fogcached-ros：ハイブリッドメインメモリ KVS サーバ

東晃希[1]　石綿陽一[2]　大川猛[3]　菅谷みどり[1]

概要：近年，ロボットをはじめとする IoT デバイスからのセンシングデータを大量に処理する目的から，クラウドより近距離におくエッジサーバを配置することが期待されている．近距離で動作するエッジサーバには高い即応性と信頼性が求められる．先行研究では fogcached として DRAM と Non-Volatile-Memory のハイブリット型メインメモリ上で In Memory Key-Value Store である memcached をデータキャッシュとして用いることで，応答性の改善を得た．ただし，これら結果はシミュレーションでしか評価されていないため，実際のアプリケーションにも有効であるかは明らかではない．このことから本研究では，エッジコンピューティングの即応性と信頼性を必要とする実用的なロボットアプリケーションとして SLAM (Simultaneous Localization and Mapping)を用いて fogcached の評価を行うことを目的とした．実現にあたり，fogcached を ROS と SLAM に適用するために，ROS からのデータを fogcached に送受信するための拡張機能である fogcached-ros を設計した．fogcached-ros は fogcached に ID を付けてデータを集約，また，データを取り出すための API を提供する．評価にて API オーバーヘッドが少なく，かつ SLAM が利用するデータに対応したことを示した．また，クラウドとエッジを接続し，fogcached-ros を評価し，応答性および信頼性の評価を行なったところ，エッジとクラウドのレイテンシ差が約 1ms であることから有効性を示すことができた．さらに，NVM にデータが保存されていることにより信頼性を示すことができた．

キーワード：In-Memory Key-Value Store, memcached, ROS, SLAM, Edge Computing

1. はじめに

近年，労力の削減を目的として倉庫や施設で稼働する自律移動ロボットが普及している．これらのサービスの特徴として，組織的な複数台ロボットの操作や，自律走行型ロボットが用いられる．例として，路面性状の測定ロボットや，警備ロボットがある．路面性状の測定ロボットは[1]，複数のドローンが上空から，路面を測定し，物体の位置や傾斜角度などを計算することで，人よりも正確かつ定期的に省コストで目的を達成する．さらに，計測したデータをリアルタイムに収集し地図生成を行い，環境測定に利用するなど，応用範囲も広い．また，警備ロボット[2]は，複数のロボットが同時に稼働し，不審者の発見や周辺の異常の検知，通報などを行う．また，これら 2 例以外にも，自律走行型配送ロボット[3]，Intel 社が開発した，複数の小型ロボットが連携して遭難者や行方不明者を捜索し，救助するシステム[4]など多くの利用がある．

こうしたロボットは，アクチュエーションと同時に大量のセンサデータの処理を行う．従来は物理情報の把握に用いられたセンサデータは，制御のために用いられることから，保存されず計算処理後は破棄されていたが，近年ネットワークに接続されたドローンなどによりロボットが撮影したデータがクラウドに送信されるといったサービス[5]では，大量のセンサデータのクラウドサービスへの保存が想定されるようになった．しかし，クラウドサーバへのデータ保存は，多数のロボットからのデータ送信が発生した場合，クラウドへの処理が集中し，クラウドサーバが過負荷になる点と無線等のネットワークの遅延が課題となる．

これに対して，近年，クラウドよりも近距離で，大量のデータを扱うためのエッジコンピューティングの研究がされている[6][7]．エッジコンピューティングでは，デバイスに近いエッジ側に存在するセンサーノードに対し，近距離でのリアルタイム応答を目的とし，大量のセンサーノードに対し応答性を重視した設計が提案されている[8]．応答性向上技術の一つとして，エッジにキャッシュを配備して，クラウドまでのデータキャッシュをエッジサーバ上で効果的に行う仕組みが提案されている．

小沢らは Key-Value Store の代表的な実装である，memcached をベースに，データの信頼性の向上と応答性の向上を目的に，NVM(Non-Volatile-Memory) メモリへの実装の拡張を行い，有効性を示した[9]．しかし，シミュレーションを使った評価での有効性に限定されており，実際のアプリケーションでの有効性は明らかでない．

そこで，本研究では fogcached のエッジコンピューティングとしての有効性を明らかにするために，エッジを用いた高い応答性と信頼性を要するロボットアプリケーションである SLAM(Simultaneous Localization and Mapping)[10]を用いた評価を行うものとした．SLAM は，ロボット等の移動体において，センサから環境情報を収集し，そのデータ解析により自己位置推定，環境地図作成を同時に行う技術の総称である．SLAM を活用することで，ロボットのみならず，自動運転など広く移動体の自律移動を実現できる[10]．これら SLAM の多くは，ミドルウエアである ROS(Robot Operating Systems)[10][11]上に実装されている．SLAM を用いた fogcached を利用するためには，現状では SLAM のそれぞれのアプリケーションを改変する必要があ

1 芝浦工業大学
Shibaura Institute of Technology
2 VA Linux Systems Japan 株式会社
VA Linux Systems Japan K.K.
3 東海大学
Tokai University

る．

これに対して本研究では，SLAM の下位レイヤである ROS ミドルウエアに fogcached との通信を行う API を設計・実装する．これにより，SLAM を改変せずに，ミドルウエアを通じて自動的に過去のデータをキャッシュサーバである fogcached 上の NVM に保存できるようにした. 本仕組みにより応答性向上と，データの信頼性の向上を実現する．fogcached と通信する ROS の拡張を fogcached-ros とした．本研究では，fogcached-ros の API を使用して，SLAM などの実用的な ROS アプリケーションを使用した評価を行った結果について述べる．

