スケーラブルなROS環境に向けた資源管理機構

福富 大輔[1,a)]　安積 卓也[2]　加藤 真平[3]　西尾 信彦[1]

概要：近年，ロボット開発のためのミドルウェアである Robot Operating System (ROS) が活用されている．ROS は複数台のホストマシンを用いた分散環境に対応している．一方，ROS では分散環境において各ホストマシンの CPU やメモリ，ディスクといった資源の管理がなされていない．ROS において資源が管理されていないことにより，分散環境の資源を効率よく使うことができずスケーラブルに処理を実行することが困難である．本研究では，複数の処理を ROS 分散環境で効率よく円滑に稼働させるために，各ホストマシンの資源の利用を把握し，適応的に負荷分散を行う機構を構築する．評価では，10 台のホストマシンで構成される ROS 分散環境上で，スケーラブルに処理を実行することが可能なことを示した．

キーワード：ROS，資源管理，分散環境，スケールアウト，自動運転

Resource Management Mechanism for Scalable ROS Environment

Daisuke Fukutomi[1,a)]　Takuya Azumi[2]　Shinpei Kato[3]　Nobuhiko Nishio[1]

Abstract: In recent years, Robot Operating System (ROS) which is middleware for robot development is utilized. ROS supports distributed environment using multiple host machines. However, ROS does not manage some resources such as CPU, memory, and disk of each host machine in distributed environment. It is difficult to efficiently use distributed environment resources and to execute processing scalably. In this research, in order to efficiently and smoothly operate the processing in the ROS distributed environment, we manage the usage of resources of each host machine and construct a mechanism to adaptively distribute the load. Evaluation showed that it is possible to execute processing scalably on the ROS distribution infrastructure consisting of 10 host machines.

Keywords: ROS, resource management, distributed environment, scale-out, autonomous driving

1. はじめに

近年，自動運転車に関する研究が交通事故の削減や無人運転タクシーなどの商用化を目的として盛んである [1], [2], [3]．オープンソースソフトウェアの「Autoware」[2] をはじめとした自動運転システムは，自動車をロボットとしてみなして開発が進められている．そこで，ロボット開発のためのライブラリやツールを提供するフレームワークとなるソフトウェアが用いられている．ロボット開発のためのフレームワークとして，オープンソースライブラリである OpenCV [4] や Point Cloud Library (PCL) [5], [6] に対応する Robot Operating System (ROS) [7] がデファクトスタンダードとして用いられている [8]．ロボットシステムは，センサ処理系と制御系の分離や大規模なデータの処理へ利用することを考慮した場合，複数台のホストマシンに処理を分散させる必要があり，ROS も複数台のホストマシンを用いた分散処理に対応している．

ROS を用いたスケーラブルな処理が可能な分散環境が必要になる例として，自動運転に用いる 3 次元地図の生成があげられる．自動運転車は，ROS 上で Light Detection and Ranging (LiDAR) やステレオカメラを用いて取得可能なデータである Point Cloud [6] から事前に生成された 3 次元地図と，リアルタイムに得られる Point Cloud のマッチングを行い自己位置を推定する．そのため，自動運転は 3 次元地図のない地点では行うことができない．3 次元地図の広域化は自動運転の普及のために解決すべき課題である [9]．自動運転の基盤として用いるような広域の 3 次元

[1] 立命館大学大学院 情報理工学研究科
Graduate School of Information Science and Engineering, Ritsumeikan University
[2] 大阪大学大学院 基礎工学研究科
Graduate School of Engineering Science, Osaka University
[3] 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
[a)] tommy@ubi.cs.ritsumei.ac.jp

地図生成のためには，予め，複数の車両から得られる大規模な Point Cloud データを処理して 3 次元地図を生成して合成する．そのため，分散環境で処理を行う必要があり，ROS にはスケーラビリティが求められる．

一方，ROS では分散環境において各ホストマシンの CPU やメモリ，ディスクといった資源の管理がなされていない．ROS の各処理はノードという単位で周期的に実行される．ROS において資源が管理されていないことにより，分散環境へ処理を割当てる際に 2 点の問題が存在する．第一に，分散環境上のホストマシン間で処理の割当てに偏りが発生し，分散環境の資源を効率よく使うことができずスケーラブルに処理を実行することが困難である．第二に，処理中のホストマシンで利用可能な資源の上限に達し，資源に余裕のある他のホストマシンへ処理を移行する場合，処理が前周期の内部状態を参照している（以降，ステートフル）場合においては，その内部状態であるステートが失われてしまうことである．そこで，提案手法では，Master-Slave 方式の資源管理機構を構築し，ホストマシン間で処理の偏りが発生した場合他のホストマシンへ処理中のノードを移行することで資源を効率よく利用する．加えて，ステートとなる情報をトピックに対し配信することによりステートフルなノードにおいても利用可能な機構を構築する．

本論文の貢献は主に 2 点である．第一に，ROS において管理されていなかった各ホストマシンの資源を管理することにより分散環境上で処理の偏りを防ぐことができる．このことは，分散環境を構築する目的であるスケールアウトをより効率よく実現し，ROS 分散環境で大規模なデータを用いた処理を可能とする．第二に，資源を管理し他のホストマシンへ処理を移行する際に，ステートフルなノードにおいても ROS のトピックを利用して解決する手法を提案したことである．これら 2 点を実現することで，複数台のマシンで構成される ROS の分散環境において実利用可能な資源管理機構を構築する．

