情報処理学会論文誌 Vol.56 No.3 879–891 (Mar. 2015)

高精度時刻同期を分散処理制御に活用したタイムアウェア処理方式

堤 智昭[1,a)] 大島 浩太[2] 小泉 稔[3] 中條 拓伯[4]

受付日 2014年6月30日, 採録日 2014年12月3日

概要：ネットワーク内のノード間における高精度な時刻同期を可能とするIEEE1588 PTPv2の登場を背景に，ナノ秒やマイクロ秒レベルの精度で同期された時刻と，あらかじめ定めた動作シナリオによる分散処理制御方式が注目されている．その動作シナリオに基づき要求されたタスクを実行することにより，進行中の処理状況を把握するための制御パケットが不要となり，分散ノード間で協調動作が可能となる．しかしながら，実際に変動するネットワーク環境では処理の進行中に遅延が発生し，その遅延によって動作シナリオとのずれが生じる．そこで本研究では，ノードの性能を含む特性により生じる処理遅延や，ノード間の通信時に発生する通信遅延を考慮し，ノード間で共有する動作シナリオと実際の実行状況との差異を計測する．その結果をもとに，実行状況を反映した動作シナリオになるよう動的にシナリオを調整することで，複数ノードに分散したタスクの実行を，あらかじめ設定された時間以内に完了させることを可能とする分散制御方式を提案する．提案方式の有効性を検証するための実験環境を構築し，評価を行った結果，提案方式が有効に機能することを確認した．

キーワード：分散処理，ネットワーク，時刻同期，時刻情報応用

A Time-aware Control Method Using High Precision Time Synchronization

Tomoaki Tsutsumi[1,a)] Kohta Ohshima[2] Minoru Koizumi[3] Hironori Nakajo[4]

Received: June 30, 2014, Accepted: December 3, 2014

Abstract: IEEE1588PTPv2 has been proposed to keep highly accurate time synchronization among nodes in a network. With the protocol, tasks can be executed in accurate timing according to operating scenarios without sending/receiving control packets which check status of other nodes. In a real network environment, time delay due to different performance among nodes and as well as to network congestion, which brings time deviation between an operating scenario and real execution. In this study, we propose a method to check the time deviation and accommodate an operating scenario to an optimal operation. This control mechanism is called as a Time-aware Distributed Processing Control Method. We have developed a prototype system to verify the effectiveness of the proposed method. From the experiment results, our proposed method shows effectiveness in some distributed processing applications.

Keywords: distributed processing, network, time synchronization,time information applied system

1. はじめに

近年，IEEE1588 PTPv2 [1] の登場により，IP ネットワーク上で複数ノードの時刻をマイクロ秒やナノ秒のレベルで

[1] 東京農工大学大学院工学府
The Graduate School of Engineering, Tokyo University of Ariculture and Technology, Koganei, Tokyo 184–8588, Japan
[2] 埼玉工業大学工学部情報システム学科
Department of Information Systems, Faculty of Engineering, Saitama Institute of Technology, Fukaya, Saitama 369–0293, Japan
[3] 株式会社日立製作所
Hitachi, Ltd., Yokohama, Kanagawa 244–0817, Japan
[4] 東京農工大学大学院工学研究院
Institute of Engineering, Tokyo University of Agriculture and Technology, Koganei, Tokyo 184–8588, Japan
[a)] t_tsu77@nj.cs.tuat.ac.jp

同期可能となり，高精度に同期された時刻と，あらかじめ定めた動作シナリオを用いた分散処理制御が実現可能となってきた．このIEEE1588 PTPv2を基にした，マルチメディア機器をマイクロ秒レベルで時刻同期するための規格としてIEEE802.1AS [2] の標準化が検討されており，今後さらに高精度な時刻同期機構を持つ機器が普及するものと考えられる．