本論文では，研究の課題のための先行研究について 2 節で述べ，提案 3 節，予備実験を 4 節，API 評価を 5 節，クラウドとの接続評価を 6 節，まとめと今後の課題を 7 節にて述べる．

2. 関連研究

ハードウェア研究の成長に伴い，不揮発性メモリの速度が向上し，より DRAM に近いものが開発された．不揮発性メモリを以後 NVM(Non-Volatile-Memory)と表記する．Intel 社は 2019 年に DCPM(Intel Optane Data Center Persistent Memory)を発表した[12]. DCPM は DDR4 規格のメモリで，DRAM と比較して約 4 倍のレイテンシにおさまる[13].

Wu らは，NVMcached を提案した[14]．NVMcached は，NVM をベースにした Key Value Cache である．従来のコピーオンライトやジャーナリング等の一貫性を保つ技術は，KVS などの高速な処理を求められるアプリケーションにおいては重い処理であると考え，NVMcached として NVM の書き込み耐性に配慮した手法を提案した．具体的には，DRAM にオブジェクトのリスト(メタデータ)を保存し，NVM にオブジェクトを保存する．なお書き込み耐性に考慮して短時間で削除, 更新されるオブジェクトは DRAM に保存する．クラッシュ後に DRAM に保存されているリストは消失するが NVM に保存されているオブジェクトはクライアントからのアクセス履歴を反映したものとなっており，それを手掛かりにリストを再構築する．メタデータ処理を DRAM で高速化しつつ，NVM でオブジェクトの永続性を保証することで性能と信頼性の両立を目指した．この研究では不揮発性の memcached と比較して，書き込み集中型の実際のワークロードのシステムスループットを最大 2.8 倍向上させた．しかし，DRAM と NVM 間でデータが移動せず，保存データの重要度が増したときに NVM に移動しないという課題がある．

小沢らはエッジサーバでの大量データの保持に NVM を利用し，DRAM と NVM からなるハイブリット型メインメモリ上で In-Memory Key-Value Store を展開することを提案した[9]．また，評価にてエッジサーバの応答性を向上させたことを示した．小沢らは NVM として DCPM を用い，In-Memory Key-Value Store（KVS）として memcached[15]からアクセスできるようにした．小沢らが提案した memcached の拡張である fogcached は DRAM と DCPM をつなぐ構造を持つ．具体的には，memcached 内部の DRAM 用の LRU と同一構造を持つ LRU を DCPM 上に作成し，二つの LRU 構造（Dual-LRU）間のメモリオブジェクトの移動を MOVE というパイプラインでつないだ．メモリオブジェクトは，アクセス頻度が低くなると自動的に DCPM に移動することで，応答性を維持しつつ信頼性を実現した．memcached の既存実装である extstore に比べシステム全体のレイテンシは約 40%，スループットを約 19%向上した．しかし，信頼性については議論されておらず，不揮発性メモリの使用にとどまっている．

3. 提案

3.1 目的

これまでの研究から明らかになっている課題は，次の２つにまとめることができる. (A)実際のアプリケーションでの評価が不十分である, (B)信頼性を考慮したクラウドとの連携が実施されていない．そのため本研究の目的を，ロボットアプリケーション使用時のエッジサーバの高速性と信頼性を実現することとする. 課題(A)(B)に対し, 本研究の目的はロボットアプリケーション使用時のエッジサーバの高速性と信頼性を実現することである．

3.2 提案概要

本研究ではハイブリットメインメモリ KVS サーバに SLAM を対応させることを提案する．また，提案(1)として課題(A)に対して，ロボットを考慮した fogcached の設計と実装，提案(2)として課題(B)に対し，クラウドと連携するシステムの設計と実装を提案する．

3.3 ROS (Robot Operating System)

本研究では ROS[10][11]を用いる．ROS (Robot Operating System) は，ロボットアプリケーション作成を支援するオープンソースのミドルウェライブラリとツール群のことである．また，BSD ライセンス採用している．

ROS は現在，ROS1，ROS2 の 2 種類が存在する．ROS1 は ROS 初期版でユーザが多い．ROS2 は ROS1 の拡張版である．ROS2 は多種のロボットに対応した汎用性の高い ROS であり，複数のロボットのチームの想定，リアルタイムシステムなど近代のシステム要求を満たすものになっている．本論文の ROS は ROS1 を想定するものとする．

ROS では(a)node, (b)master, (c)message, (d)topic, (e)publish, (f)subscribe といった用語があり，以下で説明する．(a)node はコンピュテーションを行うプロセスである．ロボットの制御システムは通常多くの node で構成される. (b)master は ROS 通信全体の管理を行う. (c)message は node 同士の通信のデータである． 1 つの message は型フィールドからなる単純なデータ構造である. (d)topic は message の内容を区別

するための名前である．(e),(f)について，ROS は Pub/Sub 通信を行う．Pub/Sub 通信はイベントを処理するサービスとイベントを生成するサービスを切り離す非同期 message サービスである．

