Vol.2021-ARC-246 No.5
2021/10/11

制御コントローラ向けコンテナ及び TSN 活用技術の研究

金成昊[1]　山本秀典[1]　飯島光一郎[1]　大塚祐策[1]　清水俊樹[1]

概要：複数のコントローラで構成される分散制御システムにおいてはコントローラ間のデータ共有のリアルタイム性向上のためにリアルタイムネットワーク上で実現される分散共有メモリ技術がよく使われている．しかし，制御ロジックをコンテナ上で運用する制御コントローラではコンテナのネットワーク処理のオーバヘッドによって分散共有メモリ上でのデータ共有のリアルタイム性が阻害される．また，1 台のコントローラ内で複数のコンテナが存在する場合，その性能劣化は顕著である．本研究ではコンテナのネットワーク処理のオーバヘッドを削減するため，コントローラ内のホスト OS 上で分散共有メモリ処理を実現し，コンテナからそのメモリをアクセスする分散共有メモリ方式を提案した．また，ホスト OS 上で分散共有メモリ処理における複数のコンテナのデータ更新に必要な送信処理のリアルタイム性保証のため，TSN(Time Sensitive Network)の送信スケジューラを活用し，その送信周期をコンテナ毎に論理的に分割する論理リング送信方式を提案した。これらの提案方式によって，複数のコンテナをもつ制御コントローラ間のデータ共有処理の性能揺らぎを 3 分の 1 に抑制した．

キーワード：制御コントローラ，分散共有メモリ，コンテナ，TSN

Study on Container and TSN technology for Industrial Controller

SUNGHO KIM[†1]　HIDENORI YAMAMOTO　KOICHIRO IIJIMA
YUSAKU OTSUKA　TOSHIKI SHIMIZU

1. はじめに

近年，制御システムにおいては現場機器の制御技術に AI 技術及びデータ利活用技術を取り入れ，制御の高度化を目指している．そのために制御コントローラにおいては既存制御システムの要件であるリアルタイム性を保証するとともに，AI 技術をはじめとする新技術の導入がしやすくするためにコンテナ技術を適用する事例が多くなってきた[1]．一方，複数のコントローラで構成される分散制御システムにおいてはコントローラ間のデータ共有のリアルタイム性向上のためにリアルタイムネットワーク上で実現される分散共有メモリ技術がよく使われている[2]．しかし，制御ロジックをコンテナ上で運用する制御コントローラではコンテナのネットワーク処理のオーバヘッドによって分散共有メモリ上でのデータ共有のリアルタイム性が阻害される．また，1 台のコントローラ内で複数のコンテナが存在する場合，その性能劣化は顕著である．この問題を解決するため，我々は，制御コントローラ上のコンテナ間のリアルタイム性を保証するデータ共有のために TSN(Time Sensitive Network) [3][4]の活用技術の研究に取り組んだ．

本論文では，2 章で制御コントローラにおける分散共有メモリの概要を述べる．次に，3 章では制御コントローラ上で運用されるコンテナ間での分散共有メモリの実現における課題及び対策について述べ，4 章ではコンテナ間での分散共有メモリの実現における論理リング送信方式について述べる．5 章で提案した方式の評価について述べる．最後に 6 章で本論文の内容を纏める．

2. 制御コントローラにおける分散共有メモリ

複数台の制御コントローラで構成される制御システムの場合，全コントローラ間のデータ共有は分散共有メモリ方式で実現している[2]．分散共有メモリ方式は，各コントローラの内部で持っているデータ共有用メモリから共有したいデータを読み出し，コントローラ間に接続されているネットワーク上に送信する処理と，他コントローラから送信されたデータを受信し，データ共有用メモリに書き込む処理で構成される．なお，ネットワーク通信によるデータ共有におけるリアルタイム性保証のため，リアルタイムネットワーク技術が要求される[3]．近年，制御システム間のリアルタイムネットワークの汎用性や互換性を実現するために，TSN(Time Sensitive Network) [3][4]が注目されている．以下，分散共有メモリと TSN の概要を記す．

2.1 分散共有メモリ概要

図 1 のように多くの制御システムの場合，Ring ネットワーク構成を採用している[2]．分散共有メモリ方式による制御コントローラ間データ共有処理の概要を示す（図 1）．

- 各コントローラ（以後，ノードと併記）はデータ共有

1 (株)日立製作所
Hitachi Ltd.