このような高精度時刻同期を用いた分散処理制御システムの適用の1つに産業オートメーションがある．工作機器が個別に単独で動作するのではなく，機器どうしがネットワークで接続され，さまざまな情報，状況を通信し合うといった機能を備えた新たな生産ラインの実現が可能である．たとえば，生産ラインの監視・制御システムで，金属製の素材をベルトコンベア上で順次加工していく場合に，加工前にセンサで加工対象の材質や形状を計測・解析し，その結果に応じて工作機械群の処理内容や処理スケジュールを自動的に調整するといった応用が考えられる．また，近年注目が集まっているCyber Physical System（CPS）分野における応用も考えられる．山中らが提案しているuGrid環境 [3], [4], [5] やAESOPプロジェクト [6], [7] のように，多数のデバイスを組み合わせたサービスフローによってユーザの要求する機能を提供するシステムにおける，デバイス制御に適用することも可能である．

従来は，モデル化が可能かつ，求められる制御が計算可能な制御系については自動化が行われていたが，複数の事象が干渉したり，予測に膨大な計算が必要であったりして，制御の自動化が難しい制御には熟練のオペレータによる調整が必要であった．これに対し，オペレータ調整に依存していた処理を，制御系とネットワーク接続された複数計算機ノード群の分散処理で高速に実現することにより，より高度な制御計算を可能とすることで，運転制御の自動化を促進することができるが，一定時間内に制御応答を収めなければならないといった課題が存在する [8]．提案方式により，こういった課題が解決でき，より省人化され自動化された生産ライン制御が可能になる．

複数ノードが連携して1つのタスクを実行する分散処理システムにおいて，高精度に同期された時刻と動作シナリオを用いてマイクロ秒単位でタスクを構成する各処理の実行を制御する場合，ノード内で生じる処理遅延やノード間の通信遅延の影響により，タスクの実行状況と決められた目標時刻との間に時間的なずれが生じる．その結果，必要な入力が得られず処理が進行できなかったり，誤った結果が出力されたりするといった問題が発生する．そのため，この時間のずれを想定したうえで，その影響が最小限となるよう，目標時刻を動的に修正するような機構が有効となると考えられる．

そこで本研究では，複数のノードが連携してタスクを処理する分散処理において，各ノードの時計をIEEE1588 PTPv2によりナノ秒レベルの高精度に同期し，その時刻に基づいてタスクを実行する分散処理制御方式を提案する．提案方式は，分散処理制御の実行手順をあらかじめ動作シナリオという形で定義したものを分散処理ノード間で共有しておき，各ノードはシナリオに記述されたスケジュールに沿ってマイクロ秒レベルの精度で処理を実行する．これにより，各ノードがお互いの処理状況を確認する通信が大幅に低減できるとともに，任意の時刻までにタスクの実行を完了することが可能な分散処理制御の実現が期待できる．このような時刻ベースの分散制御方式をタイムアウェア型分散処理制御方式と名付けた．本論文では，遅延時間の変動をともなう環境下で，タスクを一定時間以内に完了させるための動作シナリオの動的な調整方式を提案する．提案方式により，ノードが予定時刻より前に処理が完了した場合は動作シナリオの前倒しを，予定時刻より処理の完了が遅かった場合は動作シナリオの後ろ倒しを実施する．この2種類の制御により，タスクの実行時間を一定時間以内に収めることが可能となる．

以降，2章において関連研究について触れ，3章においてタイムアウェア型分散処理制御方式の概要について述べる．続く4章では，その実現方法について述べ，5章で実験評価環境を示す．6章では評価結果について述べ，7章でまとめる．

2. 関連研究

高精度に同期された時刻情報の応用は，センサネットワーク [9], [10] や電力網の位相同期の基準情報 [11] において利用されている．また，モーションコントロール [12]，産業オートメーション機器の制御 [13] などの分野への適用に関する提案もある．これらは大きく次の2つに分類できる．1つは，複数ノード間で過去に行った処理に対して処理順序の整合性を保つための情報として用いる応用である．もう1つは時刻情報に従ってこれから行うノードの動作を制御する応用である．