3.4 SLAM

SLAM(Simultaneous Localization and Mapping)[10]は，移動体の自己位置推定，環境地図作成を同時に行う技術の総称である．ロボットやセンサーノードからデータを受け取り，数値に基づき，自己位置を含めた地図を作成する．SLAM を活用することで，移動体が未知の環境下で環境地図を作成することができる．構築した地図情報を使って障害物などを回避しつつ，タスクを遂行する．本研究では，SLAM の中でも一般的に普及している gmapping[16]を利用する．gmapping はセンサ入力として 2 次元 LIDAR を想定した SLAM アルゴリズムである．特に陸上移動ロボットで最も広く使用されている SLAM 手法の 1 つである．SLAM は多様なセンサデータを扱い，多くのリソースを必要とする．そこで SLAM を評価するアプリケーションとする．

3.5 提案システム

提案システムの全体図を図 1 に示す．提案システムにはクライアントアプリケーション，エッジサーバ，クラウドが存在する．クライアント側のアプリケーションは既存のオープンソース ROS アプリケーションを使用して構築される．ROS アプリケーションとして turtlebot3[17]に SLAM アプリケーション(gmapping)を用いた．またエッジサーバには先行研究で実装した fogcached[9]を用いた．ROS アプリケーション(observer)はエッジサーバにてクライアントとデータの送受信，さらに fogcached との送受信を行う．クラウドはエッジサーバとのみ通信し，データを保存する．また，すべての通信を ROS コンポーネントで統一する．

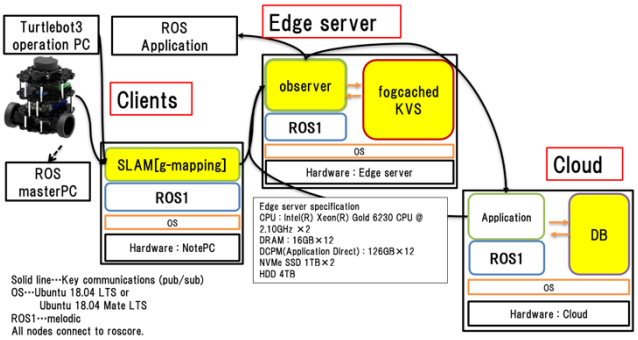

図 1　システム全体図

3.6 SLAM とサーバの連携の設計と実装

ROS は Node という通信単位に publisher/subscriber という役割を持たせ，topic という中間の共有メモリを介して非同期通信を行う（図 2）．図 2 に示したように，従来の ROS の通信モデルでは memcached と通信する手段はない．また，topic は基本的に Node により通信時に上書きされることから，最新の情報のみ保存する．本研究では topic を介して自動的に fogcached(memcached)と通信を行うための API を設計することで，過去のデータを自動で memcached 上の NVM に保存できるようにする．これによって利便性の向上のみならず，データの信頼性の向上が得られると考えた．また，ROS アプリケーション observer を設計し，observer_set, observer_get という node を構成できるようにした．図 3 は同じ topic にデータが戻されるフローを示したが，これは異なる topic にデータを出力することも可能な仕様となっている．これにより fogcached にデータを保存することで応答性を向上させるとともに NVRAM に保存することにより永続的にデータを保存することができるため信頼性を実現できると考えた．また，observer と fogcached (memcached)との通信は 2 種類の API(1)(2)によって行われる．インターフェースを図 4 に示す．詳細な動作は 3.7 で述べる．

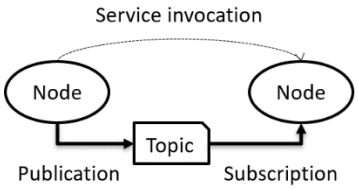

図 2　従来の ROS 通信

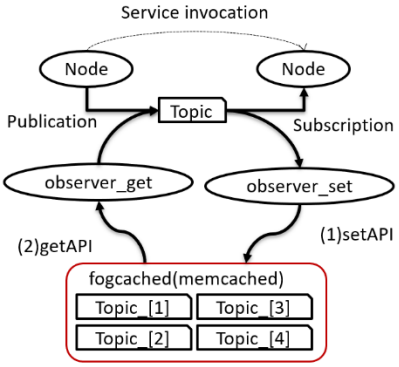

図 3　提案システムの ROS 通信

```
(1) int set_memcached(topic, T message, ID);
(2) int get_memcached(topic, T& message, ID);
```

図 4　API インターフェース

3.7 通信 API

図 4 のように，連携するための API を 2 種類作成した．図 3 に示したように ROS と fogcached (memcached)との通信は(1) setAPI，(2) getAPI で行う．これらは ROS node で実行されることを想定している．fogcached は KVS であるため，Key と Value を生成する必要がある．API はどちらも topic と ID を受け取る．ID はユーザが任意に設定する文字列であり，同じ topic のデータを識別する．setAPI は message も同時に受け取るが，getAPI の message は参照である．API インターフェースを用いた処理フローを図 5 に示す．どちらの API も topic と ID からアンダーバー(_)を挟んだ連携文字列として key を作成する．key は上限を 50Byte に設定している．setAPI の message は ROS の serialize を利用して

バイナリ文字列に変換される．value には message のバイナリ文字列が入るが，ヒープ領域が topic に応じて動的に割り当てられる．setAPI ではこれらの key と value は fogcached に保存される．getAPI ではバイナリから復元し，復元された message が返される．また，どちらの API も 1 つの型だけでなく，多くの topic に対応する．実際に，Subscriber の callback の中で実行する場合の API 呼び出しを図 6 に示す．ただし，一部の処理をコメントで省略している．

