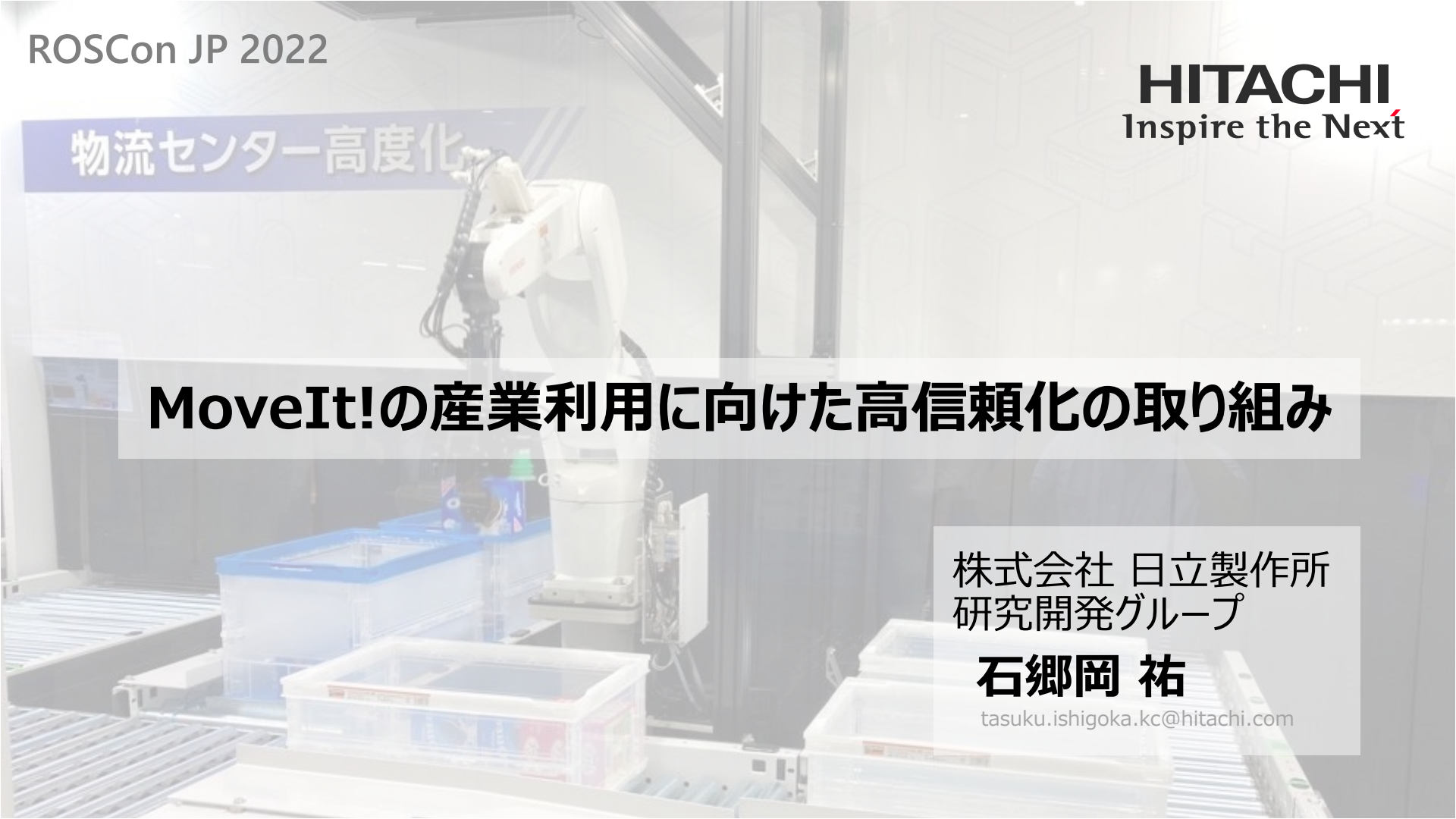
ROSCon JP 2022

HITACHI
Inspire the Next



MoveIt!の産業利用に向けた高信頼化の取り組み

株式会社 日立製作所
研究開発グループ

石郷岡 祐

tasuku.ishigoka.kc@hitachi.com

HITACHI
Inspire the Next

Contents

背景

HITACHI
Inspire the Next

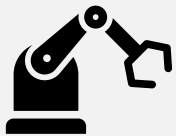
本発表では物流分野ロボット向けにMoveIt!を適用した際の高信頼化の取り組みを紹介する

