6H-08

ROS ノードの実行時間の予測に基づく動的複数コア割り当て手法の検討

福井　誠人[†]　大野 雄杜[‡]　石綿　陽一[§]　大川　猛[‡]　菅谷　みどり[†]

芝浦工業大学[†]　東海大学[‡]　Ales株式会社[§]

1. はじめに

近年，高機能なコミュニケーションロボットが広く普及している．高機能なロボットを実現するためには，高速な演算処理が必要となり，それに合わせて高性能なプロセッサへの要件も増大している [1]．しかし，ロボットに搭載する計算機は限られており，その計算資源により，高機能なロボットサービスを提供するには限界がある．

これに対し高負荷な処理をクラウドにオフロードすることが考えられている[2]．しかし，クラウドとロボットの間の通信遅延があることから，近年はロボットの近距離に位置し，豊富な計算資源を持つエッジサーバに計算負荷をオフロードして計算を行うエッジコンピューティングが注目を集めている[3,4]．これらの研究では，クラウドサーバを使う場合に比べてエッジサーバにより性能向上が達成されている．このように，クラウドサーバとエッジサーバを比較する先行研究はいくつかみられるが，エッジサーバの構成や，利用の仕組みとして many-core の使用について指針を示すものは少い．CPU の動作周波数が限られる中で，計算力向上のためにマルチコアシステムは汎用的である一方，エッジサーバとして汎用的な仕組みで，有効に活用する例は少ない．また，アプリケーションにより有効性を示した例は十分ではない．

2. 提案

2.1. 目的

本研究では，課題を解決するために汎用的なマルチコアシステムにより，実際のアプリケーションに対し，動的複数コアの資源割り当てをする手法を検討する．また，具体的なアプリケーションとしてロボットや自動運転で普及している SLAM (Simultaneous Localization and Mapping)を用い，その有効性を評価することを目的とする．実現のために，SLAM を利用するために必要となる開発に向けた分散システムのミドルウェアである ROS を拡張する．本提案により自動的に SLAM にコアの資源割り当てを行う資源管理システムの提案をすることを目的とする．実現の為に(1) many-core CPU でのプロセッサの割り当て手法の検討，(2)小規模なサンプルアプリケーションによる many-core CPU の評価を行う．

2.2. ROS (Robot Operating System)

はじめに，本研究で用いる SLAM アプリケーションを動作するための環境を提供するミドルウェアである ROS(Robot Operating System)[5]について説明する．本研究では，ROS を拡張し，動的コア制御のための仕組みを提供する．ロボット制御システムでは通常複数のプロセスが分散して計算処理を行う．ROS では非同期型の分散プロセスを node と定義している．ROS のアプリケーションは通常，複数の node から構成される．Node 間での通信には publish/ subscribe メッセージングモデルが使われる．これにより通信チャネルの確立を伴うイベントが生成され，共有メモリとして機能する topic を介して，読み書きが行われる．データを生成する node が publisher ,データを処理する側が subscriber でとなるのが一般的である．Subscriber では対象となる共有メモリ（topic）に変更があったときにコールバック関数が呼ばれる．

2.3. 提案システム

ROS では先述のように Subscriber で topic の更新のたびにコールバック関数が呼ばれる．このことから複数回コールバック関数を実行する際にキャッシュの親和性から効率的に実行できるのではないかと考えた．しかし ROS には node を特定のコアに割り当てる機能はない．そこで本研究では CPU キャッシュを利用できるようにすることでより効率良く実行することができないかと考え ROS の node をプロセッサへ割り当てする手法を提案するものとした．

3. ROS 使用時のプロセッサの割り当て手法

システムは ros melodic とそれに対応するクライアントライブラリである roscpp に変更を加えることで実現する．また，現段階では分散システムを考慮せずに 1 台のマシン内で完結する限定的な状況で適用できるとする．

まず，node を作成し任意の topic を subscribe したときに node の PID や現在設定されている affinity を記録しておく．roscpp ではコールバックが呼び出されるとき ros::callbackQueue::callOneCB 関数内でユーザが設定したコールバック関数が呼ばれる．そこでここで実際の呼び出しにかかった時間を計測する．このときに過去の関数の呼び出した時間をもとにコアの割り当てを変更可能にする．以下の図 1 では 3 次元 SLAM 処理を用いたロボットの自己位置推定を行う際に本システムの適用例を示す．ロボットに搭載された LIDER から publish された /scan topic をエッジサーバ上の slam_gmapping node が subscribe する．この node は GMapping を使用した SLAM 処理を行う．エッジサーバ上ではプロセスをコアに配置，負荷に応じた割り当ての変更を行う事を示す．

Copyright ©2021 Information Processing Society of Japan.
All Rights Reserved.