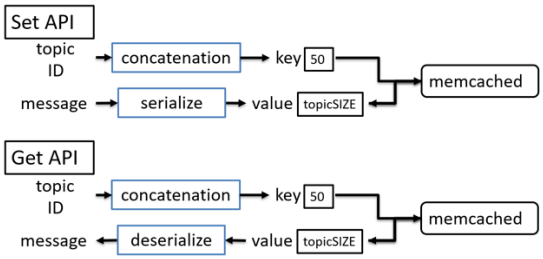

図 5　API 処理フロー

```
//callback function in Subscriber
void jointStateMsgCallback
(const sensor_msgs::JointState::ConstPtr &msg)
{
  char topic[TOPIC_BUFFER_SIZE];
  char id[ID_BUFFER_SIZE];
  // Substitute topic for key
  // Determine id arbitrarily
  set_memcached(topic,msg,id);
}
```

図 6　setAPI 例

4. 予備実験

4.1 予備実験概要

本実験は 3.2 提案(1)についてアプリケーションとして SLAM について，SLAM を実行したときの ROS 通信上のデータ量を調査することを目的とする．実験に使用した機材，またコンピュータのリソースと実行した ROS アプリケーションを表 1 に示す．

表 1　コンピュータのリソースとタスク

ROS Task	OS	Memory	HDD	PC
ROS master	Ubuntu 18.04 LTS	8GB	100GB	Windows10 Hyper-V Intel Core i7-8550U
SLAM gmapping				
rosbag record				
turtlebot3 bring up	Ubuntu Mate 18.04 LTS	1GB	32GB	Raspberry Pi 3B+
teleop (keyboard operation)	Ubuntu Mate 18.04 LTS	1GB	32GB	Raspberry Pi 3B+

ROS Task は turtlebot3 アプリケーションや ROS ツールである．実験は rosbag を使用し，すべての ROS 通信データを記録する．ROS 上の topic の流れを図 7 に示す．また，30 分間の測定であり，図 8 の約 23m^2 の環境で計測した．

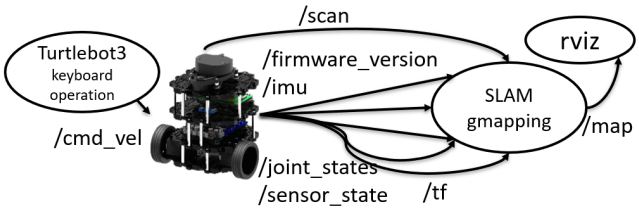

図 7　ROS 通信図

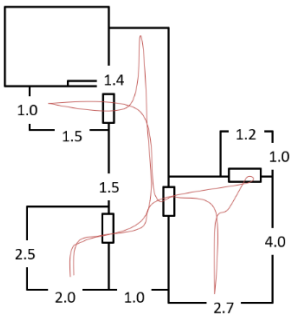

図 8　測定環境間取り(m)

4.2 予備実験結果

rosbag で記録した各 topic のデータ量を表 2 に示した．また，tf は 2 種類のデータサイズが存在した．

表 2　各 topic のデータ量

topic	size [byte]
map	147555
scan	2941
odom	718
tf_static	391
rosout	329
imu	320
diagnostics	275
tf	95, 205
joint_states	130
magnetic_field	120
map_metadata	76
battery_state	56
cmd_vel	48
sensor_state	44
turtlebot3_slam_gmapping/entropy	8
rpms	2

ROS 通信上のそれぞれの topic のデータ量はほとんど変動しなかった．このことから累積データ量は時間依存といえる．すべての topic の内，データ量に変化があった topic は tf と rosout だったが，これら二つの topic のデータ量変動はほとんどなかった．したがって，tf，rosout のどちらも

総データ量推移には影響しない．

総データ量の時間推移を図 9 に示した．SLAM で得られるデータの総量は時間に比例して増加した．また，30 分間で計測した累積データ量は倍増した．

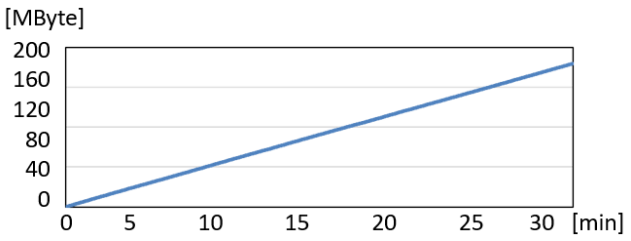

図 9　累積データ量の時間推移

4.3 考察

今回の予備実験では SLAM のデータ量は時間に比例して増大した．今回計測した空間は約 23m^2 と小さいが，実際には街，構造物での利用が想定される．この場合，計測時間に応じて，総データの基礎量は大きくなることが予測される．大量のデータ処理の応答性と信頼性の向上のためにエッジサーバのデータキャッシュを用いることは望ましい．また，SLAM アプリケーションの課題解決は，重要であると言える．

5. API 評価

5.1 評価方法

3.2 提案(2)中で使用される API を実装し，API オーバーヘッドを測定した．使用したクライアントアプリケーションは SLAM(gmapping)とシミュレーションソフトウェア gazebo 上で動かす turtlebot3 である．また，エッジサーバ上で fogcached と ROS アプリケーション observer を実行する．また，fogcached はバイナリプロトコルを有効にしている．クライアントアプリケーションを使用するコンピュータのリソースとタスクを表 3 に示す．