■社会課題

- 少子高齢化に伴い、物流業・製造業等での労働人口不足

■解決策

- 自律型ロボットシステムの産業利用に関する検討が進んでいる



■事業課題

- 多様なユースケースに対して個別開発を実施すると、開発費用・期間の観点から実用化が困難

コミュニティで共同開発しているROSを産業で利用するアプローチが検討

ROS 2 NEDO

物流分野における例：ピッキングシステム

HITACHI
Inspire the Next

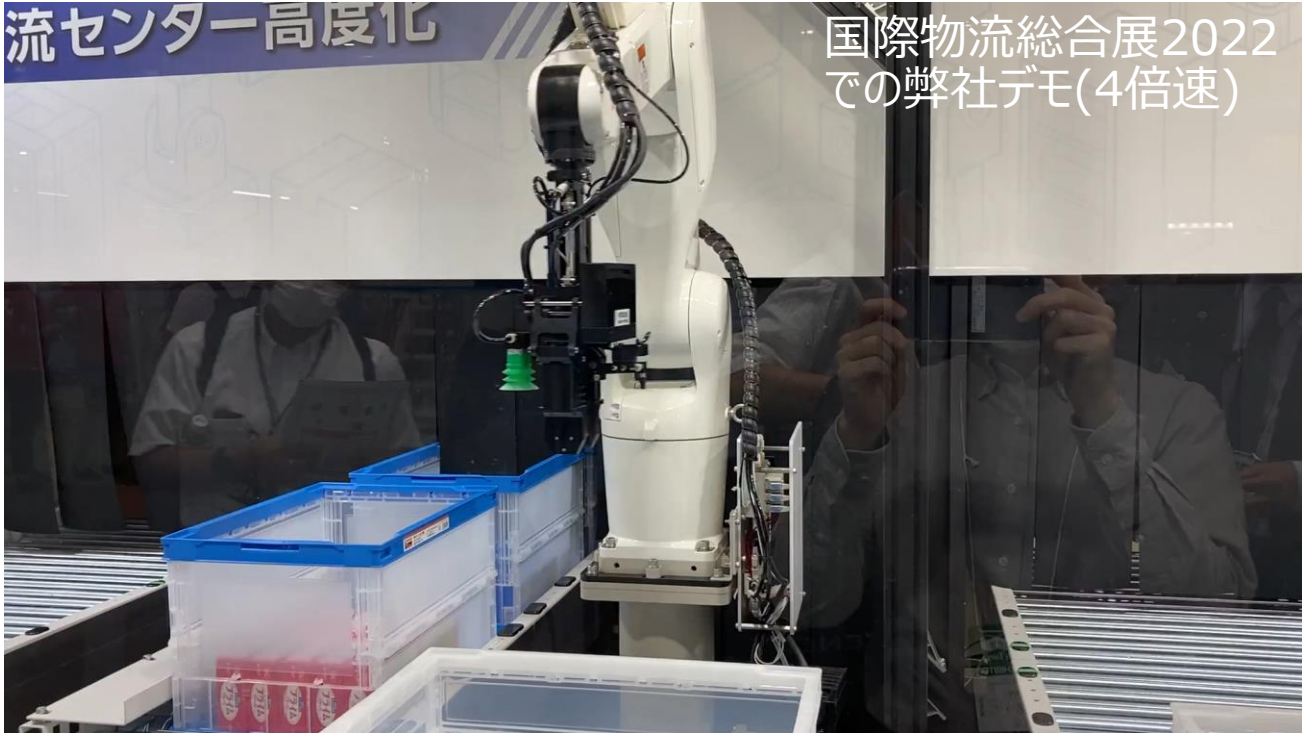
物品を目標地点に早く高信頼に運ぶことが求められる。

■背景

- 物流需要の増加
- 厳しい労働環境を背景に労働力の安定確保困難

■例

ピッキングシステムによる物流自動化



国際物流総合展2022での弊社デモ(4倍速)

物流自動化に対応 自律型ロボットシステム AIROBO　https://www.hitachi.co.jp/products/it/control_sys/airobo/

