ROSとの相互運用性に配慮した 共有メモリによる低遅延プロセス間通信フレームワーク

○入江 清 (千葉工業大学 未来ロボット技術研究センター)

1. 緒言

近年，ロボット向けミドルウェアである Robot Operating System(ROS) [1] が広く用いられている．ROSの中核をなす機能はネットワークベースのモジュール（ノード）間通信であり，送信側と受信側が同一のコンピュータで動作しているか，ネットワークで接続された分散コンピュータで動作しているのかを意識せず透過的に通信を行うことができるという利点がある．しかし，このようなネットワークベースのモジュール間通信にはある程度の伝達遅延が避けられないという欠点もある．

筆者らは状況によりこの遅延が無視できない大きさになることを指摘し，これを回避するため共有メモリ上に構築した循環バッファを介してプロセス間通信を行う手法を提案する．筆者らはこの通信フレームワークを fuRom と呼ぶ．fuRom の概要を図 1 に示す．

新たに独自のフレームワークを開発する上での課題として，既存のソフトウェア資産が再利用できないことや開発者への新たな教育が必要になることがある．そこで，fuRom では ROS のデータ型を直接扱うことができ，ROS と類似した送受信 API を持たせることとした．そのため ROS モジュールと fuRom の間で相互にデータをやり取りしたり，ROS 向けに書かれたプログラムを fuRom へ移植したりすることが比較的容易である．

本稿では，fuRom の設計及び実装の詳細について述べる．また，PC 及び組み込みコンピュータ上で行った通信遅延時間の評価結果について報告する．

2. 関連研究

2.1 SSM

共有メモリによるプロセス間通信をロボット向けに実装した例として，竹内らによる Sensor Sharing Manager (もしくは Streaming data Sharing Manager, SSM) がある [2] [3]．SSM では同一コンピュータ上で複数の機能モジュールが独立したプロセスで動作するアーキテクチャを想定し，共有メモリ上に循環バッファを構築することでモジュール間の通信を実現する．また，異種トピック間の時刻同期を支援する機能も提供する．

fuRom の基本的な構成は SSM と同一であるが，主な差異として，

- データの既読管理機能が提供される
- プロセス間の排他制御が実装されており，複数のモジュールから同一のトピックに書き込みができる
- ヘッダオンリーライブラリとして実装され，依存ライブラリは Boost のみであるため移植性が高い

といった点が挙げられる．

2.2 アドレス渡しによるプロセス内通信

ROS のメッセージ通信では，同一のプロセス内ではデータをアドレス渡し（`shared_ptr`）で送信すること

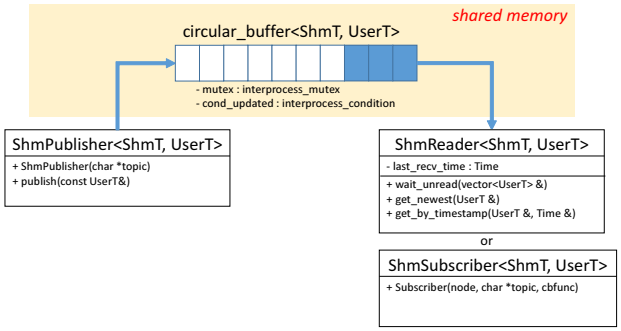

図 1 開発したプロセス間通信フレームワークの概要．

が可能である．この場合はデータ本体の複製が行われないため，通信にかかるオーバーヘッドが小さい．これを活用できる枠組みとして ROS では nodelet と呼ばれる機能が提供されている．nodelet に基づいたモジュールは同一プロセスにて複数動作させることができるため，アドレス渡しによるモジュール間通信を利用することができる．

この nodelet によるアドレス渡しでは低遅延の通信が実現できるが，複数モジュールが同一ポインタにアクセスすることになるため，アクセス競合の問題が生じる．その回避のため，排他処理やデータの複製処理を追加する必要性が生じ，結果として通信以外でのオーバーヘッドが増加することも考えられる．また，同一のプロセスで複数のモジュールを動作させることにはメモリ破壊のリスクがある他，関数名などのシンボルや環境変数が共有されてしまうことによる不便さもある．