表 3　コンピュータのリソースとタスク

ROS Task	OS	Memory	PC
SLAM gmapping **teleop (keyboard operation)** **turtlebot3 in gazebo**	Ubuntu 18.04 LTS	8GB	Windows10 Hyper-V Intel Core i7-8550U
ROS master **observer**	Ubuntu Mate 18.04 LTS	DRAM: 196GB, DCPM: 1512GB	Edge server

また，表 4 にエッジサーバの仕様を示す．エッジサーバ上の fogcached は DRAM 4GB と DCPM 32GB の領域を使用する．ただし，キャッシュミスが発生した場合，deserialize は行われないため，オーバーヘッドの考慮から除外する．SLAM を 30 分間実行し，observer からの API オーバーヘッドを測定し，API を評価する．

表 4 エッジサーバの仕様

OS	DRAM	DCPM	CPU
Ubuntu 18.04LTS	16GB×12	126GB×12 AppDirectMode Device dax	Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz ×2

5.2 評価結果

各 topic の message データサイズを表 5 に示す．図 10 は setAPI のオーバーヘッドの割合と getAPI のオーバーヘッドの割合を示す．SET, GET は，libmemcached ライブラリ[18]を使用してデータを送受信するのにかかる時間である．serialize と deserialize は ROS に含まれている serialize による操作とした．Serialize と deserialize は両方とも実行するために必要な時間は 1ms 未満となった． さらに，setAPI，getAPI はどちらも，message データサイズとオーバーヘッドに正の相関関係がある．また，データサイズが大きくなると，API 全体の時間に比べて，serialize と deserialize にかかる時間の割合が増える．また，最大のデータである map は serialize に 629 μ sec，deserialize に 596 μ sec かかった．

図 11 は 1000 秒当たりの API オーバーヘッドの合計を示している．また，送信周期は表 5 のように topic によって大きく異なる．map は 1 秒間に 0.07 回しか送信されないため，1000 秒単位で考慮すると，map のオーバーヘッドは極僅かといえる．

表 5　各 topic のデータサイズと Publish レート

topic	data size [byte]	Publish per sec
imu	320	110.9729892
scan	2941	4.9117647
joint_state	130	23.2052821
tf	205	66.3295318
map	147555	0.0798319

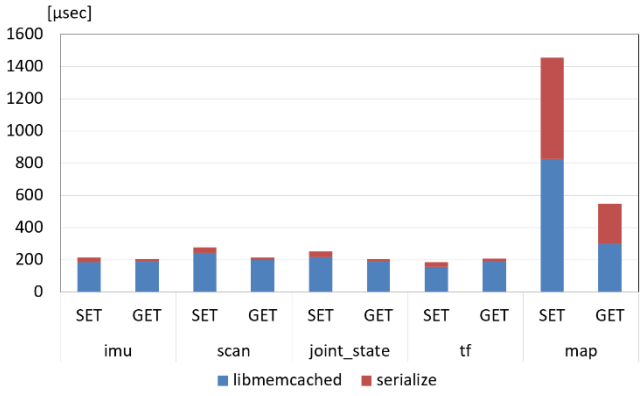

図 10　API オーバーヘッド 1 回

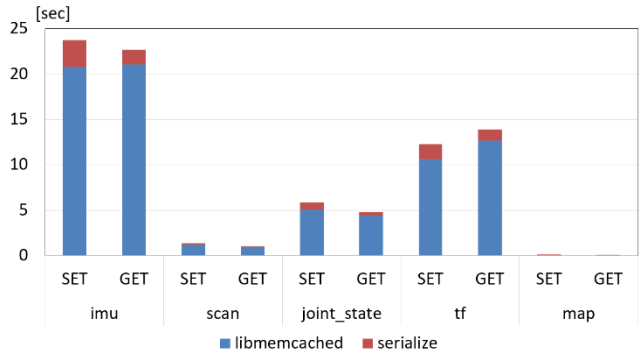

図 11　API オーバーヘッド　1000 秒

5.3 考察

Message サイズが非常に大きい場合，オーバーヘッドは serialize，deserialize の影響を受ける．ただし，約 1kB の message の場合，serialize，deserialize はオーバーヘッドにほとんど影響しない．したがって，使用時間が長くなるほど，時間当たりのオーバーヘッドが大きい map でも問題ないと考えられる．頻繁なものは imu のように大きなオーバーヘッドがあるが，送信頻度は API 側で操作できない，したがって，API に直接関係しないため，本論文では考慮しない．

6. クラウドとの接続評価

6.1 評価概要

Memcached と fogcached，fogcached-ros での違いを表 6 にまとめた．本研究の fogcached-ros は DCPM と ROS に対応し，さまざまな大きさの ROS データを扱う．

表 6　各研究の実験環境の比較

	DCPM	ROS	Data size
Memcached	No support	No support	Constant
Fogcached	No support	No support	Constant
Fogcached-ros	support	Support	Variable

