DSSM(Distributed Streaming data Sharing Manager)

DSSM(Distributed Streaming data Sharing Manager)

郡司凌太 †, 福田浩章 †
Ryota Gunji†, and Hiroaki Fukuda†
† 芝浦工業大学　工学部
†Faculty of Engineering, Shibaura Institute Technology

要旨

ロボット市場の拡大により，自律移動ロボットの需要が増加している．汎用 PC で複数のセンサを利用して自律移動ロボットを構築するには, 複数プロセス間でのセンサデータ共有や取得したデータの時刻を考慮する必要がある．それを踏まえ, SSM ではセンサデータだけでなく取得した時刻データを付与して, 共有メモリに保存することでこれらの問題を解決している．しかし, センサデータの処理量が汎用 PC の CPU の能力を超えると, ロボットの動作が遅延しリアルタイム性が失われる．そこで本研究では, ネットワークを介した共有メモリへのアクセスを可能にした DSSM の提案と実装を行う．DSSM では, 別の PC の共有メモリに保存されたセンサデータの読み出しと書き出しが可能である．これにより, 1 つの CPU が処理するセンサデータの読み書きの要求が減少し, システム全体の遅延も解消されることが期待される．

1. はじめに

近年, ロボット市場は加速度的拡大している [1]．それに伴い, 様々な種類のロボットが開発されている．また, ロボットの開発を支援するために, Robot Operating System(ROS)[2] や OpenRTM-aist[3] などのフレームワークが世界中で開発されている．本研究では自律移動ロボットのプログラムで使用されている, Streaming data Sharing Manager(SSM) に着目する [4]．SSM は, メモリの一部を複数のプロセスが参照できる共有メモリとして確保し, その領域へのセンサデータ書き込みと読み込みの手段を与えるソフトウェアである．自律移動ロボットでは, 各部品の同期のために同時刻に取得したデータが必要になる．SSM では, センサデータをメモリに書き込むときに時刻を付加することによって時刻の指定によるデータの取得を可能としている．これにより, センサが自身で時刻データを付与しなくても, SSM が管理する共有メモリのデータを参照することで, 同時刻にセンサが取得した一連のデータを取得できる．SSM は 1 台の PC での動作を想定しているため, SSM に接続されているセンサが増加して処理する書き込みと読み出しのリクエストが増加すると, CPU の処理が遅延し, 自律移動ロボットの動作に悪影響を及ぼしてしまう．

そこで本研究では, これまで 1 つの PC で構成していたシステムを, 複数の PC で構成することで CPU の負荷を分散する Distributed Streaming data Sharing Manager(DSSM) を提案する．DSSM では, SSM が動作する複数の PC をネットワークで接続し, PC を指定してデータを読み書きする仕組みを実現する．これにより, 1 つの共有メモリが保存するセンサデータを減らすことが可能になり, 1 つの CPU が処理する読み書きのリクエストも少なくすることができるようになる．結果として, CPU の負担が小さくなるのでシステムの遅延が解決できると考えられる．

2. アーキテクチャ

本項では, 最初に SSM のアーキテクチャと実際にデータを書き込むまでの手順を説明する．その次に, DSSM のアーキテクチャとデータを書き込むまでの手順を説明し, SSM と比較しどこが変更されたのかを解説する．同時に, 共有メモリの分散化によって発生する同期の問題点と解決方法を示す．

2.1.SSM

図 1 が SSM のアーキテクチャを示す．SSM を用いたアプリケーションは, ssm-coordinator(coordinator) とセンサハンドラで構成される．coordinator はセンサの書き込みリクエストに応じてメモリの空き領域を提供し, どの領域をどのセンサが利用しているかを記憶し管理する．また, 共有メモリへの書き込みと読み出しのリクエストもセンサの情報を管理している coordinator を介して行われる．

センサハンドラの役割はプログラムが必要とする読み書きのリクエストを coordinator に伝えることである．センサハンドラは coordinator と通信する手段として, SSM は SSMApi クラスを備えている．SSMApi クラスはあらかじめ決められたメッセージを利用して coordinator と通信を行い, センサハンド

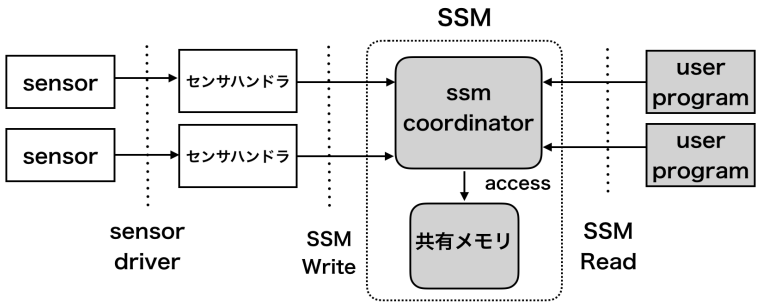

図 1: SSM のアーキテクチャ

ラの要求を実行する．一般的に自律移動ロボットでは多くのセンサで構築されるが, 多くのセンサハンドラがシステム内に存在すると, coordinator は受信したメッセージがどのセンサハンドラのものかが分からなくなってしまう．そこで, SSM ではセンサハンドラと coordinator との間にメッセージを送受信するためのキューを作成する．このキューを使用してメッセージを送り合うことでどのメッセージがどのセンサハンドラに対応するかを判別する．