事例：意図しない急停止

HITACHI
Inspire the Next

本日はROS適用時に顕現した「意図しない急停止」の事例を用いて、
課題解決までに行った解析プロセスと解決案の1例をご紹介させて頂く。

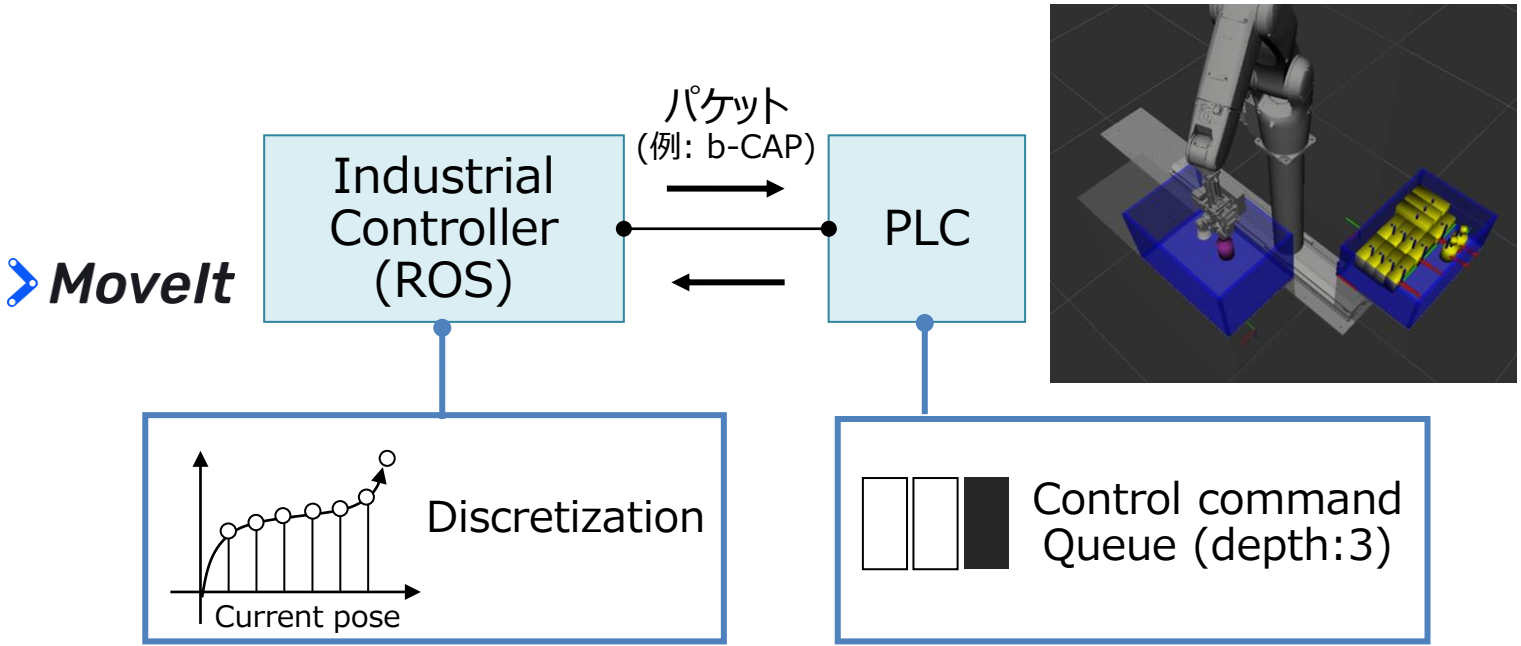
急停止発生なし
(期待値)

急停止発生あり
(課題)

物流ピッキングシステムの構成

HITACHI
Inspire the Next

周期的にPLCに軌道を送付することでアームロボットを制御する。デッドライン違反が連続発生し、キュー枯渇時にはアームが安全停止する仕組み。軌道の生成・送付にはROSのMoveIt!を利用。