メモリを持ち，コントローラ上のアプリが制御処理に必要なデータの書き込みを行う

- データ共有メモリには、全ノードで共有されるデータを格納する領域を用意している
- 各ノード上のアプリは自ノード用に割り当てられたメモリの更新を行う
- 自ノードのメモリ内容及び他ノードからのメモリ内容をネットワークから送受信するプロセス（図 1 の Tx/Rx Process）が存在する
- 送受信するプロセスは周期的に自ノードのメモリ内容を送信する(図 1 の n1)
- 他ノードからの受信データは自ノードのデータ共有メモリに書き込みすることで全ノードのデータを共有する

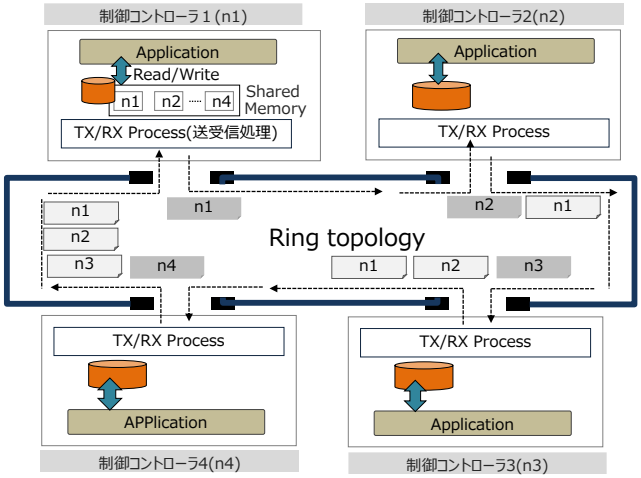

図 1　分散共有メモリ

2.2 TSN 概要

近年，制御システムにおける制御ネットワークはイーサネット集約の傾向が顕著である[3]．そこで，汎用イーサネットを用い制御システムに必要なリアルタイム性や信頼性を実現する TSN(Time Sensitive Network)技術が注目されている[3][4]．TSN はイーサネットベースのリアルタイムネットワークの IEEE 802.1Q の規格群である[8][9][10]．その特徴として，ネットワーク Traffic のリアルタイム性保証のためパケット転送スケジューラの Traffic Shaper 技術と TSN のスイッチを含む TSN ネットワークの構成技術と信頼性確保技術が挙げられる[11]．これらの技術によって TSN ネットワークに接続されている機器間での determistic な通信レイテンシを保証することが可能である．

TSN のリアルタイム性の保証する中核の技術はパケット送信のスケジューラ技術である．TSN の規格では，データ送信のリアルタイム性の保証のために多数の送信スケジューラの規格定まれている[11]．送信データの特性にあった送信スケジューラの選定が必要である．制御システムの分散共有メモリにおけるデータ伝送は周期的な送信の特定を考慮すると，TAS(Time Aware Shaper)が適切である[5]．

以下，TAS の概要を記す．

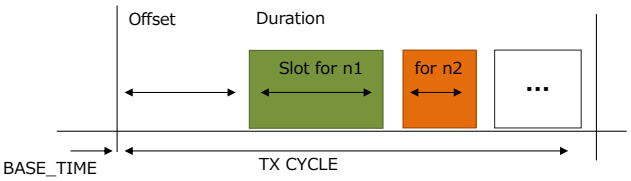

図 2　TAS 概要

TAS は送信時間を各送信パケットでタイムスロットをシェアするスケジュール方式である．ある特定の送信パケットに占有のタイムスロットを割り当てることで他パケットの送信処理の干渉を抑制ができ，送信のリアルタイム性を保証する．図 2 には，各送信パケットに割り当てるタイムスロットの概要を示している．

・TX CYCLE: 送信周期.
・Offset：送信周期のはじめから実際の送信までの送信時間オフセット
・Duration: タイムスロット幅．この幅内での送信許可
・BASE_TIME：送信開始の絶対時刻．TSN のすべてのノードは時刻同期をしていて TSN 上の送信開始時間は絶対時刻で設定する[5]

3. コンテナ間分散共有メモリ

制御システムにおいても情報技術との連携やビックデータ活用技術，AI，IoT 化への柔軟な対応が要求されている[7]．既存制御ロジックとともに AI 技術等のデータ利活用技術を制御コントローラにも運用可能にするコンテナ技術が導入されている[1]．本章ではコンテナを運用する制御コントローラ間の分散共有メモリの実現における課題及びその解決策について述べる．

3.1 課題

コンテナ上のアプリから活用可能な分散データ共有処理の実現における課題を述べる．

(1) 同ノードのコンテナ間データ共有

コンテナ内の仮想ネットワークを用いたデータ共有はオーバーヘッドが大きい[12]．

(2) 他ノード上のコンテナ間データ共有

他ノードのコンテナでデータ送信を行う場合，通常はコンテナ内の仮想ネットワークを経由してホストの物理 NIC デバイスを使う必要がある．仮想ネットワーク処理のオーバーヘッドに加え，仮想ネットワークからホストの物理 NIC デバイスへの接続方法は，コンテナの作成ツール毎に

異なるため，多様な仮想ネットワークに対するリアルタイム性保証の対策案を検討する必要がある．

(3) TSN を活用するためのソフトウェアインタフェース

TSN 対応の NIC デバイスの機能を使うために提供されているソフトウェアインターフェースは socket レイヤのコントロールメッセージで実装されているため，実際にネットワークデバイスレイヤにはコントロールメッセージが伝達しないため，仮想ネットワークを使う場合は TSN 機能が使えない[12].