次に実際にどのように共有メモリからデータの書き込みと読み出しがどのように行われているかを説明する．最初にセンサハンドラはデータの書き出しメッセージを, 事前に作成したキューを使用して coordinator に送信する．coordinator は送られてきたセンサのデータが書き込まれている共有メモリ上の番地を検索し, センサハンドラにそのメモリ番地を返す．センサハンドラは受け取ったメモリ番地にデータを書き込む．読み出しのリクエストも最初に事前に作成したキューを用いて, 読み出しのリクエストを coordinator に送信する．coordinator は書き込みのときと同様にセンサを検索し, そのデータが書き込まれているメモリ番地をセンサハンドラに返す．センサハンドラは受け取ったメモリ番地からデータを読み込むことができる．また, coordinator からセンサハンドラを切断したい場合は, キューから coordinator に切断のメッセージを送信することで, coordinator からセンサの情報が消去される．

2.2.DSSM

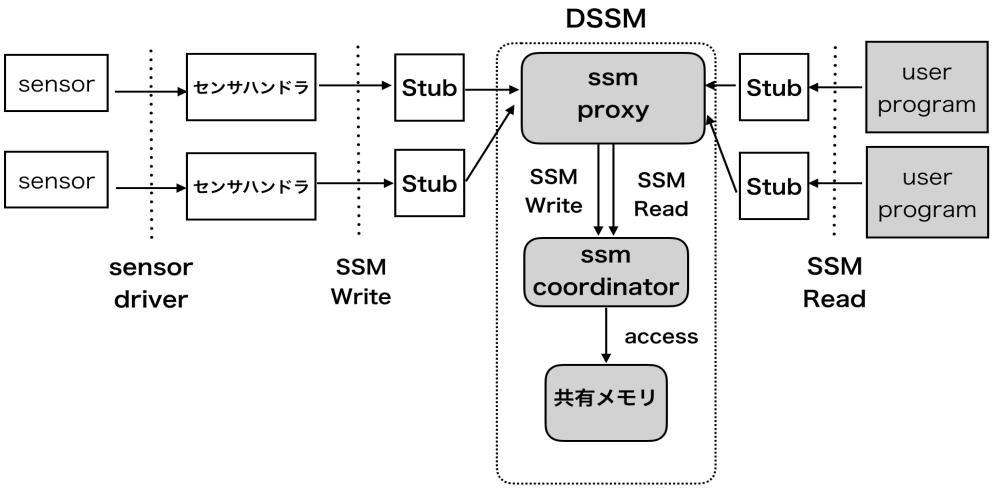

図 2: DSSM のアーキテクチャ

図 2 に DSSM のアーキテクチャを示す．DSSM ではネットワークを介したメモリアクセスを実装する．第 1 節でも述べたように SSM にはネットワークを介した共有メモリへのアクセス機能はない．そのため DSSM では, センサ情報を管理する coordinator をサーバ, サーバに接続するセンサハンドラをクライア

ントとして, ソケット通信と TCP/IP を用いてネットワークアクセスを実現する. SSM と同様に DSSM でも複数のセンサによるシステム構築を実現したいが, クライアントがサーバに接続するとソケットが埋まってしまい, 2 つ目以降のクライアントが接続できないという問題が発生する. センサ情報は全て独立であるので, サーバのプロセスをフォークすることを考える. フォーク後, 親プロセスは自身の待ち受けソケットを開放し, 再び新しいクライアントからの接続を待つようにする. 子プロセスは, サーバに接続することができたので, この通信経路を使用してクライアントが発行したメッセージをサーバに送信する.

次に DSSM におけるセンサデータの書き込み処理を考える. DSSM の共有メモリにセンサデータを書き込むときに, SSM と同様にセンサのメモリ番地を渡されても直接その場所にデータを書き込むことはできない. したがって, ネットワークを介したメモリへの書き込み処理の実現のために以下の 3 つの手段を考えた.

1, サーバからクライアントに接続をする. 2, 最初に確立した経路を使用する. 3, クライアントからサーバへデータ通信用の経路を新たに作成する.

1 つ目は, データの通信が必要になったときにサーバがクライアントに接続する方法である. ただこの方法では, クライアントの IP アドレスがサーバが所属するグループとは別のプライベート IP アドレスであった場合, クライアントと接続することができなくなってしまうという問題が考えられるため不適である.