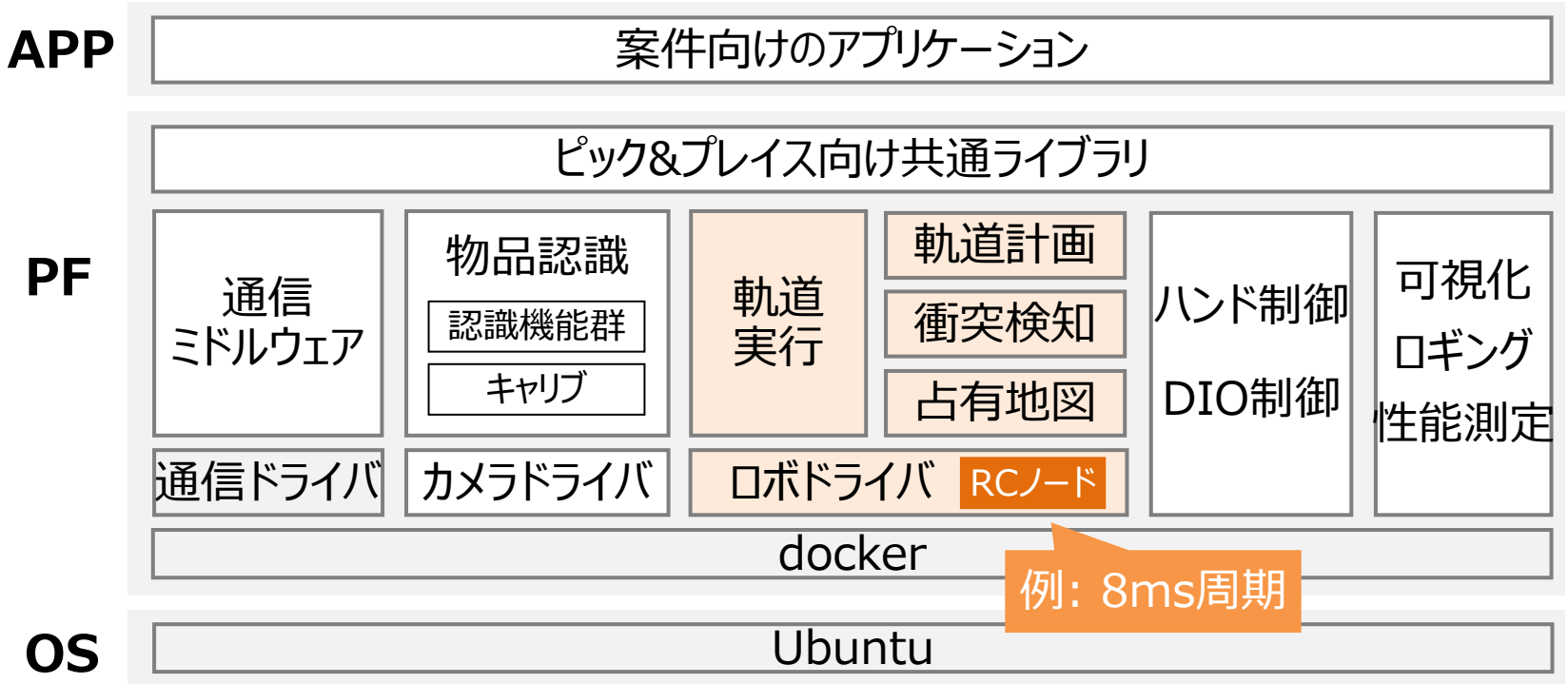


PLC: Programmable Logic Controller

ピッキングシステムのソフトウェアスタックの例

HITACHI
Inspire the Next

MoveIt!は、アームの軌道計画(move group)や、PLCに軌道を送付するRobot Controller(RC)で構成。前者はイベント駆動で動作するが、後者は短周期の動作となっている。