本論文は全 6 章で構成されている．2 章では ROS 分散環境の問題点とスケーラブルな処理の例をあげる．3 章では ROS 資源管理機構の提案手法について述べ，4 章で提案手法の動作および実アプリケーションを用いた評価について述べる．5 章では関連研究をあげる．最後に，本研究のまとめと今後の課題について 6 章で述べる．

2. 関連技術

2.1 Robot Operating System

ROS はロボット開発のためのライブラリやツールを提供するソフトウェアプラットフォームである [10]．ROS のアプリケーションはパッケージ単位で開発されており，各パッケージはアプリケーションの機能を細分化した Linux におけるプロセスに相当するノードにより構成されている．

ROS のノード間の通信は主に 2 つの方式が使われる．一つ目の ROS の代表的なノード間の通信方式である Topic は，**図 1** に示す Publish / Subscribe 型の通信を行っている．ROS では，図 1 の Publisher / Subscriber が各ノードに相当する．Topic は，データの型と中身を指定したデータの名前付きバスであり，Publisher や Subscriber はトピック名を指定することでメッセージをやり取りする一方向の非同期型メッセージ通信である．もう一つの方式である Service は，サーバクライアントの関係で提供される通信である．Service はリクエストとレスポンスの 2 つで区別されたデータの型で情報を双方向に送受信する同期型通信である．

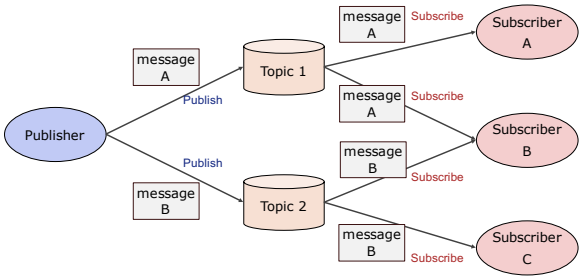

図 1 Publish / Subscribe モデル
Fig. 1 Publish / Subscribe model.

ロボットシステムを構成する際，単一のホストマシンに必要なセンサや制御系部品をすべて設置して利用することは，認識系と制御系の切り分けを容易にする観点や，障害発生時にすべての機能が失われフォールトトレラント性を持たないため効率的ではない．そのため，ROS では複数台のホストマシン上で動作できるように設計されている．一方，ROS の分散環境においてホストマシンの資源管理が行われておらず，分散環境の資源を効率よく利用できない．

2.2 ROS における SLAM の実アプリケーション

Simultaneous Localization and Mapping (SLAM) とは自己位置推定と環境地図の生成を同時に行うことをいう．LiDAR やステレオカメラなどのセンサから取得可能な Point Cloud を用いて SLAM を行うことが主流となっている．3 次元空間上での Point Cloud を用いた SLAM としては，Besl ら [11] によって提案された Iterative Closest Points (ICP) アルゴリズムや Magnusson [12] によって提案された Normal Distributed Transform (NDT) アルゴリズムを用いたグリッドベースの SLAM が存在する．グリッドベースの SLAM は自動運転が対象とする屋外でも精度が良く，PCL でもライブラリとしてアルゴリズムが提供されていることで容易に利用が可能である．

図 2 は自己位置推定と環境地図生成をした後，生成した地図をファイルとして書き出すプログラムの流れである．太枠部が SLAM の様々なアルゴリズムが存在する部分である．本研究では，ICP アルゴリズムのアプローチに対し，ボクセルに区切ることで計算量を減少させている NDT アルゴリズムを用いた SLAM を ROS で用いる．図 2 のよ

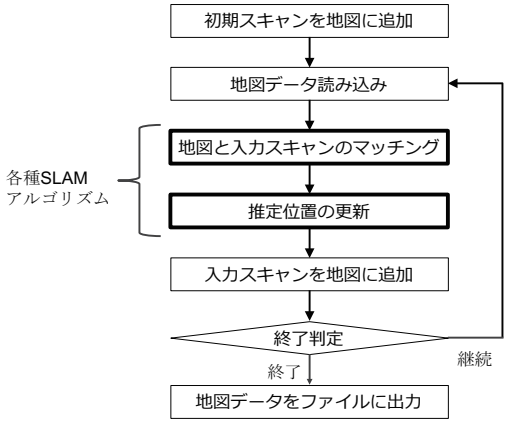

図 2 自己位置推定と環境地図の生成の流れ
Fig. 2 Flow of localization and mapping.

うに SLAM を用いて事前に 3 次元地図を生成し，自己位置推定のための参照スキャンとして活用することで，走行車両が地球上のどこにいるのかわかる．自動運転車は，自車両内でリアルタイムに事前の 3 次元地図とセンサデータのマッチングを行いながら走行する．図 2 の処理では毎スキャンごとにファイルに書き出すことは入出力による処理時間が増加するためオンメモリで処理しており，一度に生成可能な地図の大きさはセンシングデータである Point Cloud の密度とホストマシンのメモリの大きさに依存する．以上のことから，メモリ不足による障害も発生するため ROS において資源管理し，障害のリスクを減らす必要がある．一方，参照スキャンとして用いるために事前に行う 3 次元地図生成は，複数台から得られるセンサデータをクラウド上に集約してオフラインで処理し，3 次元地図を広域化することで自動運転車が走行できる範囲を広げることができる．そのため，3 次元地図を効率よく構成するためには分散基盤の資源も管理する必要がある．