本実験では 3.2 提案(2)のクラウド連携に対して，クラウドを想定した環境でのレイテンシ分布と DRAM と DCPM のヒット数の割合を調査するために条件を変えて実験する．実験条件を表 7 に記載する．実験条件のうち map_rate がほかの topic と同じ，ID の数を 1000000，set:get のレートの比を 1:9 とし，DRAM 4GB，DCPM 16GB という条件のものを標準とし，このパターンと比較してレイテンシの計測と Memory 別のヒット数の推移の計測を行い，考察する．実験は API 評価と同様，gazebo turtlebot3，SLAM，teleop をクライアントとして想定する．本実験ではクラウドはデータベースではなく，memcached1.5.16[15]を疑似的なクラウドとする．さらに，実際は複数台のロボットがデータを送信することを想定して，大量の ROS データを fogcached にデータキャッシュし，クラウドに保存するということを想定して実験を行う．

また，実験の概要図を図 12 に示す．クラウドを想定しているため，通信遅延を入れる．Observer はクラウドと通信し，データを送受信するが，今回は/topic_test という topic に publish するようにした．

表 7　実験パターン

Pattern	ID MAX	Set:Get	DRAM	DCPM	Map Pub
Standard	1000000	1:9	4GB	16GB	100%
Map_10%	1000000	1:9	4GB	16GB	10%
ID_change1	750000	1:9	4GB	16GB	100%
ID_change2	1250000	1:9	4GB	16GB	100%
Ratio1_1	1000000	1:1	4GB	16GB	100%
Ratio7_2	1000000	25:75	4GB	16GB	100%
DRAM6	1000000	1:9	6GB	14GB	100%
DRAM8	1000000	1:9	8GB	12GB	100%
DRAM10	1000000	1:9	10GB	10GB	100%

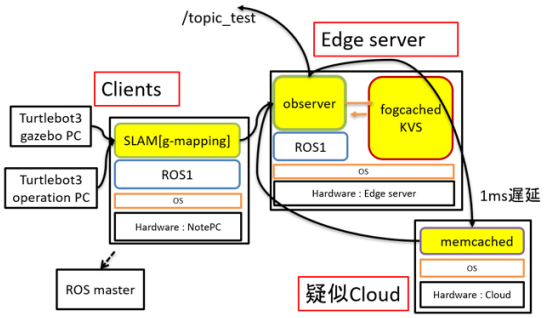

図 12　実験環境

6.2 評価結果

レイテンシはレイテンシ分布としてカーネル密度推定を用いてグラフにプロットした．実験で使用した topic は imu のほかに scan，joint_states，map があるが，これらはすべてのパターンにおいて，imu の結果と差異がでなかったため，本論文中では imu を参照する．

1)　Memory 比率を変えた場合

レイテンシ分布の結果を図 13，図 14 に imu と map を示す．Memory 比率を変えた場合，レイテンシ分布に Memory 比率に関連する変化は見られなかった．Map 以外の topic ではエッジとクラウドのレイテンシの差がはっきりと分かれた．しかし，map はエッジサーバとクラウドのレイテンシ差が imu ほどはっきりと出ず，レイテンシ分布の分散が大きいとみられる．

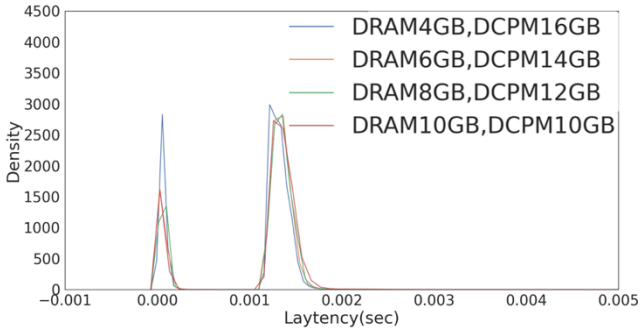

図 13　Memory 比率での比較(imu)

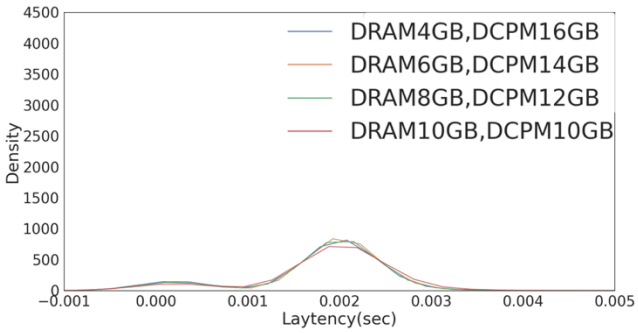

図 14　Memory 比率での比較(map)

さらに，DRAM と DCPM でのヒット数の推移を図 15，図 16，図 17，図 18 に示す．DRAM が増えるにつれ，hits する memory は DRAM の割合が高くなる時間が増える．しかし，いずれのパターンも最終的に DCPM でのヒットの割合が DRAM より約 80%高くなった．