3.2 アプローチ

前述した課題を解決するために，仮想ネットワーク使わない分散データ共有処理を実現する．

(1) コンテナ上のアプリが使う分散共有メモリはホスト上で実現し，コンテナ上のアプリと共有する．同ノード内のコンテナ同士は該当メモリを共有するだけでよい

(2) ホスト上の分散共有メモリを読み取りし，他ノードへ送信する処理及び他ノードからのデータを該当分散共有メモリへ書き込む受信処理はホスト上で実現する．ホスト上の送受信処理は直接，TSN 対応の NIC デバイスへアクセス可能であるため，仮想ネットワークによる処理オーバーヘッドは削減できる

(3) ホスト上で送受信処理を実現することで，TSN を活用するためのソフトウェアインタフェースを使うことが可能である．

分散データ共有処理機能を実装方法について図 3 に従い説明する．

(S1) 全ノードで共有する分散共有メモリ

各ノードのホスト上で共有メモリとして実装する．本メモリは全ノードのデータを格納するためのメモリであり，各ノード用のデータブロックは，後述する(L1) 分散共有メモリのライブラリ(Library for DSM)を通してコンテナ上のアプリが指定する

(S2) データメモリ内のデータを周期的に送信する送信デモンプロセス

一定周期で前述した(S1)のデータを読み取り，TSN 通信で他ノードへ送信する．送信対象のデータは後述する(L1) 分散共有メモリのライブラリを利用してコンテナ上のアプリが指定する．

(S3) 他ノードからの送信データを受信するデモンプロセス

他ノードからの送信データを受信し，後述する送信データフォーマットから送信先のノード ID を読み取り，更新対象の(S1)上のノードデータを指定し，更新する

(S4) TSN の送信スケジューラのパラメータ設定ファイル

TSN の規格では，データ送信のリアルタイム性の保証のために多数の送信スケジューラの規格が定まれている[11]. 送信データの特性にあった送信スケジューラの選定が必要である．分散データ共有処理は周期的な送信が必要であるため，TAS が適切である．TAS は送信時間を各送信パケットでタイムスロットをシェアするスケジュール方式である．ある特定の送信パケットに占有のタイムスロットを割り当てることで他パケットの送信処理の干渉を抑制ができ，送信のリアルタイム性を保証する．そのため，TAS を活用する送信プロセスは，指定タイムスロット内で送信するように,TAS のパラメータを共有する必要がある.また,TAS は，Linux カーネル 5.0 以上で Default で提供されているため，後述する提案方式の実装では，Linux カーネル 5.0 をサポートする Ubuntu18.04 を使用する[5]．以下，TAS を利用するためのパラメータを示す．

・TX CYCLE：送信周期は現実装では，ハード転写の最小周期である「1ms」とする

・Priority：使用する NIC デバイスの送信キューの指定

・Time Window：送信周期内で設定するタイムスロットの数．Time Window として本実装では，

・PTP 時刻同期用送信：TSN 上のすべてのノード間時刻同期プロセス用

・分散データ共有処理用送信

・その他データ送信：システム制御用 TCP/UDP 等

を TSN ネットワークにて使用する．

ただし，本設定ファイルは，前記の (S2) 送信デモンプロセスが送信周期生成及び TSN の送信データパケットを作成する際に必要なパラメータを提供する[5].