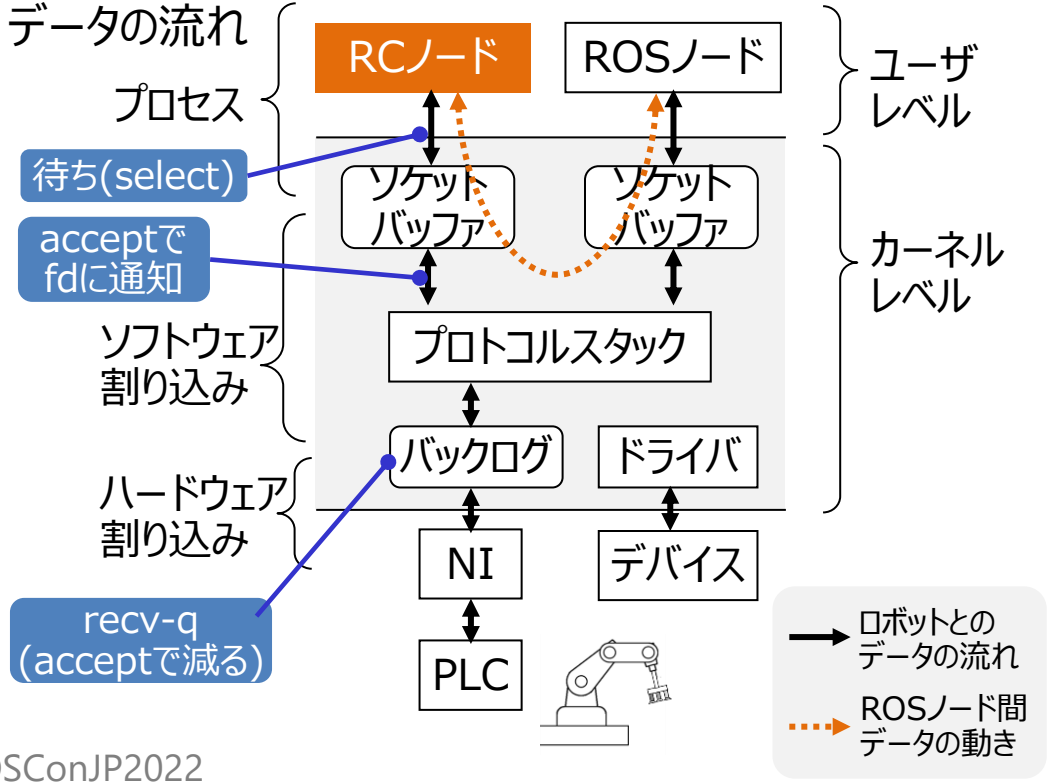


急停止事象の解析方針

HITACHI
Inspire the Next

RCとPLC間の通信がどこかで停止・遅延していることによって、急停止が発生していると予想。
通信のどこで停止・遅延が発生しているかの裏付けと、原因究明に向けてROSシステムを解析する。

■データの流れ



■解析方針

項目	確認事項	方針
現象 裏付け	待ち発生確認 (select)	OSの 状態監視
原因 究明	RCが他のプロセス によって停止・遅延 していないか	各コアの 処理状況 の監視