前者の例としては，センサネットワークにおいて複数のセンサを用いて計測を行うとき，それぞれのセンサが計測した結果の時間的整合性を保証するために時刻情報が利用されている例があげられる．この場合，ノード間の時刻同期精度とタイムスタンプ精度が重要となる．文献 [14] では，適用例として1ミリ秒の時刻同期精度が求められる火山活動モニタリングと，100マイクロ秒の時刻同期精度が求められる地震モニタリングをあげている．これらの研究は，分散ノード間における事象の記録を主な目的としており，制御を目的とした本論文とは時刻の利用目的が異なる．

後者の応用例としては，工場のベルトコンベアの切替弁制御への適用を想定し検討したものがあげられる [13]．高精度に同期された時刻に応じてノードを制御するという点では，本論文と同様の時刻利用方法であるといえる．しか

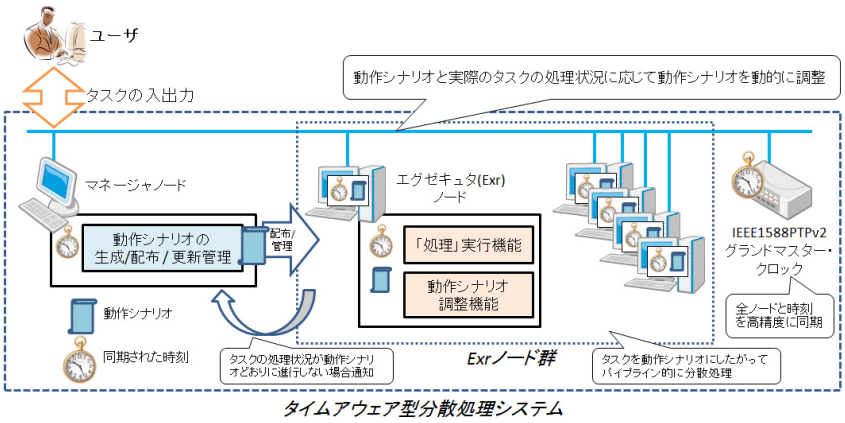


図 1 タイムアウェア型分散処理制御の概要

Fig. 1 Systeme Model of time-aware distributed processing.

し，一定時間以内に処理が終了するシステムを制御対象としており，処理遅延が発生した場合の対応については課題となっている．また，我々の先行研究である時刻同期型分散モバイルネットワークエミュレータ [15], [16], [17] も，分散ノード間の協調動作に時刻情報を利用しており，応用例の 1 つとしてあげられる．

時刻同期型分散モバイルネットワークエミュレータ（以下，モバイルネットワークエミュレータ）では，IEEE1588 PTPv2 を用いて分散ノードの時刻を同期している点は本論文の提案システムと同様である．モバイルネットワークエミュレータはネットワークを利用する移動端末用システムの開発段階での機能・性能検証を目的として，実際のフィールド（線路や道路上など）に実機を展開した機能検証が困難である点に着目し，その実機の無線通信部分や移動状況を仮想的に模擬試験できる機能を備えたネットワークエミュレータであるのに対し，本論文は分散処理制御を目的としている点で対象としているシステムが異なる．モバイルネットワークエミュレータでは，任意の 2 ノードがどの時刻に通信可能かを定義した動作シナリオに基づいて，その 2 ノード間の通信をエミュレータシステムが仲介することで移動および無線通信の模擬を行う．動作シナリオは，通信発生時にノードがどのような処理を実行するか判断するために利用され，作成後に変更されることはない．これに対して本研究では，処理時間や通信に遅延が生じかつそれが変動する環境で実施する分散処理を対象としている．このような対象において，時刻情報と動作シナリオに基づいて分散処理を制御するために，本研究では動作シナリオにおいて，ノードが処理を実行するスケジュールを定義する．一定時間以内にタスクを完了するために，処理状況が動作シナリオと異なった場合に動的に動作シナリオを調整する機能を提案している．

3. タイムアウェア型分散処理制御方式

同期時刻と動的に調整される動作シナリオに従って実行するタイムアウェア型分散処理制御方式のシステム概要と，パイプライン的にタスクを実行する例を**図 1** に示す．