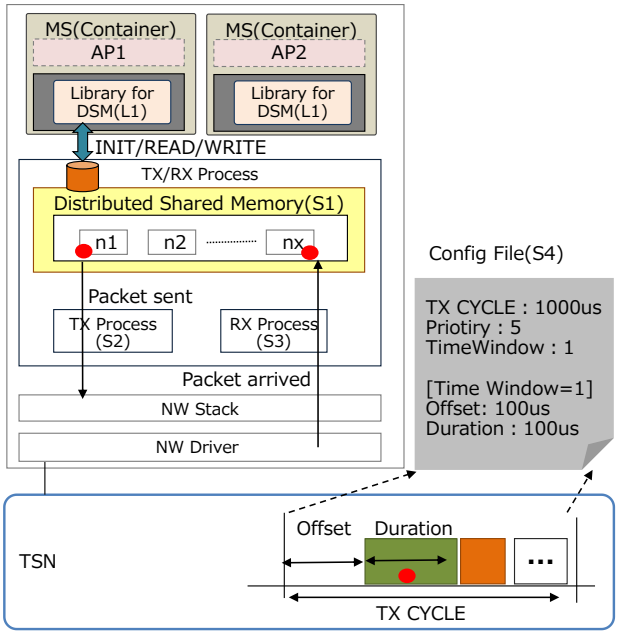

図 3　コンテナ間分散共有メモリ実現方式

(L1)　分散共有メモリ用ライブラリ

コンテナ上のアプリがホスト上で作成される (S1) 全ノ

ードで共有する分散共有メモリへアクセスするためのライブラリである．コンテナの作成時にコンテナ内に本ライブラリを含む必要がある．

3.3 送信デモンプロセスの実装方法

分散データ共有処理のリアルタイム性の実現のために，前述のように，TSN の送信スケジューラを活用する．そのため，TSN の送信スケジューラが許容しているタイムスロット内で，送信周期生成及び送信パケットの作成が必要となる．本節では，前述した (S4) TSN の送信スケジューラのパラメータ設定ファイルを用いた送信デモンプロセスの実装について述べる．

（1）通信フォーマット

TSN は L2 レイヤの EtherFrame を取り扱う通信であるため，ユーザプロセスである送信デモンプロセスは，送信用 socket として RAW SOCKET を使い，TSN 用の EtherFrame を直接操作し，パケット作成を行う．以下，送信デモンプロセスで使用する通信フォーマットを示す（図 4）．図 4 の「Ether Header」において，TSN 関連ビットの詳細を以下に示す．

・13 バイト目から 18 バイト目に,TSN 規格の 802.1Q の tag header である．

・～16 ビットまで：TPID(Tag Protocol Identifier)

・次の 3 ビット：PCP(Priority Code Point) TSN の優先度設定場所

・次の 1 ビット：CFI(Canonical Format Identifier) 正規 etherframe かを指定

・次の 12 ビット：VID(VLAN ID)

図 4 の「Pay Load」におけるデータ構成を以下に示す．送信データは 64 バイトを 1 つブロックとするブロック単位で構成する．

・node_id: 送信先のノード ID

・total_blknum：送信ブロック数

・blk_data[]：送信データブロック配列

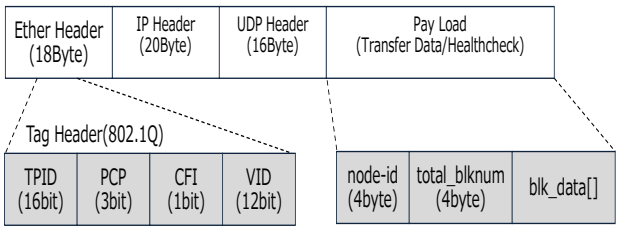

図 4　データフォーマット

（2） 送信処理の実装

図 5 に従い，送信処理の詳細を述べる．

(T1) 図 5 の TSN の送信スケジューラの TAS のパラメータ設定ファイルからの TAS パラメータを読み取り，パケット送信時間を設定する．ただし，最初の送信開始時間（図 2 の BASE_TIME）は前述した全ノードの PTP 時刻同期が終了した時刻を設定する必要がある[5].

(T2) 送信するデータを図 3 の分散共有メモリから読取り，図 4 の「Pay Load」へ設定する．なお，図 4 の「Tag Header」へ TSN 送信用 Priority 及び VLAN ID を設定する．ここで VLAN ID は，前述した PTP と同様な ID とする．設定コード例は以下に示す．

・rawpkt[14] = (Priority << 13 | VLAN_ID) >> 8

・rawpkt[15] = (Priority << 13 | VLAN_ID) & 0xFF

(T3) UDP 送信を行う．一回の送信パケットのサイズは通常のネットワークの default MTU である 1500 バイト以下である．

(T4) (T1)で読み取った TAS のパラメータで，次の送信時間を算出する

(T5) 次の送信周期を待つ．

・clock_nanosleep(CLOCK_TAI, TIMER_ABSTIME, 次回周期までの時間, NULL)

TSN の時刻同期の絶対時刻を使うため，クロックソースは「CLOCK_TAI」を使用し，wakeup の時刻は,「TIMER_ABSTIME」により，設定した「次回周期までの時間」を既に，経過した場合は，sleep せずに次の送信を行うようにする．