3. 提案手法

3.1 ROS 分散環境における資源管理機構の構築

本研究では，ホストマシンの資源を CPU・メモリ・ディスクと定義する．提案手法では，分散基盤における代表的な資源管理機構である YARN でも採用されている Master-Slave 方式を採用する．Master-Slave は Master による一方的な制御であり，通信方式に Publish / Subscrive モデルを採用している ROS との親和性が高いため，本研究において資源管理機構を構築する上でも最適である．

本研究では，**図 3** に示す，各ホストマシンの資源を管理し処理を割当てる Master Node と，資源情報を配信し処理（本研究では，ROS のノード）を呼び出す Slave Node に分割し，資源管理機構を構築する．

図 3 に示すように，複数のホストマシンのうち一台で Master Node を立ち上げ，他のホストマシンでは Slave Node を立ち上げることで Master-Slave 方式の管理を実現する．Master Node を起動する一台のホストマシンを

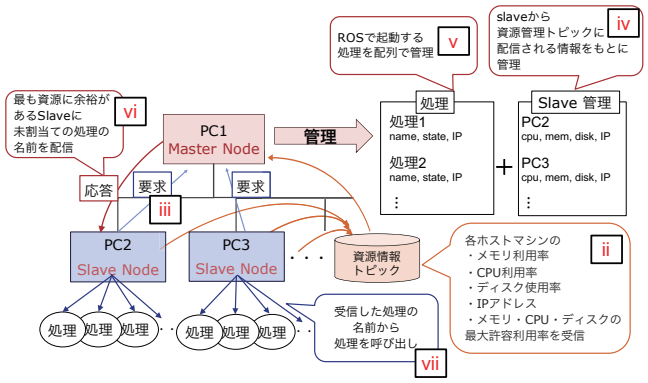

図 3 ROS 分散基盤における資源管理機構
Fig. 3 Resource management mechanism for ROS distribute environment.

Master，Slave Node を起動する任意の台数のホストマシンを Slave と呼ぶ．提案手法における資源管理機構と処理の割当ては以下の流れとなり，図 3 では，箇条書きに対応する部分を赤字で示している．

(i) 各ホストマシンの上限となる CPU・メモリ・ディスクの利用率（資源情報）を Slave Node に設定
(ii) Slave Node が起動しているホストマシンの現在の資源情報と各資源利用率の上限を資源管理トピックに配信
(iii) Slave Node は現在の CPU・メモリ・ディスクの利用率が各リソース上限を超えていない場合，処理の割当てを Master Node に要求
(iv) Master Node は資源管理トピックの情報を購読し，IP アドレスから新規のホストマシンであればホストマシンの管理リストに登録し，既存のホストマシンであれば管理リストの資源情報を更新
(v) Master Node は割当てる処理を標準入力から受け付け，配列を用いて First-In First-Out で管理
(vi) Slave Node から処理の割当てが要求されていた場合，ホストマシン管理リストのうち資源に最も余裕がある Slave に処理を割当て
(vii) Slave Node で割当てられた処理を呼び出し

以上の流れにより，分散基盤を構築する各ホストマシンの資源を管理しながら ROS の各ノードを立ち上げることができる．本資源管理機構により，各ホストマシン間の処理の偏りを低減することが可能となる．加えて，資源不足による障害を未然に防ぐことに寄与する．

資源情報を用いた処理割当ての詳細について述べる．Slave の資源の余裕は，分散機構管理者が 0 – 1 の値を取る任意の重み付け変数 $w_{mem}, w_{cpu}, w_{disk}$ を決定した上で，本研究で扱うメモリ，CPU，ディスクの各利用率を $memusage, cpuusage, diskusage$ とし，式 1 により score を算出する．ROS のノードは，実際に処理が行われるまで，どの程度資源を利用するかわからないため，この score が最も小さい Slave を (vi) で述べた資源に最も余裕がある Slave とみなし処理を割当てる．一方，一度処理したこと

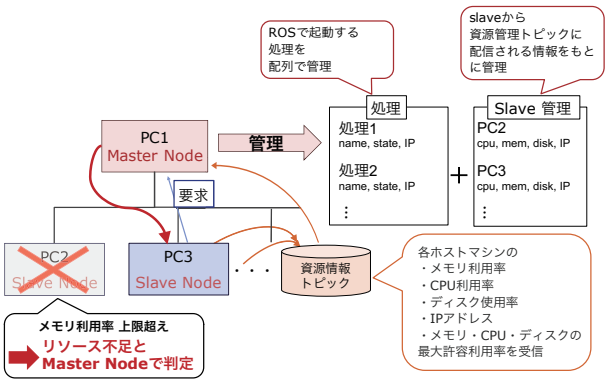

図 4 ROS 資源管理機構におけるノードの移行動作
Fig. 4 Shif of Node in ROS resource management mechanism.