2 つ目は, 最初に確立したサーバとクライアント間の経路を使用する方法である. しかし, これは 1 つの経路にメッセージとセンサデータが混在することになる. サーバ側は受け取ったデータがメッセージかセンサデータかを判別しなくてはいけないので, 複雑な処理が要求される.

3 つ目は, クライアントがサーバにもう 1 つの経路を作成して, その経路を使ってデータを送信する方法である. これは 1 つ目の方法と接続を待つ方向が逆になったものである. この方法であれば, クライアントがサーバとは別のプライベートネットワークに所属していてもサーバにセンサデータを送信できる.

以上 3 つの方法のうち, すでに確立された接続をメッセージの通信のために使用したいので, DSSM ではサーバに新しい通信経路を作成する 3 つ目の方法を採用する. 3 つ目の方法でサーバと接続するときに, 最初の接続と同様にプロセスをフォークしてしまうと, 1 つのセンサを扱っていたプロセスが親プロセスと子プロセスの 2 つのプロセスに分岐してしまう. 1 つ 1 つのプロセスは独立であるため, クライアントから送られてくるメッセージに対応した処理を, メッセージを受信するプロセスがデータを受信するプロセスに指示する必要がある. これを防ぐため, サーバのプロセスにクライアントから新しい接続を待ち受けるソケットを生成する. 新たに生成したスレッドで通信を待ち受けることで同じプロセス内でメッセージとデータの 2 つの経路を分けて動作させることができる.

以上のことを既存の SSM に実装するために, 新しく ssm-proxy(proxy) と ssm-proxy-client(client) というプログラムを実装する. proxy はこれまで述べたサーバ側の処理, client はクライアント側の処理を実装する. この 2 つのプログラムを実装せずに直接 coordinator や SSMApi クラスを変更することも考えたが, これまで正常に動作していた SSM を変更するのは既存の処理に未知のバグを埋め込んでしまう恐れがあるので回避した. client は SSMApi クラスのインターフェースを殆ど変更せずに, ネットワークを介したメモリアクセスに対応した PConnectorClient クラスを実装する. これによって, 既存のプログラムを殆ど変更せずにネットワーク通信に対応した DSSM に置き換えられるというメリットも存在する. proxy は client から受け取ったメッセージを見て, それに対応する coordinator の処理を呼び出すプログラムである. この仕組みにより, coordinator は proxy からセンサの情報を受け取るので, 内部の処理を変更せずに別の PC 上に存在するセンサのデータをメモリに書き出すことができる. しかし, DSSM では複数の PC を使用するので, coordinator が受信したデータをそのまま共有メモリに記録すると, 時刻のずれが生じる可能性がある. この問題を回避するために, DSSM では Network Time Protocol(NTP) を利用して時刻データを付与する.

3. 評価

DSSM はすでに実装済みであるので, 実際に SSM で構築していた自律移動ロボットを DSSM に置き換えて動作するかを確認する．その際, TCP/IP を使用したセンサデータの送信にかかる時間と共有メモリからデータを取得して動作するまでの時間を計測し, リアルタイム性が重要なシステムでも動作可能かを評価する. 想定通りの動作を確認できたら, システムに接続するセンサを増やしてこれまで発生していた遅延が解消されているかを調査し評価する.

4. 今後の課題

システムにはセンサや動作を決定するプログラムだけではなく, データを記録する Logger などのプログラムも存在する. 現状の DSSM ではデータを取得するには coordinator に読み出しのリクエストを送る必要がある. しかし Logger などのデータを受け取るだけで良いプログラムは, coordinator へデータをリクエストするのは手間である. このようなプログラムが増えると coordinator へのアクセスが増加してしまう. この解決策としてセンサハンドラが proxy にデータを送信する際にマルチキャストでデータを送信することを考えている. 特定のセンサからの情報が必要なプログラムはこのマルチキャストグループに所属することで coordinator を介さずにデータを取得できるようになる. だがマルチキャストは UDP を使用するので送信するデータの欠損が考えられる. そのため, データの送信側の負担をできるだけ少なくした代替手段を現在考えている.

参考文献

[1] 株式会社三井住友銀行コーポレートアドバイザリー本部企業調査部, “産業用ロボット市場の動向, ” 2018.

[2] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew Ng, “ROS: An open-source robot operating systems, ” Proc. ICRA Open-Source Softw. Workshop, 2009.

[3] Noriaki Ando, Takashi Suehiro, Tetsuo Kotoku, “A Software Platform for Component Based RT-System Development: OpenRTM-Aist, ” SIMPAR2008: Simulation, Modeling, and Programming for Autonomous Robots pp.87-98, 2008.

[4] 竹内栄二郎, “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 31(3), pp.236-239, 2013-04-15.