など，本研究とは関係のない機能が多数含まれている．したがって，本研究では，実装の容易さを考慮し，cgroup と tc コマンドを用いて，計算資源消費量の制限を行うこととした．Fukutomi らは，ROS 分散環境においてノードを動作させるホストマシン上の計算資源を効率的に利用するための計算資源管理機構を提案した [9]．この研究では，計算資源使用率が一定に達したノードを動的に他のホストマシンに移行することで，分散環境上でも効率的にノードを動作させることを可能としている．本研究とはノードの計算資源消費量を制限せずにノードの動的移行を行うという点で異なっている．Fukutomi らの実現手法は，事前にノードの消費する計算資源量を把握していなくても，ノードを動的に移行することにより，システムを短い処理時間で動作させることができるという利点がある．しかし，ノードの動的移行を行う際, ノードが保持しているデータのバックアップを取る必要があるため，システムの動作中にオーバーヘッドが発生する．そのため，本研究では事前にノードの計算資源消費量を見積もり，サンドボックスを作成することで，システム動作中のオーバーヘッドなしに効率的な計算資源の利用を可能としている．

5 まとめ

本研究では，OSS ノードによるシステムへの想定外の負荷を最小限に抑えることを目的とする．この目的を達成するために，各ノードの計算資源消費量を実行時に制限するサンドボックス機構を実装する．具体的には，Gazebo シミュレータ上でノードを動作させ, Linux の procfs と帯域幅監視ツールの一つである NetHogs を用いて計算資源消費量を記録し，これを基に計算資源消費量の見積もりを行う．見積もり結果より，コンテナ型仮想化機構の一つである cgroups と tc コマンドを用いて，ノードごとの CPU，メモリ，ネットワーク帯域の各計算資源における最大消費量に対して制限を課すことでサンドボックス機構を実現する．今後の課題として，ノードの計算資源消費量の見積もり手法の具体化がある．シミュレータ上でシステムをどれほどの時間動作させれば, 求めている計算資源消費量やノードの動作の詳細を得ることができるのかについて検討する必要がある．また，シュミレータ上でシステムを動作させることで得られる計算資源消費量の記録から，システム全体の動作に悪影響を及ぼさず，ノードが最低限動作するような計算資源量を導出し，それをもとにサン

ドボックスの設定を行う必要がある．

6 知能システムコースにおける本研究の位置づけ

知能システムコースでは，知能に関する課題および人と人工物の新たな関係性を構成論的な手法で追究する観点から，人の知的能力や機能の解明，数理モデル化，実世界への実装に関する具体的な課題に取り組み，その結果の評価を通じて，新しい方法論や，学問領域を切り拓く能力を育むことをカリキュラムポリシーとして掲げている．
本研究では，ROS という実世界への実装を補助する技術における課題を構成論的な手法で追究している．今後は実装を行い，サンドボックス導入前と導入後で各ノードの計算資源消費量がどう変化したか，またシステムの動作パフォーマンスがどう変化するかを評価する．

参考文献

[1] SoftBank：特集 | ロボット | ソフトバンク，入手先 <https://www.softbank.jp/robot/special/>（参照 2020-10-30）．

[2] SHARP CORPORATION：ロボホン，入手先 <https://robohon.com/co/introduction.php>（参照 2020-10-30）．

[3] Patrick Mihelich，James Bowman：ros-perception/image_common，入手先 <https://github.com/ros-perception/image_common>（参照 2020-11-03）．

[4] Caroline Pantofaru：ROS Index，入手先 <https://index.ros.org/p/face_detector/>（参照 2020-11-04）．

[5] 2014 OpenSource Robotics Foundation：Gazebo，入手作<http://gazebosim.org/>（参照 2020-11-04）

[6] Michael Kerrisk：proc（5）-Linux manual page，man7.org，入手先 <https://man7.org/linux/man-pages/man5/proc.5.html>（参照 2020-10-30）．

[7] Michael Kerrisk：cgroups（7）-Linux manual page，man7.org，入手先 <https://man7.org/linux/man-pages/man7/cgroups.7.html>（参照 2020-10-30）．

[8] Docker Documentation|Docker Documentation, 入手先 <https://docs.docker.com/>（参照 2020-11-01）．

[9] Fukutomi,D.，Azumi,T.，Kato,S.，et al.：Resource Manager for Scalable Performance in ROS Distributed Environments，Proc.DATE 2019，pp.1088-1093，IEEE（2019）．