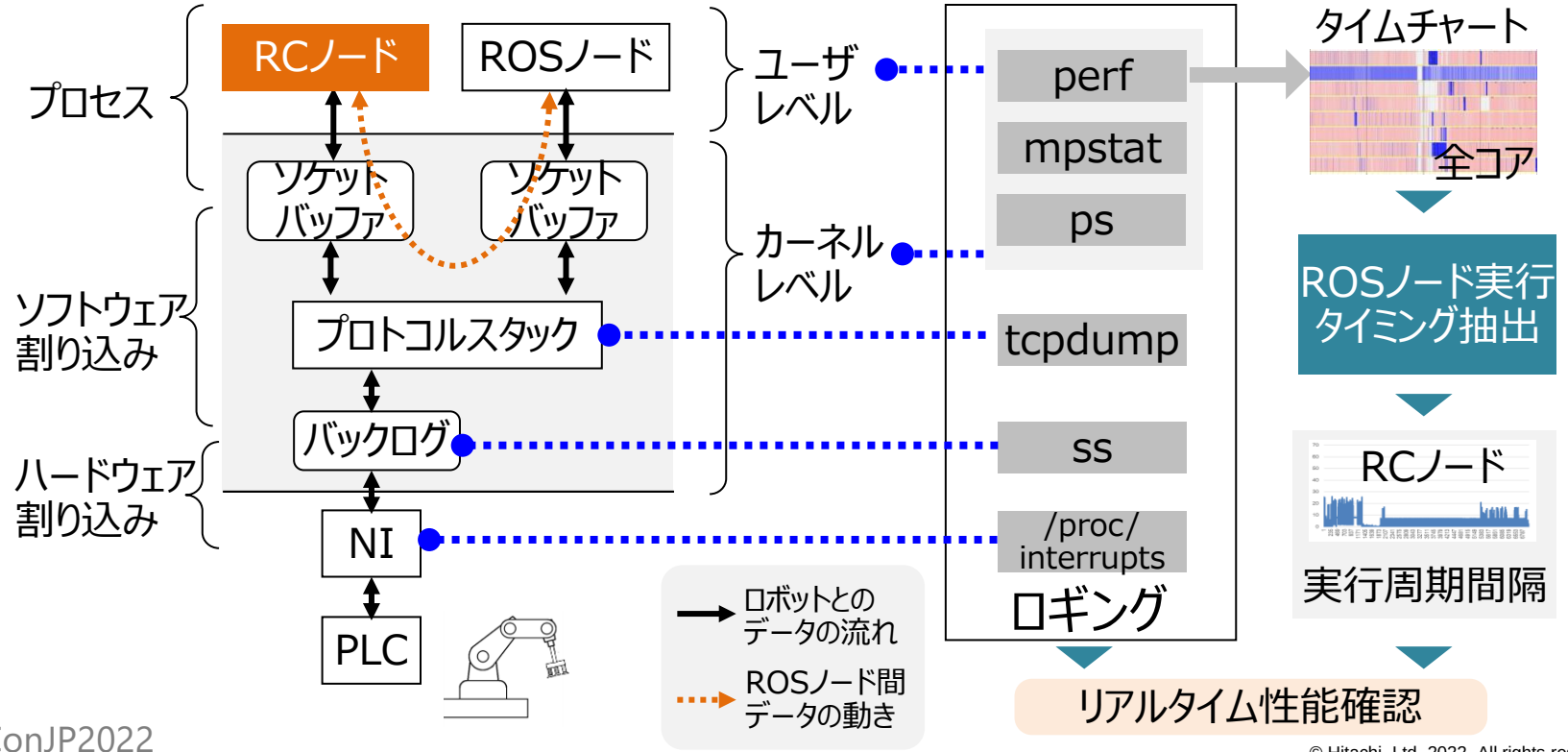
ROSとOSを統合したロギングが必要

RC: Robot Controller

解析に向けたロギングの概要

HITACHI
Inspire the Next

PLCからの受信が届いているかの確認のためにss、全体プロセスを確認するためにperfを使用。
perfのシステム全体ログからRCノードの実行周期を抽出し、リアルタイム性能を確認。



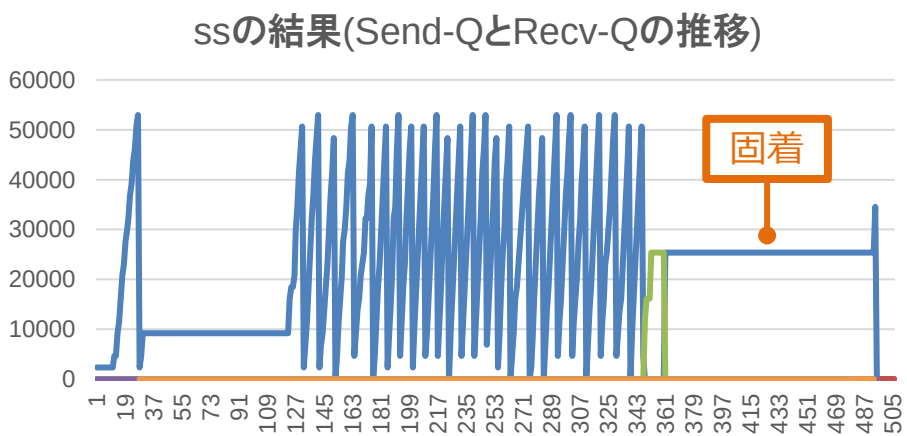
ログ解析の結果

HITACHI
Inspire the Next

ログ結果を可視化して原因分析。PLCデータがRCまで未着となっていることが判明。
原因としては割り込み処理が関係か。詳細分析が必要。

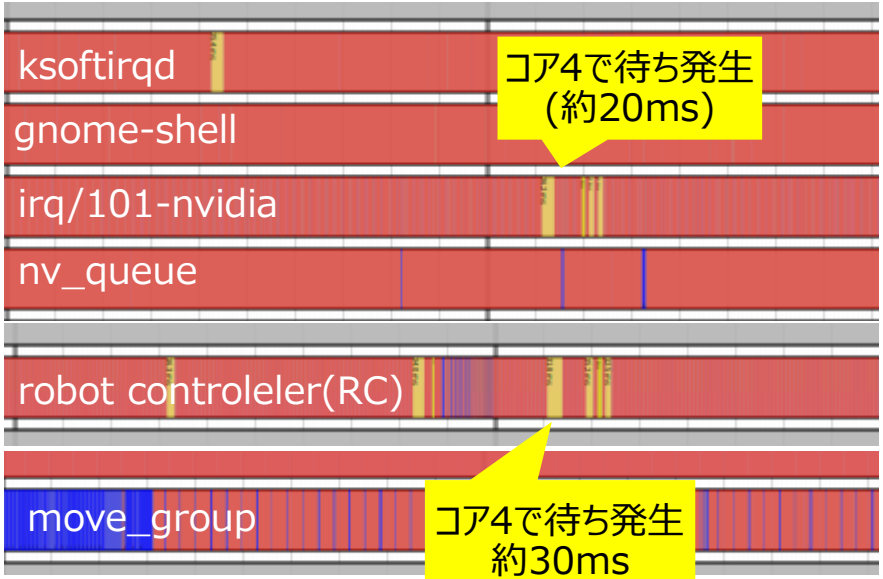
■OSの状態確認

受信バックログ(Recv-Q)が一定値で固着しており、
通信ドライバが受信した後に、ROSノードに届いて
おらず停止していることが確認できた



■各コアの処理状況の監視

RCの大きな待ち発生の直前に、
同コアでNvidiaドライバの割り込み処理が発生。



ログ解析結果の深堀分析と原因推定

HITACHI
Inspire the Next

perfをscriptでテキスト化し、エクセルで取り込み。RCを最高優先度で実行する設計だが、起動から実行までの間に、優先度無視のハードウェア割込み処理が発生。RCの実行遅延の原因となっている。