2.3 DDS

ROS では独自仕様のメッセージ通信が用いられているが，OMG により規格化された出版購読型メッセージ通信として Data Distribution Service (DDS) がある．DDS では共有メモリによるプロセス間のデータ転送も規定されており，これを用いて通信の遅延を低減できる可能性がある．

DDS は自律移動ロボットにおける利用例 [4] も報告されているが，筆者らの経験では DDS は仕様が膨大で API も複雑であるため，直接扱うのは容易ではない．ROS の次世代バージョンである ROS2 ではメッセージ通信のバックエンドとして DDS を採用しており [5]，現行の ROS に比べ通信遅延の改善が期待できる．

ただし，DDS の実装の全てが共有メモリによる転送をサポートするわけではない．また，共有メモリを用いてもデータの直列化など他のオーバーヘッドを削減しなければ通信遅延の低減にはつながらない可能性がある点に注意が必要である．

2.4 OpenRTM における例

OpenRTM においても共有メモリを用いたモジュール間通信の遅延低減の試みが報告されている．清水ら

によって提案された手法 [6] は ART-Linux の共有メモリ機能を用いて CORBA 通信を置き換えることで通信遅延の低減を図るものである．

3. 開発したプロセス間通信の概要

fuRom はトピックで管理される出版購読型のメッセージ通信を提供する．ここでは API およびバックエンドの実装について述べる．

3.1 送受信 API

送信側の API として，ShmPublisher クラスを提供する．これはテンプレート引数にデータ型を，コンストラクタ引数にトピック名を与えて実体化する．送信メソッドは

```
void publish(const UserT &data)
```

であり，メッセージを一つ配信する．

受信側はアプリケーションをプログラミングする上での自由度を考慮し，2 種類のクラスを提供する．一つは ROS のプログラミングスタイルと合わせた ShmSubscriber であり，データが到着する度にコールバック関数が呼び出されてデータを受信する．

このコールバック関数による受信スタイルの欠点は，送信側のデータ配信周期に比べ，受信側でデータを必要とする周期が遅い場合に，本来は不必要な関数呼び出しによりリソースを消費してしまうことである．

受信用のもう一つのクラスは ShmReader であり，よりアプリケーション開発上の自由度が高く，任意のタイミングでデータを取得することができる．ShmReader の主要なメソッドとして

```
size_t wait_unread(const vector<UserT> &data)
```

があり，これは未読データが届くまでブロッキングで待った上でメッセージを取得する（オプション引数でタイムアウト時間を指定できる）．メッセージの未読・既読は ShmReader の内部でタイムスタンプによって管理される．その他のメソッドとして，既読・未読にかかわらず最新のデータのみを取得する

```
bool get_newest(const UserT &data)
```

や，他のトピックとの同期のため時刻を指定して近いデータを取得する

```
bool get_by_timestamp(const UserT &data,
                      const Time &time)
```

などが提供されている．

ここまでに述べた基本的な送信及び受信のサンプルコードを図 2 に示す．

3.2 バックエンド

3.2.1 循環バッファ

fuRom ではデータの送受信を共有メモリ上に構築された循環バッファを介して行う．この循環バッファには読み出し及び書き込み時にプロセス間ミューテックスを用いた排他が実装されており，データの整合性を保ちつつ複数のプロセスから読み込み・書き込みを行うことができる．