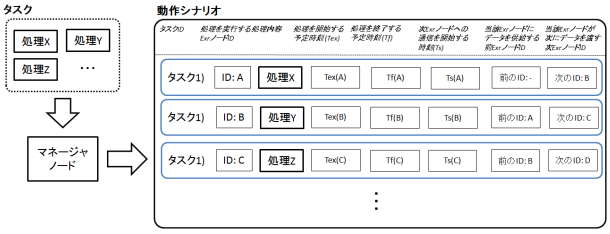


図 2 タスクと動作シナリオ

Fig. 2 Example of task and processing scenario.

本システムは，タスクを実行するエグゼキュタノード（以下，Exr ノード）群と動作シナリオの管理を行うマネージャノード，およびすべてのノードと時刻を同期する IEEE1588 PTPv2 グランドマスタ・クロックから構成される．システムが実行するタスクは，Exr ノード間で共有する動作シナリオにおいて処理として記述される．Exr ノード群は動作シナリオに基づいてタスクを分散処理し，結果はユーザのもとへ出力される．

Exr ノードが遅延の影響で動作シナリオどおりに処理を実行できない場合，その時点の処理状況に応じて動的に動作シナリオを調整する．その調整は Exr ノードが行い，調整した動作シナリオはマネージャノードを通して，他の全 Exr ノードに配布される．また，マネージャノードは新たな動作シナリオを適用できるよう，その切替えタイミングを管理する．

タスクと動作シナリオの概要を**図 2** に示す．タスクは複数の処理に分割され，それをマネージャノードがその処理を実行できる Exr ノードの選択や処理のタイムスケジューリングを行う．この動作シナリオは，各 Exr ノードがパイプライン的にタスクを実行するためのスケジュールとして使用する．図 2 中のそれぞれのパラメータを以下に示す．

(1) タスク ID

(2) 処理を実行する Exr ノード ID

(3) 処理内容
(4) 処理を開始する予定時刻（*Tex*）
(5) 処理を終了する予定時刻（*Tf*）
(6) 次 Exr ノードへの通信を開始する時刻
(7) 当該 Exr ノードにデータを供給する前 Exr ノード ID
(8) 当該 Exr ノードが次にデータを渡す次 Exr ノード ID

動作シナリオに従って，Exr ノード A は時刻 $Tex(A)$ に割り当てられた処理を開始し，時刻 $Tf(A)$ までに終了する．その後時刻 $Ts(A)$ に Exr ノード B へデータ転送を開始する．データを受信した Exr ノード B は，Exr ノード A と同様に $Tex(B)$ に割り当てられた処理を開始し，タスクを進行する．

本システムでは，他の Exr ノードがどういった処理を行うかは動作シナリオに記述されているため，動作シナリオどおりにタスクが進行している限り，タスク処理中の Exr ノード間で処理状況を確認するための制御パケットを送る必要はない．しかしながら，実際のネットワーク環境においては動作シナリオどおりに進行するとは限らず，そのため，動作シナリオの動的な調整を行う必要がある．

4. タイムアウェア型分散処理制御方式の実現

4.1 動作シナリオの動的な調整

実際の動作状況を考慮して動作シナリオの調整を行うためには，まず動作シナリオに記述された時刻とノードで処理が実行された時刻とのずれを計測する必要がある．計測値から，動作シナリオと照らし合わせて生じているタスクの処理遅延を推定し，動的に動作シナリオを調整する．遅延はその発生要因とその程度によって動作シナリオに与える影響が異なる．そのため，発生要因ごとに生じている遅延を分類し，その分類に基づいて動作シナリオの調整の程度を推定する方式を検討する．