no	process	pid	CPU	elappsed ti	event	info
634031	move_group	10192	[004]		sched:sched_switch	prev_comm=move_group
634032	irq/101-nvidia	1773	[004]		sched:sched_wakeup	comm=Xorg
634035	irq/101-nvidia	1773	[004]		sched:sched_switch	prev_comm=irq/101-nvidia
634058	move_group	10192	[004]		**sched:sched_wakeup**	comm=rviz
634143	move_group	10192	[004]		**sched:sched_wakeup**	comm=nodelet
634146	move_group	10192	[004]		**sched:sched_wakeup**	comm=rviz
634381	move_group	10192	[004]	0	**sched:sched_wakeup**	**comm=robot_con**
634429	move_group	10192	[004]	0.763	**sched:sched_wakeup**	**comm=rcu_sched**
634620	move_group	10192	[004]	4.972000002	**sched:sched_wakeup**	**comm=irq/101-nvidia**
635798	move_group	10192	[004]	28.763	**sched:sched_wakeup**	**comm=ksoftirqd/4**
635923	move_group	10192	[004]	33.022	**sched:sched_wakeup**	comm=single_axis_nod
635967	move_group	10192	[004]	34.806	**sched:sched_switch**	prev_comm=move_group
635968	**robot_con**	9916	[004]	34.844	sched:sched_switch	prev_comm=robot_con
635972	irq/101-nvidia	1773	[004]			comm=gnome-shell
635978	irq/101-nvidia	1773	[004]			prev_comm=irq/101-nvidia
635980	ksoftirqd/4	34	[004]			prev_comm=ksoftirqd/4

起動

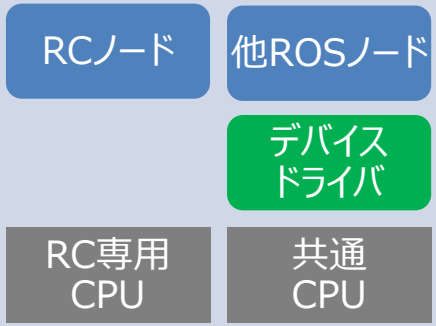
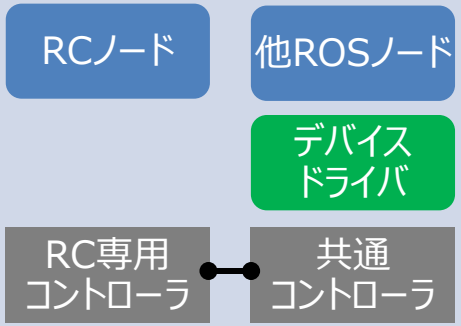
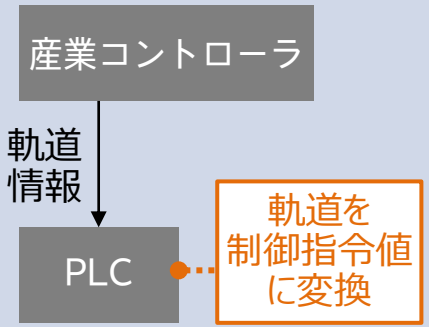
実行までに約34ms
かかっている

RCは動的な(OS任せの)コア割り当てとなっており、偶然Nvidiaドライバと RCが同コアで実行されたときに本遅延が発生するタイミング依存不具合

根本原因に対する対策案

HITACHI
Inspire the Next

3つの対策案を考案。本発表ではリアルタイム性が求められるRCとハードウェア割り込みを分離することで、意図しない遅延発生を抑制する「RC専用コア(割り込み分離)」の方式を紹介する。

項目	RC専用コア(割り込み分離)	複数コントローラで分散処理	PLC側でRC相当処理の実行
対策案			
リアルタイム性	○	△	○
CPU利用効率	△(優先度が低いアプリと混在させれば○)	○	○
工数・コスト	○	×	○
汎用性	○	○	△(ロボット依存)