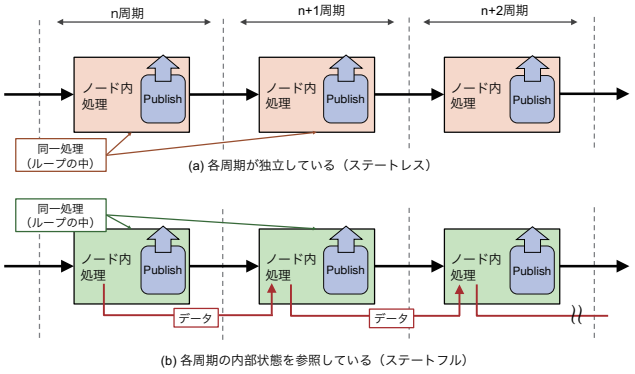

図 5 ROS の処理におけるステート
Fig. 5 Stateful and stateless in ROS processing.

のあるノードに関しては，必要となる資源の種類と量がわかるため，より効率的な処理の割当てができる可能性があり今後検討する．

$$score = w_{mem} \times memusage + w_{cpu} \times cpuusage + w_{disk} \times diskusage \quad (1)$$

資源の情報を配信するに当たり Slave の各ホストマシンでは (ii) , (iii) のように資源の上限値への到達の有無を Slave Node 側で判断している．これには 2 点理由があり，1 点目は各 Slave Node ごとに許容する利用率が異なることが想定されることと，2 点目は Master Node での Slave からの要求処理と資源上限値と現在値を比較する計算処理を軽減することである．ホストマシンの増加などの拡張性を考慮した場合，Master での計算処理の軽減は重要である．

3.2 処理のホストマシン間の移行手法

3.1 節で述べた資源管理機構を用いた際，Slave において資源の利用率が事前に設定した上限に達する場合がある．その場合に，資源の上限となった Slave に割当てられている処理を他の Slave に再割当てする処理の移行手法を提案する．図 3 で示した資源管理機構では，**図 4** で示すように PC2 に割当てた処理がある状態で資源の上限値に達する場合を Master で判断できる．図 4 では PC3 に PC2 で行っていた処理の割当てを行い PC3 に処理を移行する．資源の上限に達した Slave から移行させる処理は，資源の上限を上回っている Slave に最初に割当てられた処理から順に選択する．選択した処理を，Slave の資源が上限値を下回るまで，順次他の Slave に移行する．すべてのホストマシンの資源の利用率が事前に設定した上限に達した場合は，新たな処理の割当てを停止し，処理の移行も行わないため，移行が頻発することはない．

ノードの移行動作において再割当てが行われる処理，つまり図 4 中での PC2 で立ち上がっていた処理には，ROS におけるノードでは**図 5** に示す 2 つのパターンがある．一つは，図 5 の (a) に示すようなステートレスな場合である．ステートレスとは，一定周期で Publish / Subscribe しているノードにおいて配信・購読する情報が前周期の情報に影響されないものをいう．例えば，一定周期で同じメッセージを配信し続けるノードや，センサから得られる生データをそのまま配信するノードの場合などである．もう一つは，図 5 の (b) に示すようなステートフルな場合である．ステートフルとは，一定周期で Publish / Subscribe しているノードにおいて配信・購読する情報が前周期の情報に影響されるものをいう．例えば，ノードが起動してからの時間を配信するノードや，2.2 節で述べたような地図生成における車両の位置推定結果などである．

ステートレスな場合

ROS では同一名のノードを起動することができないため，同一名のノードを起動した場合，以前立ち上がっていたノードは終了される．そのことを利用し，同一名で他のホストマシンに同じノードを割当てることで，ホストマシン間でステートレスなノードを移行できる．この手法では既存のパッケージやアプリケーションに含まれるノードを書き換える必要がないことが利点である．

ステートフルな場合

図 5 の (b) のようにステートフルな場合，新たに同一名のノードを立ち上げた際，すべてのステートが失われてしまう．なぜなら，一定周期で実行されているノードにおいて前周期に依存する場合，変数などによってデータを保持して利用しており，新しく立ち上げたノードでは引き継ぐことができないためである．そこで，ステートがあるノードでは以下の 2 点の制約を設けた手法を提案する．

(i) ステートがある情報は毎周期必ず配信
(ii) Master Node から配信されるトピックにより呼び出される関数内において，ステートのある変数を Publish されている情報で初期化

この制約は，ステートを把握しているアプリケーション開発者によって満たされる必要がある．本制約のうち，(i) に示した制約は，ステートのある情報は他のノードで利用されることが多いため，毎周期 Publish することは，アプリケーション開発者にとって負担ではないと考えられる．

Master Node ではステートフルなノードの移行が発生

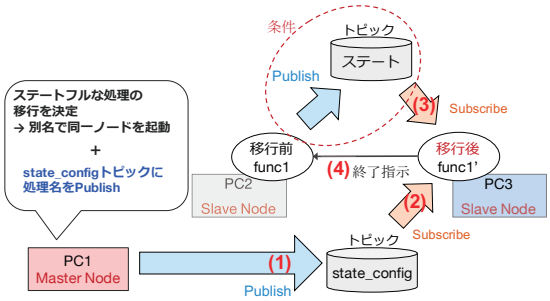

図 6 ステートフルなノードの移行動作
Fig. 6 Shift of stateful node.