fuRom は循環バッファ初期化時に必要な全ての共有メモリを確保し，実行中の共有メモリ確保を行わない．これは設計上，システムの堅牢性を重視するため，実行中の共有メモリ枯渇を防ぐためである．反面，制約

```
// publish data
typedef ShmPublisher<furom::ShmLaserScan,
    sensor_msgs::LaserScan> LaserScanPublisher;

LaserScanPublisher pub("laser_scan");
while(true) {
  sensor_msgs::LaserScan scan;
  .. // prepare data
  pub.publish(scan);
}
```

```
// receive data by ShmReader
typedef ShmReader<furom::ShmLaserScan,
    sensor_msgs::LaserScan> LaserScanReader;

LaserScanReader reader("laser_scan");
std::vector<sensor_msgs::LaserScan> vec;
while(true) {
  if (reader.read_unread(vec) > 0) {
    ... // process data
  }
}
```

```
// receive data by ShmSubscirber
typedef ShmSubscriber<furom::ShmLaserScan,
    sensor_msgs::LaserScan> LaserScanSubscriber;

void cbfunc(const sensor_msgs::LaserScan &msg)
{
  ... // process data
}

void main() {
  furom::Node nh;
  LaserScanSubscriber sub(&nh, "laser_scan",
      cbfunc);
  nh.spin();
}
```

図 2　fuRom によるデータ送信/受信の例.

として共有メモリに配置するデータは固定長である必要がある．

ただし，ユーザには固定長の型は隠蔽され，API のレベルでは可変長の型（ROS で定義された型，あるいは任意に定めた型）を扱うことができる．共有メモリへの書き込み・読み出し時に変換・逆変換を行うことにより実現する（単純なスカラー値など，本質的に固定長のデータは変換しない）．この変換は通信上のオーバーヘッドとなるため，利便性・相互運用性とのトレードオフである．また，型ごとに変換を実装する必要があり，これは今後の課題である．

3.2.2 更新通知

前述の通り，ShmReader には新しいデータが届くまで待つという `wait_unread()` メソッドが提供されている．この待ち時間に CPU 時間を消費しないよう，条件変数を用いた待機及び更新通知が実装されている．このシーケンスを図 3 に示す．

各循環バッファは共有メモリ上に条件変数（Boost の `interprocess_condition`）を持つ．受信側の `wait_unread()` 呼び出し時に未読データが無い場合，`interprocess_condition` の `timed_wait()` によって待機する．一方，送信側は新しいデータを循環バッファに追加する毎に `notify_all()` を呼び出す．これを契機に待機中の全ての ShmReader は `timed_wait()` から戻り，処理を再開してデータを読み出す．