RC: robot controller

RC専用コア(割り込み分離)方式の実現方法

HITACHI
Inspire the Next

各デーモンやROSノード各々に対して、実行コアを指定することは現実的でない。リアルタイム性が求められる処理(RC)の実行コアを指定し、他処理はOS任せで別コア実行となるようを設定。

ROSノードは、プロセスの大元から派生して生成される特性を利用し、
プロセスの大元にCPU affinityを設定することで、各ROSノードに同一の設定を継承させる。

CPU Affinity：動的コア割り当ての対象範囲

■ Host

Host内で実行する際の大元のプロセスは /lib/system/systemd である。
systemdが参照する/etc/sysemd/system.confに設定する。

```
CPUAffinity=0 2 3 4 5 6 7
```

例：RCを動作させるコア1として、その他を除外する

■ Docker

Docker内で実行する際の大元のプロセスは /usr/bin/xfce4-terminal となる。
Docker runの起動時にtasksetを用いて設定する。

```
docker run <オプション> image名 taskset -c 0,2-7 /usr/bin/xfce4-terminal -x /bin/bash -l
```

例：RCを動作させるコア1として、その他を除外する

実験方法

HITACHI
Inspire the Next

効果の確認のために実験を行う。解析時と同様にperfを用いてROSシステム動作を検証する。

■実験の内容
・高頻度で急停止が発生していたユースケースを対象に、5回実行し、perfを用いてログを収集する。
・一定時間(20~30秒ほど)経過後、ctrlCで終了する。

■プログラムの起動順序
・perfを実行
・roslanchを実行(ロボットの起動)

動作例

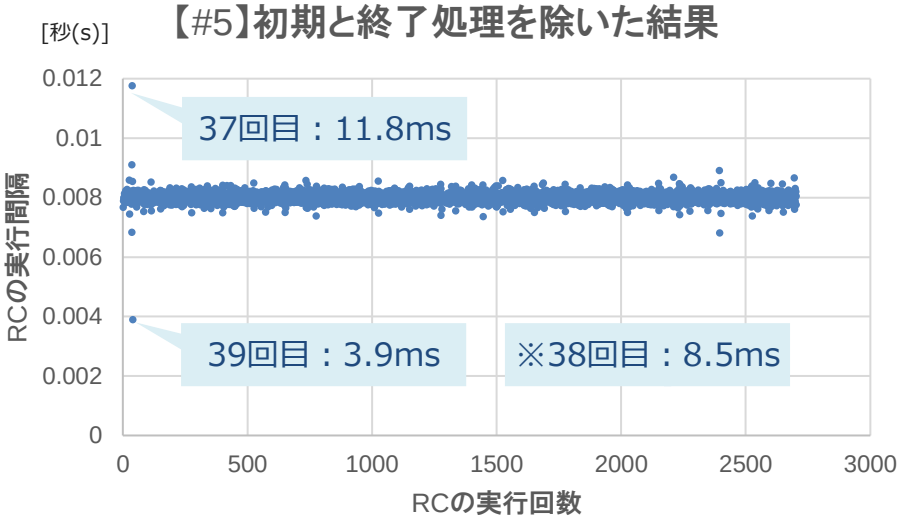
実験結果

HITACHI
Inspire the Next

PLCのキューが枯渇するようなデッドラインミスの連続発生なし。解決案で意図しない急停止が発生しないことを確認できた

・最大周期で11.8ms、周期が10ms以上もほとんど発生しない。
・遅延が発生しても、以降の実行が8msより短い間隔で実行される。(8ms周期に戻る)

回次	RC(8ms)の実行回数	最大周期	周期>10msの回数
#1	2654	10.2ms	1回
#2	2694	9.1ms	0回
#3	2633	10.4ms	2回
#4	4264	9.4ms	0回
#5	2703	11.8ms	1回



まとめ

HITACHI
Inspire the Next

■概要

- 物流ピッキングシステムにおける「意図しない急停止事例」を例題に、ROSの高信頼化に向けた解析事例と解決案を共有

■解析事例と解決案

- リアルタイム性が必要なROS処理とハードウェア割り込み処理が同コアで実行されると、リアルタイム性が満たせないことをログ情報から解析
- 前記ROS処理とハードウェア割り込み処理を別のコアで実行する「RC専用コア(割り込み分離方式)」を事例として紹介。
- RC専用コア(割り込み分離方式)の実現方法の１例を紹介。

■実験

- 解決案によって「急停止」が発生しないことを確認できた

HITACHI

Inspire the Next