まず，以下の 3 つのタイミングで取得したタイムスタンプについて，動作シナリオに記述されている時刻と比較し，発生している時間のずれを計測する．

(1) Exr ノードが処理の実行を開始した時刻
(2) 処理を終了した時刻
(3) 次の Exr ノードにデータ送信を開始した時刻

4.1.1 遅延の要因

ここでは，発生する遅延の原因を大きく (A) Exr ノードの性能や特性に起因する遅延，(B) ネットワーク的要因に起因する遅延の 2 つに分ける．

(A) は，時刻取得処理の実行にかかる時間といった Exr ノードの動作速度や，OS の割込み処理のタイミングといった各 Exr ノードのタイマ分解能が影響を及ぼすと考えられる．具体的には以下が考えられる．

(1) タイムアウェア分散処理制御のための時刻取得処理やタイムスタンピング
(2) NIC などのデバイスとのやりとりや OS による割込み処理による遅延
(3) システム負荷による処理遅延

(B) は，Exr ノード間の通信性能やホップ数の違い，通信経路中のネットワーク機器上の輻輳といった，ネットワークの特性が原因で生じる．具体的には，次の 2 つが考えられる．

(4) ポートからパケットが送出されるまでの伝送遅延や Exr ノード間の伝搬遅延による通信遅延
(5) ルータによる再送やキューイングといった転送処理にかかる遅延

4.1.2 Exr ノードの性能や特性に起因する遅延の計測方法

要因 (1)–(3) については，Exr ノードにおけるタイムスタンピングに要する時間を計測する．また，動作シナリオに指定された時刻まで待機した後，タスク実行に復帰するまでの時間を計測する．タイムスタンピングに要する時間はシナリオの実行前に計測する．待機状態からの復帰に要する時間は，待機直前と復帰直後の時刻を取得し，その差分から求める．

4.1.3 ネットワーク的要因に起因する遅延の計測方法

要因 (4)，(5) に起因する遅延に対しては，各ノードにおいてノード間の通信に要した時間を計測する．送信ノードは，パケットに送信した時刻をタイムスタンプする．受信ノードはパケットを受信した時刻とパケットにタイムスタンプされた時刻との差分を計算する．計測のタイミングは動作シナリオの実行前およびパケット通信発生時に行う．

4.2 動作シナリオ調整のための要件

提案方式の開発にあたり，動作シナリオを全ノードで整合性を保ったまま書き換え，配布することが課題となる．動作シナリオには複数の Exr ノードで実行されるタスクが記述されているため，新たな動作シナリオに変更するタイミングを全ノードで同期しなければならない．動作シナリオの調整モデルを構築するためは，以下の 3 点を決定する必要がある．

(1) 動作シナリオの調整項目
(2) 動作シナリオの調整タイミング
(3) 調整した動作シナリオの共有方式

4.2.1 動作シナリオの調整項目

動作シナリオの調整では，Exr ノードが処理の実行を開始する時刻 *Tex* と，次のノードへパケットを送信する時刻 *Ts* を調整する．今回提案する調整の種類は，**図 3** に示す処理を後ろ倒しにする調整と，前倒しにする調整の 2 種類とした．*Tex*，*Ts* それぞれの時刻について，実測した遅延時間データをもとに，動作シナリオの調整を行う．図中の例では，Exr ノード A から Exr ノード B へ転送するときの遅延が大きく，Exr ノード B での処理開始時刻までにデータが届かないと予想される場合に，Exr ノード B におけるタスク 1 の *Tex* を後ろ倒しにしている．また Exr ノード A

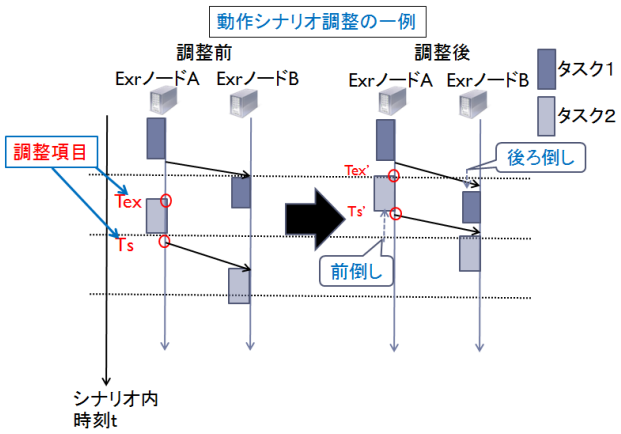


図 3 動作シナリオの調整項目
Fig. 3 Coordinate item of scenario.