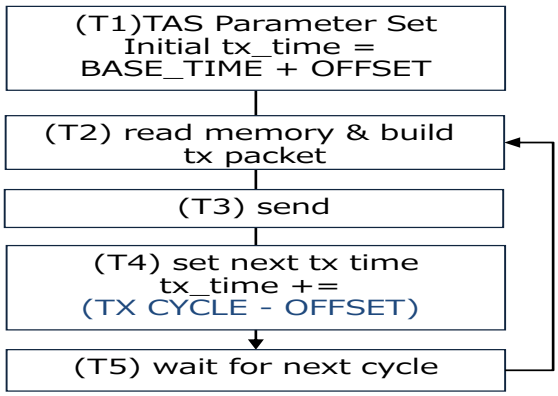

図 5　送信処理概要

以上，制御コントローラ上のコンテナ間でのデータ共有を行う分散データ共有処理を TSN の送信スケジューラを用いた実現方法について述べた．一方，TSN を用いるネットワークの Topology はスター型が広く使われている．スター型のネットワーク場合，ネットワークスイッチを中心に各ノードが接続されている．ネットワーク上の全ノードから送信データがネットワークのスイッチに集まり，スイッチによって送信先へ送信されていく．その場合，スイッチ内で他ノードからの送信パケットによる干渉を防ぐために，各ノードに占有の送信タイムスロットをスイッチ内に設ける必要がある．しかし，TSN の規格上，占有タイムスロッ

トは最大８個まで設定できない[3]．制御システムにおいては 16 台以上のコントローラ構成が必要なシステムが多く，各コントローラノードからの送信パケットのリアルタイム性保証のために，各ノードに占有タイムスロットの設定が必要となる．この問題を解決するために，本研究では，１つのタイムスロットに複数のコントローラノードがシェアしながら送信パケットのリアルタイム性も保証できる論理リング送信方式を提案し実装を行った．その詳細は次章にて詳細を述べる．

4. 論理リング送信方式

前述した分散データ共有処理において，送信処理のリアルタイム性保証のため，TSN の送信スケジューラを活用した．本章では，16 台以上のコントローラ構成が必要な制御システムにおいても TSN を用いたリアルタイム性を保証した分散データ共有処理を可能とする論理リング送信方式について述べる．

4.1 課題

TSN を用いた分散共有メモリを実現する場合，TSN の規格上，占有タイムスロットは最大８個まで設定できないため，16 台以上のコントローラ構成の場合，各コントローラノードからの送信パケットのリアルタイム性保証のための占有タイムスロットの数が足りない．この問題を解決するために，本研究では，１つのタイムスロットに複数のコントローラノードが共有しながら送信パケットのリアルタイム性も保証できる論理リング送信方式を提案し実装を行った．以下にその詳細を述べる．

4.2 論理リング送信方式の概要と実装

(1) 概要

従来の制御ネットワークは Ring 状のネットワークで，データの送信はあるノードから次にノードへ順次に行われる．その送信特性を活用すると，一つの送信タイムスロットを複数のノードで共有することが可能である．具体的に，前述した各ノードの送信デモンプロセスは１つのタイムスロット中で設けられた論理的なリング型のタイムスロットに従い順序に送信すれば，TSN の送信スケジューラによる送信処理のリアルタイム性の保証が可能である（図 6）．

TSN のスイッチは，一つのタイムスロットで認識して，該当タイムスロット内に到着した送信パケットの送信処理のリアルタイム性は保証する．論理的なリング型のタイムスロットは，各ノードの送信デモンプロセスで認識する仮想的なタイムスロットであり，前述した図 3 の (S4) TSN の送信スケジューラのパラメータ設定ファイルの「Time Window」パラメータで送信周期及び送信タイミングを設定することとする．

＜TSN スイッチ内送信スケジューラのパラメータ＞

・TX CYCLE：送信周期は現実装では，ハード転写の最小周期である「1ms」とする

・Priority：使用する NIC デバイスの送信キューの指定

・Time Window：送信周期内で設定するタイムスロットの数．本実装では，

・PTP 時刻同期用送信：TSN 上のすべてのノード間時刻同期プロセス用

・分散データ共有処理用送信

・その他データ送信：システム制御用 TCP/UDP 等

を TSN ネットワークにて使用する．

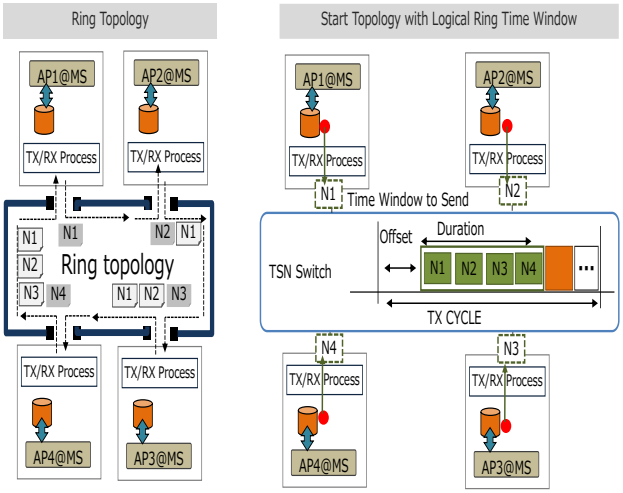

図 6　論理リング方式

＜各ノード上の送信デモンプロセス用送信スケジューラのパラメータ＞

・TX CYCLE：上記 TSN スイッチと同様な値

・Priority：上記 TSN スイッチと同様な値

・Time Window：送信周期内で設定する論理リング型タイムスロットの数．本実装では，４つのノードで，上記の TSN スイッチ用の「分散データ共有処理用送信」の Time Window の「Duration」を４つに分けた値