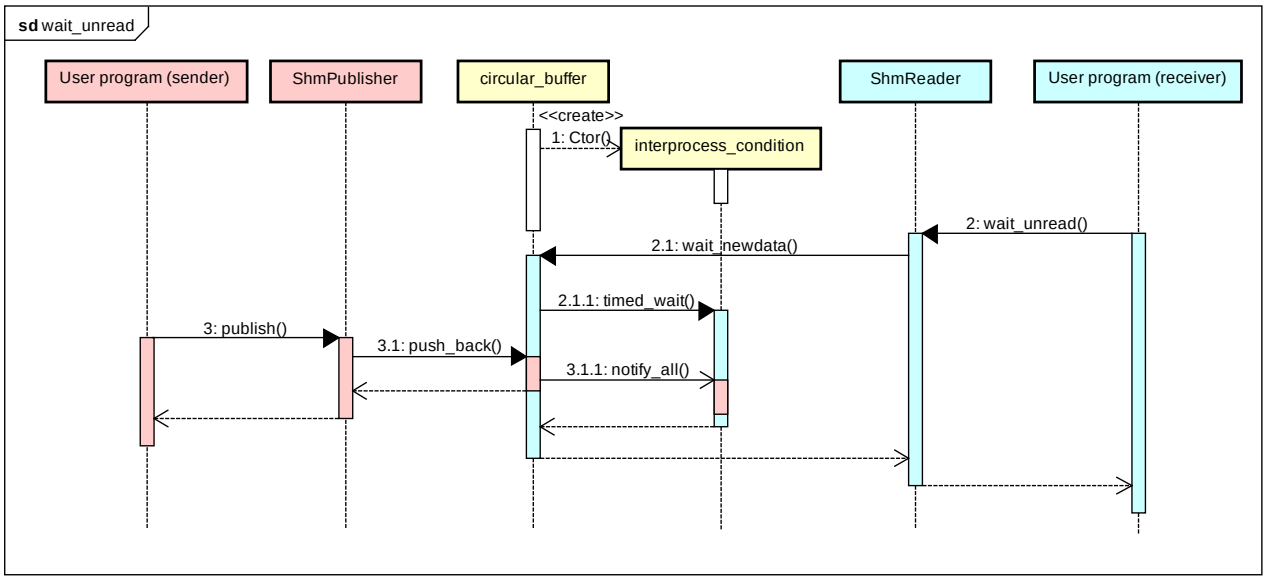

図 3　データの更新待ちと通知シーケンス．

4.　評価

提案手法による通信遅延評価をノート PC（Thinkpad X240）及びシングルボードコンピュータ（Raspberry Pi 3）上で行った．評価に用いたコンピュータ環境を表 1 に示す．

通信するデータとしては，Imu（約 40Bytes），LaserScan（約 8KBytes），OccupancyGrid（約 1MBytes）の 3 種類のデータを用意し，それぞれ 2000Hz, 100Hz, 10Hz の周期でモジュール間通信を行った．送信側にてメッセージに送信時刻をタイムスタンプとして付与し，受信側で実際にデータを受信した時刻との差を通信の遅延時間として測定した．fuRom での受信には ShmReader を用いた．

比較対象として ROS の nodelet として送受信ソフトウェアモジュールを実装し，2 つの独立したプロセスで動かして通信を行わせた場合と，同一のプロセスで動かしてデータをポインタ渡しする場合とそれぞれでデータ通信にかかる遅延時間を測定した．

各手法による通信を 10 秒間行った際の平均遅延時間と標準偏差を表 2 に示す．提案手法では全ての環境及びデータについて，ROS によるプロセス間通信よりも遅延が小さく，また，Imu 及び LaserScan データでは ROS でのアドレス渡しによるプロセス内通信よりも遅延が小さかった（いずれも Wilcoxon の順位和検定において $p < 0.001$）．

fuRom による通信遅延は送受信データサイズに応じて増大する傾向が見られる．これをより詳細に調べるため，OccupancyGrid の格子数を様々に変化させて送受信を行い，遅延時間を測定した．測定結果のプロットを図 4 に示す．この図から fuRom と ROS ではいずれも比例的に通信遅延時間が増大しているが，増加の傾向は fuRom の方が小さいことが見て取れる．

なお，ROS のプロセス間通信では LaserScan データの場合に目立って大きな通信遅延が観測された．筆者らの実験では，数キロバイト以下のデータを 100Hz 前後で送信する場合に，複数のデータがまとめて受信されることにより，通信遅延のばらつきが非常に大きくなる．この問題に対してはデータサイズを大きくする，あるいは送信周期を上げるまたは下げると改善が見られる．

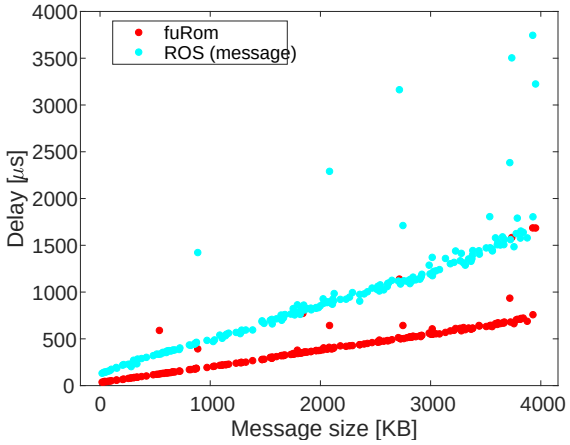

図 4 プロセス間通信における遅延時間のデータサイズによる変化．測定にはデスクトップ PC（CPU: Core-i7 6950X）を用い，OccupancyGrid データを 10Hz で送信して測定した．