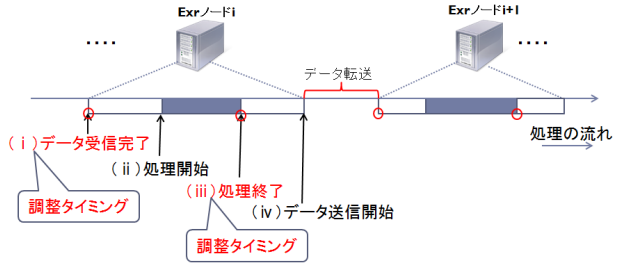


図 4 動作シナリオの調整タイミング
Fig. 4 Coordinate timing of scenario.

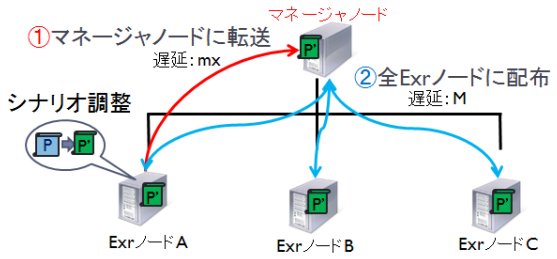


図 5 マネージャノードを利用した動作シナリオの配布
Fig. 5 Distribution method using the manager node.

では，タスク 1 の処理終了後から次のタスク 2 の処理開始まで，リソースに余裕があり，タスク 2 の Tex を前倒しにしている．

4.2.2 動作シナリオの調整タイミング

Exr ノードでの処理実行は，図 4 に示すように以下の順で行われる

(i) データ受信完了
(ii) 処理開始
(iii) 処理終了
(iv) データ送信開始

調整はそのうち (i) のデータ受信完了時と (iii) の処理終了時の 2 つのタイミングで行う．

4.2.3 動作シナリオの共有方式

調整された新しい動作シナリオを他 Exr ノードと共有するために，図 5 に示したマネージャノードを通じて共有する方式を用いる．Exr ノードが動作シナリオ P を P' に調整した後，マネージャノードに P' を送信し，マネージャノードから他のノードへ P' を配布する．マネージャノードはシステム内のすべての Exr ノードに動作シナリオを配布するのにかかる遅延を計測し，新しい動作シナリオを適用するタイミングを管理する．

動作シナリオの調整を行った Exr ノード A とマネージャノードとの通信にかかる時間を mx，マネージャノードからすべての Exr ノードに新しい動作シナリオを配布するのにかかる時間を M とすると，P' は調整が行われた時刻から $M + mx$ 後に適用される．ここでの M はマネージャノードから動作シナリオを各 Exr ノードへ配布する時間と各 Exr ノードからの受信応答を受け取る時間を含む値とする．

Ts，Tex を調整した新しい動作シナリオを適用した時点で，新しい動作シナリオに沿った動作ができない場合，動作シナリオの調整は行えない．記載された時刻に調整できる場合の条件は，Exr ノードが動作シナリオ調整処理を実行する時刻を Tch，調整後の新しい Ts，Tex の時刻を TX' とすると式 (1) で表すことができる．

$$Tch + M + mx <= TX' \tag{1}$$

4.2.4 動作シナリオの調整モデル

実際に Exr ノード i が前の Exr ノード $i-1$ からデータを受信した時刻を $Tr(i)$，動作シナリオに記述されている Exr ノード i が処理を開始する時刻を $Tex(i)$，処理が終了する時刻を $Tf(i)$，Exr ノード i から次の Exr ノード $i+1$ へのパケット送信を開始する時刻を $Ts(i)$，次の Exr ノード $i+1$ における処理の開始時刻を $Tex(i+1)$，処理の終了時刻を $Tf(i+1)$，パケットの送信開始時刻を $Ts(i+1)$ とする．また，計測したネットワーク的要因に起因する遅延を dn，時刻取得にかかる遅延を dl，Exr ノードが待機状態から復帰するまでにかかる時間を dt とする．