した場合，**図 6** に示すような順序で処理の移行を実現する．まず，MasterNode は別名で移行先となる同一のノードを立ち上げ，示すようにノードの初期化のトリガとなる state_config topic を配信する（図 6 中，(1)）．移行先のノードでは，Master Node から配信された state_config topic を購読し（図 6 中，(2)），引き継ぎ前のノードが Publish している情報を Subscribe するための関数を呼び出す．呼び出された関数では，移行前のノードから Publish されている情報でステートフルな変数などを初期化し（図 6 中，(3)），移行前のノードの終了指示を出す（図 6 中，(4)）．このことにより，前周期の情報によって変数を初期化することが可能であり，ステートフルなノードに関してもホストマシン間の移行ができる．

4. 評価

4.1 評価環境

評価では式 1 による CPU，メモリ，ディスクの各利用率の資源管理への影響度は $w_{mem} = 1, w_{cpu} = 0, w_{disk} = 0$ としメモリの利用率のみが処理の割当てに影響を及ぼす．これは，2.2 節で述べたような実アプリケーションでメモリの利用が問題になることが多いことや，評価の妥当性を容易に確認するためである．さらに，各ホストマシンにおけるメモリの上限利用率は，処理が安定して動作する 80%に設定し，動作周期は ROS の標準周期である 10Hz とした．

Slave 台数を増やした環境でも提案機構では同様の動作であるが，リソースの状態の時間経過図などが複雑となり動作の確認という目的を達成することが難しいため以下の 2 つに分けた評価環境を構築した．

(i) 本資源管理機構における動作を評価するための環境
(ii) 本資源管理機構を用いた ROS 分散環境のスケーラビリティを評価するための環境

上記 (i) では，**表 1** のホストマシンを 3 台用意し，1 台で Master Node と Slave Node の両方を起動し，残りの 2 台で Slave Node を起動する．3 つの Slave により評価を取ることで，提案機構の動作が適切に行われていることを評価した．上記 (ii) では，**表 2** の 10 台のホストマシンそれぞれで Slave Node を起動する．10 台の Slave を用いること

表 1 資源管理機構の動作に関する評価環境
Table 1 Evaluation environment about operation of resource management.

OS	Ubuntu 14.04
Kernel	Linux Kernel 3.13.0
Memory Size	16 GB
ROS version	ROS Indigo Igloo

表 2 スケーラビリティに関する評価環境
Table 2 Evaluation environment about scalability.

OS	Ubuntu 14.04
Kernel	Linux Kernel 3.13.0
Memory size	4 GB
ROS version	ROS Indigo Igloo

で，ROS のスケーラビリティと起動するノード数や Slave の増加に対する資源管理機構の処理限界を評価した．

4.2 ROS 分散環境における資源管理

提案手法に則った資源管理機構の動作となることを示す．本項における評価項目は以下の 3 点であり，評価環境は表 1 である．

(i) 資源管理トピックのモニタリング
(ii) 資源に最も余裕のあるホストマシンへの処理の割当て
(iii) 指定した資源の上限に達した場合の処理の移行

資源管理トピックのモニタリング

各 Slave は，自ホストマシンの資源状態を資源管理トピックとして配信する．**図 7** は，評価環境における資源管理トピックを購読したもののうち 1 つの Slave から配信される情報を示す．Slave は，上から順に，CPU の利用率上限値 [%]，メモリの利用率上限値 [%]，ディスクの利用率上限値 [%]，IP アドレス，現在の CPU 利用率 [%]，現在のメモリ利用率 [%]，現在のディスク利用率 [%] を配信する．CPU，メモリ，ディスクの利用率の上限は利用者が任意に定める．このように，Slave は各資源の利用率を資源管理トピックに配信し，本資源管理機構の Master Node はこの資源管理トピックを購読することによって Slave を管理することが確認できた．資源管理トピックを ROS のノードで購読することで，資源の情報を容易に取得することができるため，本資源管理機構のみではなく任意のアプリケーションで資源の利用率を把握することができる．

資源に最も余裕のあるホストマシンへの処理の割当て

資源管理機構により，資源に最も余裕のある Slave に処理が割当てられ，評価における資源の上限値であるメモリ利用率が 80 %までに制限ができていることを示す．**図 8** では，ほぼ同時刻に 3 台の Slave を起動した場合の時間経過と資源の利用率を示した図である．ここで，各 Slave に割当てている処理は 1 処理当たり 8-80 MByte のメモリを利用する処理を各ホストマシンの資源の上限に達するのに

```
1 cpuusage_max: 80
2 memusage_max: 80
3 diskusage_max: 100
4 ipaddr: 192.168.2.226
5
6 cpuusage: 2
7 memusage: 11
8 diskusage: 9
9 allocate: True
```

図 7 資源管理トピックのモニタリング
Fig. 7 Monitoring of resource management Topic.

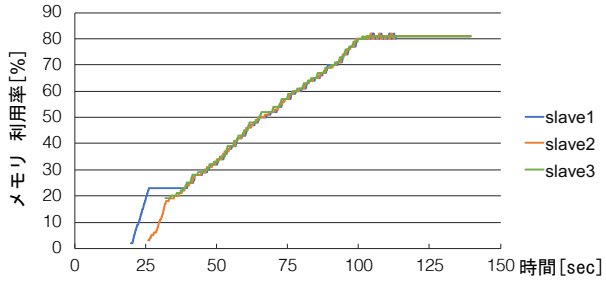

図 8 資源管理機構による処理の割当て
Fig. 8 Allocation of processing by resource management mechanism.