具体的には設定ファイルの内容を以下に示す．

表 1 設定ファイル例

パラメータ名	値
TX CYCLE	1000us
PRIORITY	5
Time Window	4
Time Window =1	OFFSET: 100/Duration: 100
Time Window =2	OFFSET: 100/Duration: 100
Time Window =3	OFFSET: 100/Duration: 100
Time Window =4	OFFSET: 100/Duration: 100

(2) 送信処理の実装
図 7 に従い，送信処理の詳細を述べる．

(LT1) TSN の送信スケジューラの TAS のパラメータ設定ファイルからの TAS パラメータを読み取り，パケット送信時間を設定する．
(LT2) 送信周期内で送信可能な論理リング型タイムスロットの数を読み取る
(LT3) 論理リング型タイムスロットの数分ループする
(LT4) 送信するデータを図 3 の分散共有メモリから読取り，図 3 の「Pay Load」へ設定する．
(LT5) UDP 送信を行う．スターTopology のため，マルチキャスト送信が可能である．一回の送信パケットのサイズは通常ネットワークの default MTU である 1500 バイト以下である．
(LT6) (T1)で読み取ったステップ(LT3)で読み取った論理リング型タイムスロットのパラメータで，次の送信時間を算出する
(LT7) 次の送信周期を待つ．

ただし，各ノードの送信デモンプロセスはどの論理リング型タイムスロットを使うかはコンテナのアプリへ提供する「図 3 の(L1) 分散共有メモリのライブラリ」を通して設定する．
それによって，一つの物理ノード上で，複数のコンテナを運用することも可能である．

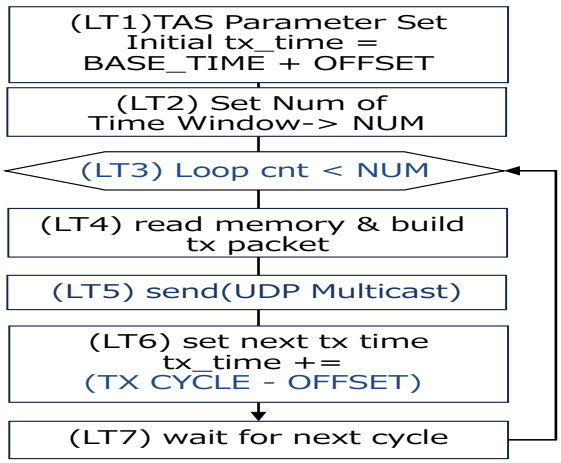

図 7　論理リング送信処理概要

5. 評価

本研究で実装した分散データ共有処理を図 8 のテストベッドへ適用し，その有効性を確認した．

5.1 評価環境

テストベッドは４つのノードと２つの TSN スイッチで構成されている．図 8 のノード１とノード２はノード３とノード４からのデータを処理するノードで，ノード３とノード４は IO 処理（カメラやロボット制御信号）とその処理結果である IO データを，前記したノード１とノード２と共有する．ノード３とノード４は，AI を活用する AI ノードとするため，FPGA 内蔵の計算機ノードとする．各ノードの仕様を表にまとめる．

表 2 テストベッド仕様

#	名称	項目	仕様	備考
1	ノード１ ノード２	CPU	Intel i7-4790 3.6Ghz	Dell PC × 2
2		MEM	8G	―
3		OS	Linux 5.3	Ubuntu 18.04
4		NW	1Gbps	i210T(TSN NIC)
5	ノード３ ノード４	CPU	Intel Aria10	AIxFPGA
6		MEM	8G	―
7		OS	Linux 5.3	Ubuntu 18.04
8		NW	1Gbps	i210T(TSN NIC)
9	TSN スイッチ	-	Kontron 社 ２台	4 ポート／台