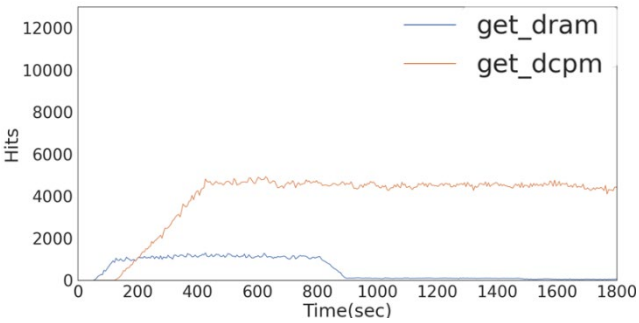

図 15　DRAM 4GB, DCPM 16GB でのヒット数の推移

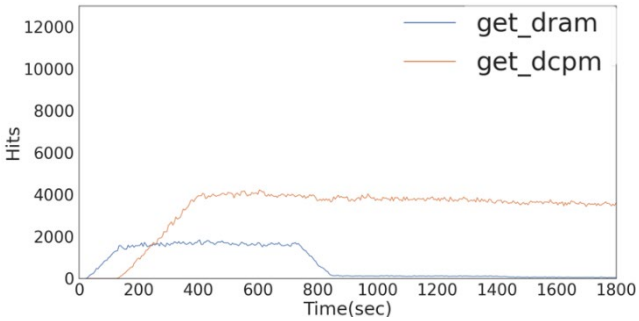

図 16　DRAM6 GB, DCPM 14GB でのヒット数の推移

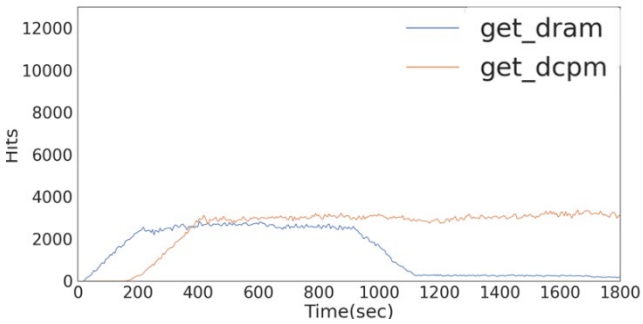

図 17　DRAM 8GB, DCPM 12GB でのヒット数の推移

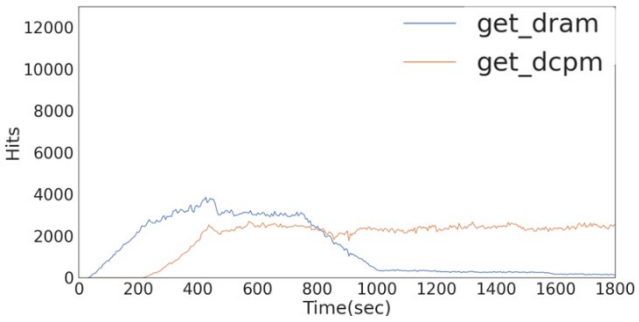

図 18　DRAM 10GB, DCPM 10GB でのヒット数の推移

2)　Map に対する操作のレートを変更した場合

レイテンシの分布は図 13 図 14 の標準と同様であった．標準のパターンと比較した場合，レート差による違いは見られなかった．また，map に関しては標準との差異は見られないもののレイテンシの分散が大きい．DRAM と DCPM のヒット数の推移は標準と変わらなかった．

3)　ID の種類を変えて実験した場合

imu，map のレイテンシ分布は図 13 図 14 の標準と同様の結果になった．ID が 750000 種類ある場合，標準に比べて，キャッシュに残りやすく，エッジでのアクセス数が多い．ID が 1250000 種類ある場合，標準に比べて，キャッシュに残りにくく，エッジでのアクセス数は少なくなる．この実験パターンでは map に対しても同様の結果になった．ただし，map はその差が少なかった．また，分散が大きい状態は先ほどのパターンと変わらなかった．どのメモリにヒットするかは ID の種類と関係が見られなかった．DRAM と DCPM へのヒット数の推移でこの実験による差は見られなかった

4)　set と get の割合を変更する場合

imu と map を．図 19 図 20 に示す．この結果は，set と get の割合が等倍に近づくにつれ，エッジサーバでのアクセスが増えることを示している．また，これについても set:get の比率を変えることと DRAM と DCPM へのヒット数の推移の差が見られなかった．

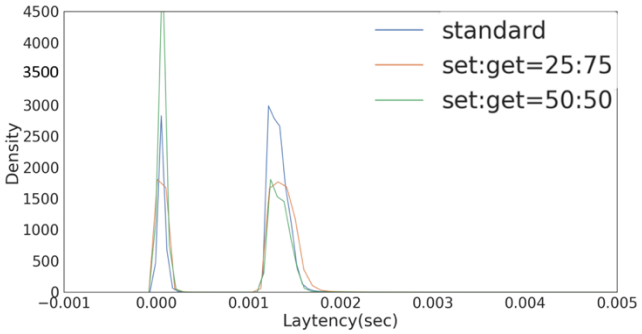

図 19　　set と get の比率の比較(imu)

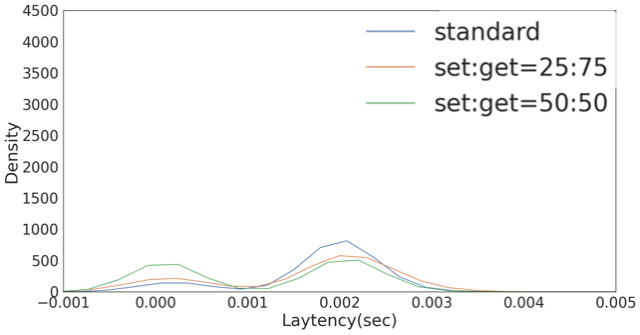

図 20　set と get の比率の比較(map)