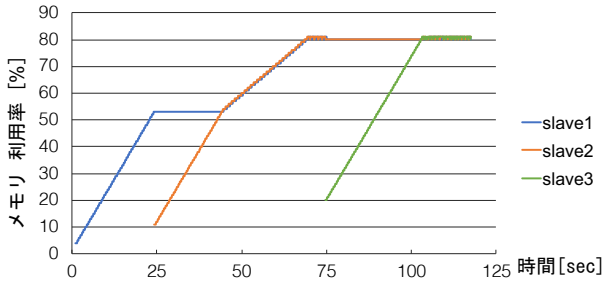

図 9 Slave 追加時の処理の割当て
Fig. 9 Allocation of processing when Slave is added.

十分な数を与えた．図 8 から，各 Slave がロードバランスを保ちながら処理を行い，設定した資源の上限まで処理が割当てられ続けることがわかる．**図 9** では，時間差で 3 台の Slave を起動した場合の時間経過と資源の利用率を示した図である．ここで，各 Slave に割当てている処理は 1 処理当たり 32 MByte のメモリを利用する処理を各ホストマシンの資源の上限に達するのに十分な数を与えた．図 9 から，25 秒あたりや 75 秒の Slave の追加のタイミングを見ると資源に最も余裕のある Slave に処理が割当てられていることがわかる．以上より，資源に最も余裕のある Slave に処理が割当てられ，資源の上限値に処理の割当てを制限することができていることを示せた．

指定した資源の上限に達した場合の処理の移行

指定した資源の上限に達した場合の処理の移行が行われていることを示す．ここでは，ステートレスな一つの処理でメモリリークを発生させ，常にメモリを確保し続ける処理を 3 つ順次割当てる．**図 10** は 3 台の Slave を起動した状態で開始した時間経過と資源の利用率を示した図である．図 10 にあるように，処理 1 の開始は先程示した通り資源の利用率が最も小さい Slave1 に割当てている．100 秒経過時に処理 2 が Slave2 に割当てられた．約 180 秒経過時，Slave1 ではメモリの利用率が設定した上限である 80%を超えたため処理の移行を行う．この時，処理の以降先となるのは，その時点で資源に最も余裕のある Slave となる．ここでは，Slave3 に処理が割当てられ，正しい動作となることが確認できた．処理 2 に関しても同様に資源に最も余裕のある Slave に移行が行われることが図 10 からわかる．評価開始から 300 秒経過時，処理 1 が Slave2 に移行しているため，図 10 で Slave2 では処理が 2 つ行われておりグラフの傾きが大きくなっていることがわかる．以上より，指定した資源の上限に達した場合の処理の移行が行われていることを示せた．

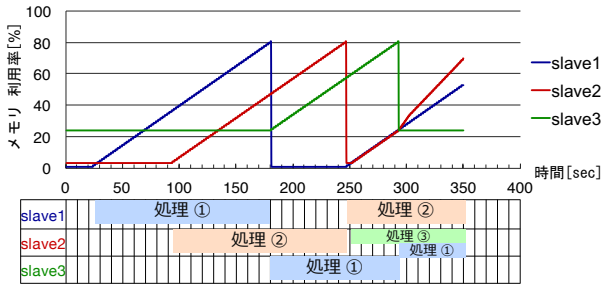

図 10 資源の上限値に達した場合の処理の移行
Fig. 10 Shift of processing when the upper limit value of resources is reached.

4.3 資源管理機構を用いた分散環境の処理限界

資源管理機構を用いて分散環境に対し処理を割当てた際のスケーラビリティに関して示す．本項の評価環境は表 2 である．分散環境では，処理可能な処理数は Slave の台数に対してできるだけ比例になることがスケーラビリティを持つといえる．加えて，資源管理機構では Slave の資源情報を Master で購読し，Slave の score を計算し処理を割当てる．よって，Slave 台数の増加に伴い Master の処理負荷が大きくなるため，本資源管理機構における処理限界についても評価した．

資源管理機構を用いて，分散環境を構築した各 Slave に対し，1 処理あたり 40 MByte のメモリを利用する処理を各 Slave の資源が上限に達するまで割当てた．**図 11** は，その際の，Slave の増加に対する割当て可能処理数を示す．図 11 からは，Slave 数の増加に伴って，割当て可能な処理数が比例関係を持ち増加していることがわかる．このことにより，資源管理機構を用いて処理を管理した分散環境はスケーラビリティを持つことを示せた．

次に，**図 12** は Slave の増加に対する，Master の 1 周期での処理時間を示したものである．ここでは，分散資源に

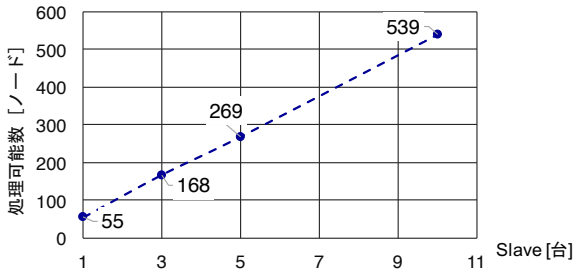

図 11 分散環境の Slave 増加に伴う処理可能数
Fig. 11 Processsable number due to increase of Slave in distributed environment.