5.2 評価方法

(1) 分散データ共有処理の有効性評価
テストベッドで実現する AI を取り組んだ制御システムにおいて，本研究で提案した分散データ共有処理の有効性を評価する．
下記にてテストベッドで実現したシステムの処理全体を説明する．
・全体に４つのノードと TSN スイッチで構成されている．
・システムは，組立作業を行うロボットを制御するノードと，その作業場に人間の侵入検知を行うノードがあり，作業場に人間侵入を検知すると，ロボットを即座に停止する処理を実現するものである．
・ノード４（図 8 の Node#4）のコンテナ上のアプリは，カメラからの画像から人間の侵入を検知する AI ロジックを OpenVINO を活用して実現したものである．本アプリは，人間の侵入を検知すると，前記した分散データ共有処理の分散共有メモリにその情報を書き込み,全ノードと共有する．
・ノード１（図 8 の Node#1）のコンテナ上のアプリは，ノード４のアプリが書き込んだ人間検知の情報を用い，作業

停止要否を判断する．その判断情報を前記した分散データ共有処理の分散共有メモリに書き込み，全ノードと共有する．

・ノード２（図８の Node#2）のコンテナ上のアプリは，ノード１のアプリが書き込んだ作業停止要否の情報を用い，作業停止コマンドを発行する．そのコマンドを前記した分散データ共有処理の分散共有メモリに書き込み，全ノードと共有する．

・ノード３（図８の Node#3）のコンテナ上のアプリは，ノード２のアプリが書き込んだ作業停止コマンドを用い，組立作業のロボットを停止する．

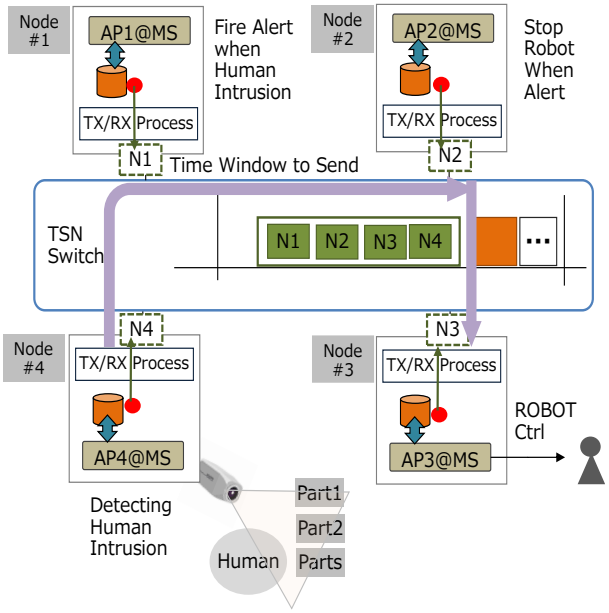

図８　テストベッド構成

本共通テストベッド上で，全ノードから書き込まれる情報を制御周期 1ms 内で共有され，人間の侵入検知からロボット停止までの処理全体のリアルタイム性が保証されることが確認できた．また，OpenVINO を活用して開発したアプリ含め，全ノードのアプリのコンテナ化及び，分散データ共有処理の実現が可能であることを確認した．

以後，分散データ共有処理のリアルタイム性について詳細を述べる．

(2) 分散データ共有処理のリアルタイム性評価

図８のノード１とノード２間において，分散データ共有処理上の通信のリアルタイム性を評価するために，TSN を使わないコンテナのアプリでの通常の UDP 通信との性能比較を行う（図 9）．送信データは default MTU で送信可能な送信サイズの 640 バイトとし，高リアルタイム性が必要なコントローラの制御周期である「1ms」で送信する．そこで，送信から受信までの時間を評価する．評価ポイントは，

（１）送信デモンプロセスの送信から tc の送信キューまでの時間（図 9 の T①）

（２）TSN の NIC デバイス間送受信時間（図 9 の R①－１）

（３）カーネルスタックから受信処理までの時間（図 9 の R①－２）

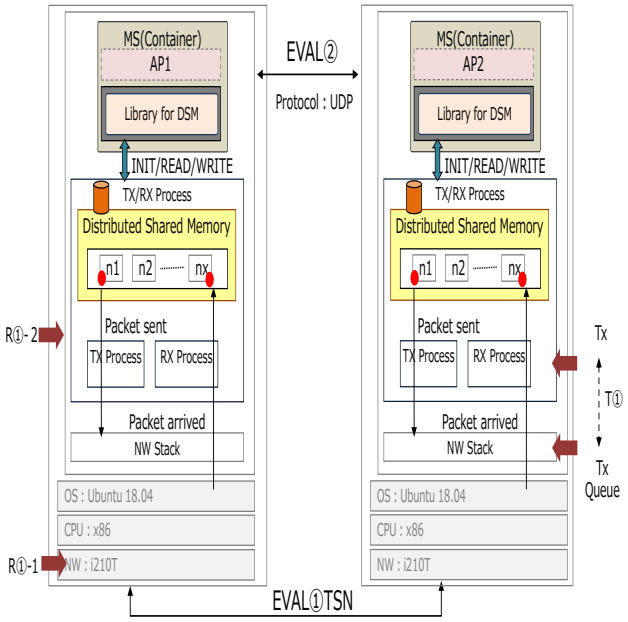

図９　リアルタイム性評価

5.3 評価結果と考察

下表にて分散データ共有処理のリアルタイム性評価の結果を示す．

表 3 Evaluation Results

	EVAL①TSN（図 9）			EVAL②
	T①	R①-1	R①-2	
AVG	83us	13ns	32us	104us
MAX	93us	25ns	119us	222us
STDEV	1.5us	12ns	12us	33us