他の Exr ノードが同一のタスクに対する調整を同時に行わないようにするため，シナリオの調整は，Exr ノードが実行中のタスクについてのみ行う．調整は，それぞれの調整タイミングに近い時刻の動作から順番に調整を行う．すなわち，ノード i がタスク n を処理するとき，データ受信完了のタイミングではタスク n の $Tex(i)$ を調整後に $Ts(i)$ を調整する．処理終了のタイミングではタスク n の $Ts(i)$，$Tex(i+1)$ の順で調整する．Tex，Ts の調整はそれぞれ次のように行う．

Ts の後ろ倒し調整

図 6 に示すように，Exr ノードでの実行時に処理が遅延したり $Tex(i)$ の調整により $Tf(i)$ が遅くなり式 (2) を満たす場合，式 (1) を満たす時刻になるまで $Ts(i)$ を後ろ倒しにする．

$$Tf(i) > Ts(i) \tag{2}$$

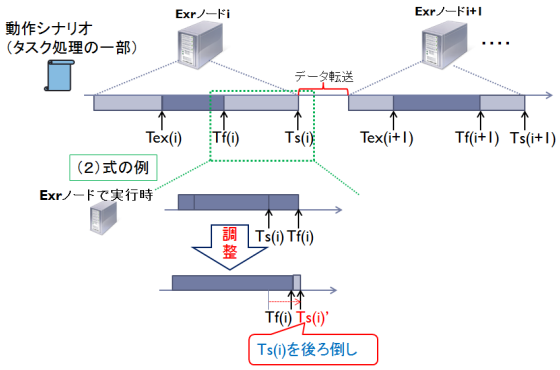


図 6 Ts 後ろ倒し例
Fig. 6 Model of postpone Ts.

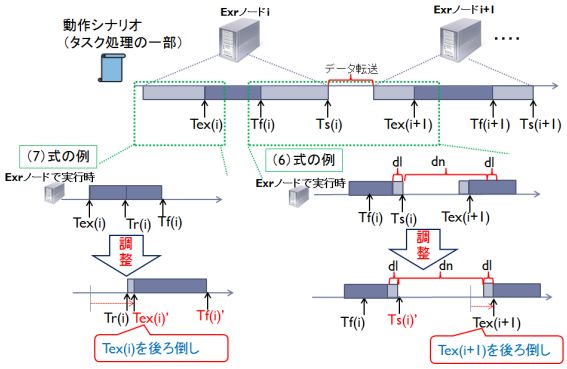


図 8 Tex 後ろ倒し例
Fig. 8 Model of postpone Tex.

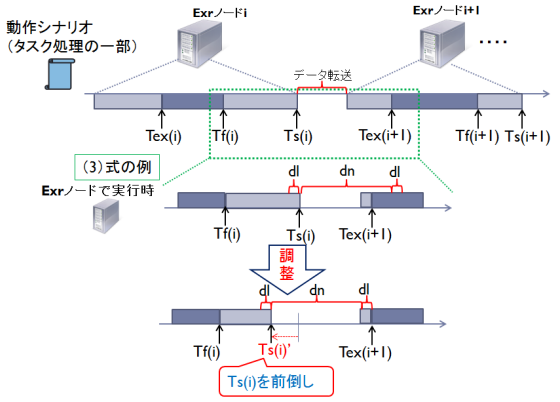


図 7 Ts 前倒し例
Fig. 7 Model of accelerate Ts.