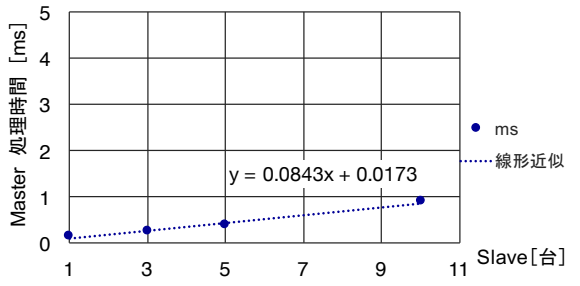

図 12 Slave 増加に対する Master 処理時間と近似グラフ
Fig. 12 Master processing time and approximate time associated with increasing of Slave.

与える処理の個数は Slave の台数によらず，1 処理当たり 256 kByte 利用する処理を 500 個に固定した．図 12 から，Slave の増加に伴って処理時間が増加していることがわかる．内部の処理では最小の score を持つ Slave を走査する $O(n)$ の処理が行われることから，図 12 では線形近似したグラフを載せている．Slave の増加に対し，この線形近似式 $y = 0.0843x + 0.0173$ に則って Master の処理時間が増加すると考えると，理論上 1000 台以上の Slave を立ち上げた際でも資源管理機構で要求する 1 周期あたり 100ms 以内の処理時間を満たす．しかし，評価するのに十分な台数であるとはいえないため，今後，より大規模な分散基盤を構築して検証する必要がある．

10 台のホストマシンを用いて評価したことで ROS では台数の多い分散システムにおいて，資源管理機構は十分な処理性能がある．理論上，より大規模な分散基盤が構築できる可能性を示せたため，スケーラブルな処理が要求される実アプリケーションでも活用可能性が高い．資源管理機構では，Master-Slave 間の通信を用いて処理を割当てているため，通信によるオーバヘッドや，ホストマシンの資源に対してのみ制約を受けるといえる．

4.4 3 次元地図の生成アプリケーションによる評価

実アプリケーションとして 3 次元地図の生成ノードを利用し，ステートフルなノードに関しても処理の移行ができることを示す．2.2 節で述べた通り，3 次元地図の生成には自動車の位置と姿勢が必要となる．そのため分散環境で処

```
1  ndt_mapping1
2  RANGE: 0
3  SHIFT: 0
4  use_openmp: 0
5  (tf_x,tf_y,tf_z,tf_roll,
       tf_pitch,tf_yaw
       ): (1.2, 0, 2, 0,
        0, 0)
```

(a) 引き継ぎ前ログ

```
1  ndt_mapping167rep
2  RANGE: 0
3  SHIFT: 0
4  use_openmp: 0
5  (tf_x,tf_y,tf_z,tf_roll,
       tf_pitch,tf_yaw
       ): (1.2, 0, 2, 0,
        0, 0)
6  −0.0573255
7  −0.344271
8  0.650112
9  0.00105349
10 −0.00244442
11 0.000541393
```

(b) 引き継ぎ後ログ

図 13 3 次元地図生成アプリケーションの初期化ログ
Fig. 13 Initialization log of application for generating the 3D-map.

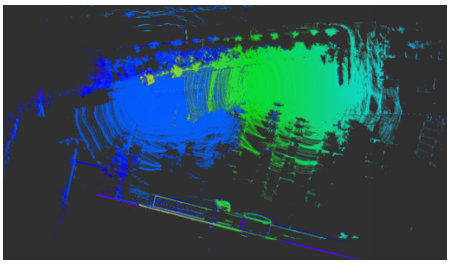

図 14 異なるホストマシン間で作成した 3 次元地図の重ね合わせ
Fig. 14 Superimposition of 3D-map generated between different host machines.

理をしていた 3 次元地図の生成を移行する処理では，車両の位置と姿勢を引き継いで地図生成することにより，引き継ぎ前で作成した地図と，引き継ぎ後の車両の地図を合成することが可能となる．**図 13** では車両の位置と姿勢の引き継ぎを行った際のログであり (a) が最初に起動していた処理の起動時のログ，(b) が移行先の処理の起動時のログである．移行先の処理では図 6 の流れに沿って車両の位置と姿勢を初期化している．図 13 (b) の 6-11 行目は，引き継ぎ前の処理から配信される 3 次元地図の生成自動車の位置と姿勢，車両と位置と姿勢を配信しているトピックを受信して初期化した値を示している．加えて，**図 14** では処理の引き継ぎ前のホストマシンで作成した 3 次元地図（青色）と，移行した先で作成した 3 次元地図（緑色）を重ね合わせたものである．図 14 からは，移行先のホストマシンで同じ車両の Point Cloud データを受信して地図が作られたことがわかる．本研究の資源管理機構により実アプリケーションにおいても処理を移行し，ステートの情報である車両の位置と姿勢も引き継ぐことができた．大規模 3 次元地図の生成のために分散環境でデータを処理することが可能となったため，今後は分散環境でそれぞれ作成した地図を

車両の位置を把握し，効率よく合成する手法を検討する．

5. 関連研究

Lauer ら [13] は，ROS のノード間通信を利用する処理においてフォールトトレラントメカニズムを構築した．Primary-Backup 型の Replication システムにより，ROS の分散環境において一台のホストマシンで障害が発生した場合でも処理を継続できる仕組みを提案している．しかし，Lauer らの手法では，実際の分散基盤を想定しておらず，ホストマシンで起動している各ノードが利用している資源を管理していない．そのため，ROS の分散基盤上にある各ホストマシン間で処理の偏りが発生する場合があり，スケールアウトの観点から資源を有効に活用できていない．加えて，ホストマシン間の資源を管理せず処理を立ち上げることは資源不足による障害の可能性を高め，他のホストマシンに影響を及ぼすことも考えられる．