5.　結言

本論文では，共有メモリを用いたプロセス間通信フレームワーク fuRom の設計と実装を紹介した．データの授受に共有メモリを用い，更新通知に条件変数を用いることで，通信遅延を低減する．ROS の共通のデータ型と ROS に類似した API によって，ROS のソフトウェア資産の活用と，開発者教育コスト低減を目指している．Boost のみに依存するヘッダオンリーライブラリであるため移植性が高く，Linux，Windows 及び Mac OS での動作を確認している．

実際的な設定で通信遅延の評価を行い，ROS が提供するプロセス間通信に比べて通信遅延が小さいことを確認した．

今後の拡張の余地として，C++以外の言語バインディ

表 1　評価に用いたコンピュータ環境

Computer	CPU	System oftware
Thinkpad X240	2.1GHz Intel Core i7 4600U Dual Core	Linux 3.13.0-123-lowlatency (64bit), ROS indigo
Raspberry Pi 3	1.2GHz ARM Cortex-A53 Quad Core	Linux 4.4.38-v7+ (32bit), ROS kinetic

表 2 モジュール間通信の遅延時間 [μs]．fuRom および ROS node はプロセス間通信，ROS nodelet はプロセス内のアドレス渡しによる通信である．

		fuRom	ROS (message)	ROS (pointer)
Thinkpad X240	IMU, 2000Hz	**3.9 ± 15.8**	72.3 ± 20.9	18.3 ± 58.9
	LaserScan, 100Hz	**7.7 ± 3.6**	23527.7 ± 11411.7	22.3 ± 7.0
	OccupancyGrid, 10Hz	316.4 ± 66.1	1121.7 ± 116.4	**24.8 ± 18.9**
Raspberry Pi 3	IMU, 2000Hz	**29.7 ± 10.2**	364.7 ± 72.9	54.4 ± 15.3
	LaserScan, 100Hz	**45.1 ± 10.2**	19419.9 ± 11223.5	60.1 ± 27.1
	OccupancyGrid, 10Hz	3068.5 ± 127.3	9010.0 ± 246.9	**212.4 ± 77.3**

ング，分散コンピュータ間での通信などが挙げられる．またソフトウェアの開発効率を高めるためには，プロセス間通信データの可視化やロギングなどのツールも重要である．現在これらのツールの整備も進めており，詳細については別稿にて紹介したい．

謝辞

本論文で紹介した ROS プロセス間通信の遅延に関する知見の一部は NEDO 委託事業「環境・医療分野の国際研究開発・実証プロジェクト／ロボット分野の国際研究開発・実証事業／災害対応ロボット研究開発 (アメリカ)」を通して得られたものである．

参　考　文　献

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng,“ROS: an open-source robot operating system”, ICRA Workshop on Open Source Software, 2009

[2] 竹内 栄二朗，坪内 孝司，油田 信一，“移動ロボット用センサ情報処理ミドルウェアの開発”, ロボティクス・メカトロニクス講演会，2P2-3F-C6, 2003

[3] 竹内 栄二朗, “汎用 PC を用いた知能移動ロボット開発” 日本ロボット学会誌, Vol. 31, No. 3, pp.236–239, 2013

[4] 入江 清, 吉田 智章, 小柳 栄次, 友納 正裕, “つくばチャレンジ 2011 に向けた長距離ナビゲーションシステム”, 第 12 回計測自動制御学会システムインテグレーション部門講演会, 101-2, 2011.

[5] “ROS 2.0 Design”, `http://design.ros2.org/`

[6] 清水 昌幸，石綿 陽一，尹 祐根，加賀美 聡，“AMP 版 ART-Linux の共有メモリを用いた RT コンポーネント間のデータ通信の実現”，SI2012，2G3-4, 2012