6.3 評価考察

Memory 量の割合による変化が見られなかったことは，DRAM と DCPM の速度の差が nsec 単位であって，今回の実験は msec 単位の時間がかかることから，差が表に出なかったと考えられる．したがって，ROS アプリケーションにおいてはどちらのメモリでヒットしても　影響が少ないことが考えられる．また，DRAM のヒット数が一時的に増加することは DRAM の量が増えたことを踏まえると，当然の結果である．

また，ID の種類の増減はデータキャッシュにデータが残るかどうかに関連した．これはエッジサーバでのアクセスが増えていることを示すが，この結果は ID かぶりが多くなるということとトレードオフの関係にある．ID かぶりが多くなるということは fogcached に保存できる量は減るため，fogcached-ros として使用できる key の数に制約がうまれる．逆に多くの ID を使用することで fogcached に保存できる量が多くなるが，データキャッシュに残る量が減るため，エッジコンピューティングの効果が低くなる．

さらに，今回はすべてのパターンにおいて map のレイテンシ分布の分散が大きいという結果になった．エッジサーバでのこの結果から，map のような大きなデータはエッジコンピューティングの効果が低いということが考えられる．

7. まとめと今後の課題

本論文ではハイブリッドメインメモリ KVS サーバと SLAM を対応させることを提案した．予備実験では SLAM がエッジコンピューティングの評価に適していると判断したため，ROS を拡張して，fogcached と通信する API を作成し，オーバーヘッドを評価した．その結果 serialize のオーバーヘッドは 1ms 未満であったため，API は多くの topic に対応できると考察した．また，クラウドとの連携システムに向けて，クラウドを想定した応答性と信頼性の評価を行った．DRAM と DCPM の速度差によるレイテンシ差が少ないことで ROS 通信上ではどちらでヒットしても影響が少ないこと，大きなデータはエッジコンピューティングの効果が低い可能性があることがわかった．

今後の課題として，大きなデータに対して有効なエッジコンピューティングの効果を見出すことがあげられる．また，fogcached から押し出されるデータ量がどのくらいあるかどうかを明らかにすることでデータキャッシュとしてどのくらいデータがたまっているかを評価する必要がある．さらに ROS2 に移行することで汎用性の向上が考えられる．

参考文献

[1] V.A.Knyaz, A.G.Chibunichev.. Photogrammetric Techniques for Road Surface Analysis, ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2016, Vol. XLI-B5, p. 515-520

[2] “Security Robot System”. https://smprobotics.com/security_robot/robot-security-system/ (参照 2020-09-28)

[3] Dr Khasha Ghaffarzadeh, Dr Na Jiao.. Mobile Robots, Autonomous Vehicles, and Drones in Logistics, Warehousing, and Delivery 2020-2040”, IDTechEx Research.

[4] Vinayak Honkote, Dileep Kurian, Sriram Muthukumar, Dibyendu Ghosh, Stish Yada, Kartik Jain, Bradley Jackson, Ilya Klotchkov.. Distributed Autonomous and Collaborative Multi-Robot System Featuring a Low-Power Robot SoC in 22nm CMOS for Integrated Battery-Powered Minibots, IEEE International Solid-State Circuits Conference. 2019, p. 48-50.

[5] “FlytBase”. https://flytbase.com/ (参照 2020-09-29)

[6] Michele De Donno, Koen Tange, Nicola Dragoni.. Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog. IEEE Access. 2019, Vol. 7, p. 150936-150948

[7] Jie Yuan, Xiaooyong Li.. A Reliable and Lightweight Trust Computing Mechanism for IoT Edge Devices Based on Multi-Source Feedback Information Fusion. IEEE Access. 2018, Vol. 7, p. 23626-23638.

[8] Kashif Bilala,b, Osman Khalidb, Aiman Erbada, Samee U. Khan.. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. Computer Networks. 2018, Vol. 130, p. 94-120.

[9] Kouki Ozawa, Takahiro Hirofuchi, Ryousei Takano and Midori Sugaya.. DRAM-NVM Hybrid Memory - Based KVS server for Edge Computing. International Conference on Edge Computing. 2020.

[10] Lum, J.S.. Utilizing Robot Operating System (ROS) in Robot Vision and Control. National Technical Reports Library U.S Department of Commerce. 2015.

[11] “ROS”, https://www.ros.org/ (参照 2020-08-04)

[12] “Intel®Optane™PersistentMemory”. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html (参照 2020-08-04)

[13] Takahiro Hirofuchi, Ryousei Takano.. A Prompt Report on the Performance of Intel Optane DC Persistent Memory Module. IEICE Transactions on Information and Systems. 2020, Vol. E103.D(5), p. 1168-1172.

[14] X. Wu, F. Ni, L. Zhang, Y. Wang, Y. Ren, M. Hack, Z. Shao, and S. Jiang, “NVMcached: An NVM-based Key-Value Cache”. in Proceedings of the ACM SIGOPS Asia-Pacific Workshop on Systems, pp. 1–7, August.2016.

[15] “memcached” http://memcached.org/ (参照 2020-08-04)

[16] Jiun Fatt Chow, Basaran Bahadir.. Toward Underground Localizatio: Lidar Intertial Odometry Enabled Aerial Robot Navigation. IEEE IROS 2019 workshop, Vol. 1.

[17] “turtlebot3-emanual” https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (参照 2020-9-29)

[18] “libmemcached”, https://libmemcached.org/libMemcached.html (参照 2020-9-29)