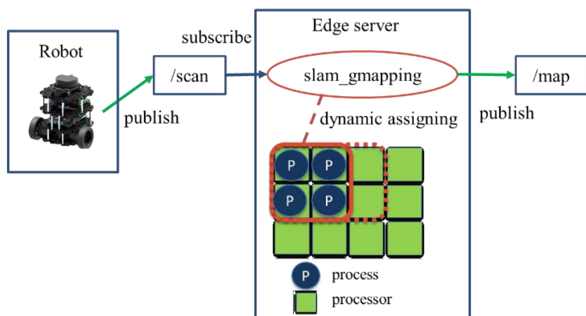

図 1 提案システムの概要

4. 予備実験

4.1. 予備実験 1: CPU バウンドなプログラム

Many-core CPU 上への処理割り当ての基礎的な性能を評価するため、算術計算を行うプログラムを many-core CPU 上で実行する実験を行った．この実験では複数回 fork() することで子プロセスを作るマルチプロセスプログラムに割り当てるプロセッサ数を変化させながらプログラム全体の実行時間を調査した．結果を図 2 に示す．ここではそれぞれの子プロセスでは多数のループを伴う円周率の計算を行い CPU を消費しているため，プロセス数を増やすにつれ実行時間は増加した．このプログラムに対して割り当てる CPU を増やしていくことで反比例的に実行時間は減少する．本実験では単純な fork() を複数回呼び出すプログラムであるがこのようなプログラムが複数回呼び出されるソフトウェアについて考える．ここで，本結果をもとに次の一般式を導いた．

$$(\textit{実行時間の推測値})\ T_{exec} = \sum_{i=0}^{jobs}\left(\alpha_i \times \frac{Process_i}{Core_i} + \beta_i\right)$$

数式 1　実行時間の推測式

ここで，α，βはそれぞれプログラムによって定まる定数であり jobs はそのソフトウェアを構成する個々のアプリケーションユニットの個数である．

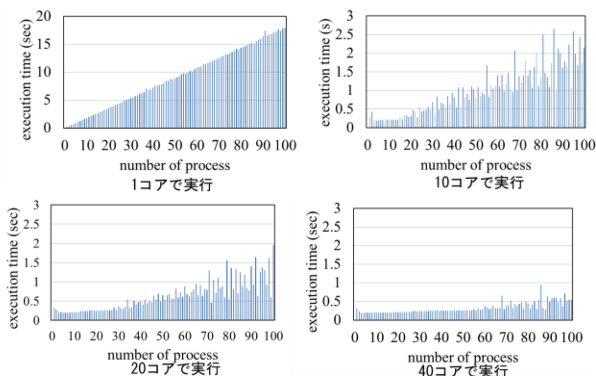

図 2 予備実験 1 結果

4.2. 予備実験 2: Python による機械学習

Python で書かれた AI アプリケーションに任意の CPU を割り当てて実行しこのときの性能について調査した．Python でのプログラムは sklearn ライブラリを使い並列で処理できるようにしたものを使用した．今回作成したプログラムは次の動作を行う．まず，c 言語で書かれたランチャープログラムが起動する．このプログラム上から Python スクリプトを呼ぶために fork() し，子プロセスのメモリ空間を execve() システムコールを使い対象の Python スクリプトに書き換える．このときに sched_setaffinity() を使い指定した CPU を割り当てることを可能にした．

4.3. 予備実験 2: 結果

実験は Intel Core i7-7700 の CPU と DDR4 のメモリを 16GB 搭載した一般的な PC サーバを使い行った．また，OS は Fedora32，Python のインタプリタは Python3.8.5 をそれぞれ使用した．このときの結果を図 3 に示す．割り当てたプロセッサを 1 から 2 に増やしたとき実行時間は 2 倍以上増加しその後 5 プロセッサまで増やしたとき増加の傾向は緩やかになった．その後はおおむね横ばいとなった．当初の予測では割り当てるプロセッサ数を増やすことで実行時間を短縮できると考えていたが，それに反して実行時間が増加する結果となった．現段階では要因の特定ができていないため今後の課題としたい．

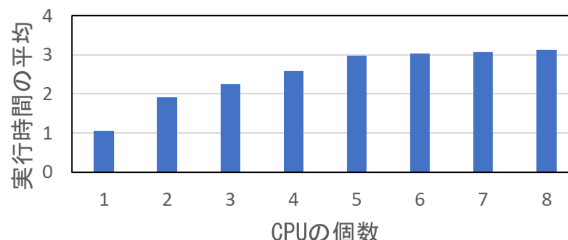

図 3 CPU の割り当てと実行時間

5. おわりに

本論文では ROS に向けた node のコア割り当て手法を提案した．予備実験では単純な計算を many-core のマシンにおいてコアに割り当てて実行した結果，CPU のキャッシュが生かされたため実行時間の減少が見られた．

今後の課題として提案するシステムの実装及び評価，また現在対応できていない分散システムへの完全対応をすることがあげられる．

謝辞

本研究は,JST,CREST,JPMJCR19K1 の支援を受けたものです．

参考文献

[1]"Artificial Intelligence and Robotics", https://arxiv.org/ftp/arxiv/papers/1803/1803.10813.pdf (参照 2020-11-5)

[2]Sandeep Chinchali, "Network Offloading Policies for Cloud Robotics:a Learning-based Approach" , arXiv:1902.05703v1

[3]Youdong Chen and Qiangguo Feng, "An Industrial Robot System Based on Edge Computing: An Early Experience", HotEdge'18

[4]R. M. Shukla and A. Munir, "An efficient computation offloading architecture for the Internet of Things (IoT) devices," 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, 2017, pp. 728-731, doi: 10.1109/CCNC.2017.7983224.

[5]" ROS.org | Powering the world's robots", https://ros.org (参照 2021-01-07)

Copyright ©2021 Information Processing Society of Japan.
All Rights Reserved.