TSN 対応 NIC デバイス間の通信処理は最大 25 ナノ秒であるため，データ送信から受信までの処理時間は，大半はカーネルスタックから受信処理までの時間（表 3 の R①-2）に左右されることがわかる．また，TSN を使わないコンテナ上の UDP 通信（表 3 の EVAL②）に対して性能の揺らぎ及び最大処理時間が抑えられていることがわかる．本研究での実装では，全ノード間は UDP のマルチキャストでデータ送信をするため，１つのノードから送信したデータを全ノード間のデータ共有あるためにかかる時間は，「表 3 の R①-2」に依存する．上記の結果から約 120μ秒で全ノードでデータが受信可能であることがわかる．そのため，コントローラの制御周期である「1ms」以内のデータ共有可能であり，制御周期内でのデータ受信が可能と言える．一方，従

来の制御ネットワークでは，最大のコントローラの数は 16 台で，Ring 構成であるため，マルチキャストではなく，あるノードから次のノードへ順次に送受信を行う.そのため，全ノードでデータが受信できるのは 8 台分の受信処理時間が必要である.上記の結果から約 960μ秒(119μ秒×8 台)で全ノードでデータ受信が可能であることが判る．それはコントローラの制御周期である「1ms」以内の結果であるが，本結果は前記の評価環境での実測値であるため，実際のシステムへの適用のためには性能改善は必要である．受信処理の最大処理時間の原因としては，カーネルネットワークスタックの通信データ処理オーバーヘッドと，その処理中の資源の排他処理によりスケジュールオーバーヘッドが考えられる．また，２つのノード間での通信オーバーヘッドも存在する．これらのオーバーヘッドを削減するために，受信処理をカーネル処理にオフロード可能な XDP を採用する方式が考えられる[6].

6. おわりに

本論文では，複数のコンテナを運用する制御コントローラ上のコンテナ間のデータ共有を実現する分散共有メモリ方式を提案し，その設計や実装方法について説明した．また，スター型のネットワーク topology において TSN 技術が活用適用可能な論理リング送信方式を提案し，コンテナ間でのデータ共有におけるリアルタイム性保証の有効性を示すとともに，課題を述べた.

今後，今回の評価においての課題を解決するために，分散共有メモリ方式の受信処理を，カーネル処理にオフロード可能な XDP を採用する方法ついて研究していく.

謝辞 本研究にご協力頂いた皆様に，謹んで感謝の意を表する.

参考文献

[1] Yanghu Guo, et al,“A container scheduling strategy based on neighborhood divison in micro service”, NOMS 2018, 2018
[2] 近藤翼. 組み込み制御システム向け分散共有メモリ機構を持つリアルタイム OS, 第 80 回全国大会講演論文集,2018, no. 1, p. 109-110.
[3] Dietmar, et al, "An Introduction to OPC UA TSN for Industrial Communications Systems", IEEE 2019, 1-11
[4] Siwar Ben, et al, "SDN-based configuration solution for IEEE802.1 TSN", ECRTS, 2018
[5] Jesus, et al, "The Road Towards a Linux TSN Infrastructure", ELC, 2018
[6] Dominik, et al, "Performance Implications of Packet Filtering with Linux eBPF", ITC30, 2018, 209-217
[7] Cristian, et al, “An Edge Computing Architecture in the Internet of Things”, IEEE ISORC, 2018, 99-102
[8] IEEE Std 802.1Qca-2015 - IEEE Standard for Local and Metropolitan Area Networks --Bridges and Bridged Networks -- Amendment 24 (IEEE, 2015)
[9] IEEE Std 802.1Qbv-2015 - IEEE Standard for Local and Metropolitan Area Networks --Bridges and Bridged Networks -- Amendment 25 (IEEE, 2015)
[10] IEEE Std 802.1Qbu-2016 - IEEE Standard for Local and Metropolitan Area Networks --Bridges and Bridged Networks -- Amendment 26 (IEEE, 2016)
[11] A.Mifdaoui, et al, "Performance Enhancement of Extended AFDX via Bandwidth Reservation for TSN/BLS Shaper", ECRTS, 2018
[12] Yongki Park, et al, “Performance Analysis of CNI”, 2018 ICTC, 2018