Shao ら [14] は，ネットワーク上にある複数の計算資源を一つの計算資源とみなして活用するグリッドコンピューティングの分散基盤において，Master-Slave 方式を採用した処理の割当てを実現した．Shao らは，異なる環境のホストマシンで構成された分散基盤に対し，ネットワークによる処理の転送時間や CPU の処理能力に対し重み付けを行い Master と Slave を決定し処理の管理を行った．一方，Shao らの手法が対象としている処理は，どの程度の計算処理を必要とするかが事前に分かり，それにより，ネットワークの転送時間と CPU による処理時間を算出している．そのため，ROS の処理のように事前に資源をどの程度使うか予測が難しいものに対してこの手法を適用することは困難である．加えて，各ホストマシンのメモリなどの資源は考慮されていない．効率的に資源を割当てるためには，CPU の処理能力だけでなく，処理割当て時点での CPU やメモリの利用率といった資源の余裕を考慮する必要がある．

6. おわりに

本研究では，ROS 分散環境における Master-Slave 方式の資源管理機構を構築した．Master で各 Slave が配信する CPU・メモリ・ディスクの利用率を監視し，最も資源に余裕のあるホストマシンに処理を割当てることを可能とした．加えて，資源の利用率が事前に定めた上限値に達した場合，効率よく分散基盤の資源を利用するため他のホストマシンに処理を移行する機能を実現した．処理の移行動作では，ステートのあるデータを Publish するという制約のもと，Master Node においてデータを Subscribe するためのトリガとなるトピックを配信した．このことにより，ステートフルなノードに関してもホストマシンをまたいだ処理の移行が可能となった．評価では，資源管理機構が提案手法で述べた動作となることを示し，各ホストマシン間の資源の偏りを防ぎ，処理の引き継ぎを達成した．加えて，効率的に処理を各ホストマシンに割当てることで ROS 分散環境においてスケーラビリティが確保されることを示した．さらに，実アプリケーション評価として分散環境における ROS の資源管理が必要な 3 次元地図の生成を行った．その結果，3 次元地図生成におけるステートを保ち移行することが可能であり，より大規模な地図生成に提案手法が活用可能となることを示した．

今後は，分散環境上の共有資源の資源管理機構の構築および，分散環境上での地図合成手法を提案する．地図合成手法では，Master において，複数台の車両から得られるデータを管理した上で Slave に処理を受け渡す．このことにより，近くの車両から得られる重複したセンシングデータの処理を避けるなどの効率の良い地図管理生成を行う．加えて，地図合成アプリケーションで負荷分散の効果とステートフルな処理の移行についての評価を行う．

参考文献

[1] Markoff, J.: Google cars drive themselves, in traffic, *The New York Times*, Vol. 10, No. A1, p. 9 (2010).
[2] Kato, S., Takeuchi, E., Ishiguro, Y., Ninomiya, Y., Takeda, K. and Hamada, T.: An Open Approach to Autonomous Vehicles, *IEEE Micro*, Vol. 35, No. 6, pp. 60–68 (2015).
[3] Geiger, A., Lenz, P. and Urtasun, R.: Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite, *In Proc. of 2012 IEEE CVPR*, pp. 3354–3361 (2012).
[4] Itseez: OpenCV, http://opencv.org/. Retrieved Jan. 11th, 2017.
[5] Open Perception: Point Cloud Library, http://pointclouds.org/. Retrieved Jan. 2nd, 2017.
[6] Rusu, R. B. and Cousins, S.: 3D is here: Point Cloud Library (PCL), *In Proc. of 2011 IEEE ICRA*, pp. 1–4 (2011).
[7] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A. Y.: ROS: an open-source Robot Operating System, *ICRA workshop on open source software*, Vol. 3, p. 5 (2009).
[8] O'Kane, J. M.: *A Gentle Introduction to ROS*, Independently published (2013).
[9] 内閣府：戦略的イノベーション創造プログラム（SIP）自動走行システム 研究開発計画，http://www8.cao.go.jp/cstp/gaiyo/sip/keikaku/6_jidousoukou.pdf (2016). Retrieved Dec. 10th, 2016.
[10] Pyo, Y., Kurazume, R. and Watanabe, Y.: *ROS Robot Programming*, Kurazume Laboratory (2015).
[11] Besl, P. J. and McKay, N. D.: Method for Registration of 3-D Shapes, *Robotics-DL tentative*, International Society for Optics and Photonics, pp. 586–606 (1992).
[12] Magnusson, M.: The Three-Dimensional Normal-Distributions Transform: an Efficient Representation for Registration, Surface Analysis, and Loop Detection, *Orebro university* (2009).
[13] Lauer, M., Amy, M., Fabre, J.-C., Roy, M., Excoffon, W. and Stoicescu, M.: Engineering Adaptive Fault-Tolerance Mechanisms for Resilient Computing on ROS, *In Proc. of 17th IEEE HASE*, pp. 94–101 (2016).
[14] Shao, G., Berman, F. and Wolski, R.: Master/Slave Computing on the Grid, *In Proc. of 9th IEEE HCW*, pp. 3–16 (2000).