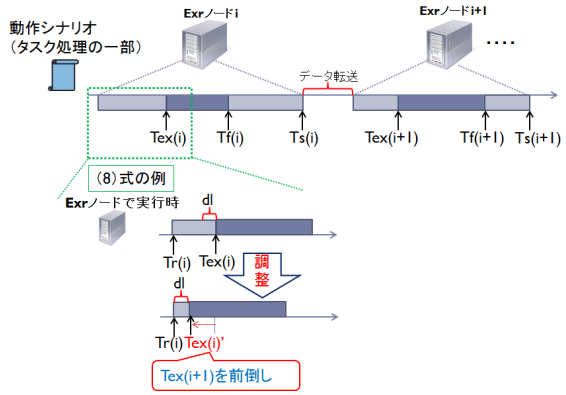


図 9 Tex 前倒し例
Fig. 9 Model of accelerate Tex.

Ts の前倒し調整

Exr ノードが処理を開始するタイミングで，**図 7** に示すような，遅延の影響により Exr ノード $i+1$ の動作時刻にパケットが届かないと予測される場合，すなわち式 (3) を満たす場合，$Ts(i)$ が前倒し可能であれば $Ts(i)$ の前倒しを行う．

$$Ts(i) + dn + dl > Tex(i+1) \tag{3}$$

また，Exr ノードが処理を終了するタイミングで，実際に処理が終了した時刻が動作シナリオに記述された $Tf(i)$ よりも早かった場合，$Ts(i)$ の前倒しを行う．それにともない，それ以降の $Tex(i+1)$ も前倒しされる．調整後の時刻 $Ts(i)'$ は，処理終了からデータ転送開始までにかかる遅延 dl と動作シナリオの調整にかかる遅延を考慮し式 (4)，および式 (1) を満たす時刻まで前倒し可能である．また，処理終了からデータ転送開始まで待機時間が生じる場合，待機可能な最短時間 dt よりも待機時間が短くならないように式 (5) を満たす必要がある．

$$Tf(i) + dl < Ts(i)' \tag{4}$$

$$Ts(i)' - Tf(i) > dt \tag{5}$$

Tex の後ろ倒し調整

図 8 に示した式 (6) の例のように，$Ts(i)$ を $Ts(i)'$ に調整した後も式 (6) を満たす場合 $Tex(i+1)$ を後ろ倒しする．

$$Ts(i)' + dn + dl > Tex(i+1) \tag{6}$$

また，図 8 に示した式 (7) の例のように，実際に Exr ノード i がデータを受信した時刻が処理を開始する時刻よりも遅かった場合，すなわち式 (7) を満たす場合，$Tex(i)$ を後ろ倒しする．それにともない，$Tf(i)$ も後ろ倒しされ $Tf(i)'$ となる．

$$Tex(i) < Tr(i) \tag{7}$$

Tex の前倒し調整

図 9 に示す例のように，Exr ノード i がデータを受信してから処理の開始までに余裕がある場合，すなわち式 (8) を満たす場合，調整後の $Tex(i)'$ が式 (1) を満たす時刻まで $Tex(i)$ を前倒しにする．

$$Tr(i) < Tex(i) \tag{8}$$

5. プロトタイプ実装

提案方式の有効性を確認するため，プロトタイプシステムを開発し評価を行った．プロトタイプシステムにおいて，動作シナリオの調整は，タスク処理中の Exr ノードと次にタスクを処理する Exr ノードに対してのみ行うものと

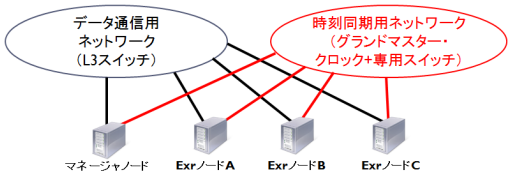


図 10 プロトタイプシステムのシステム構成
Fig. 10 Configuration of prototype system.

表 1 ノード仕様
Table 1 Specifications of nodes.

項目	仕様
CPU	Intel Core2Duo *2 E6750（2 コア 2.66 GHz）
Memory	2 GB（DDR2 PC2-6400）
Storage	HDD（SATA2 300, 7200 rpm, 160 GB）
OS	Linux *3（Ubuntu *4 10.10（Kernel2.6.31 CONFIG PREEMPT RT パッチを適用））
NIC	Marvell 製 1 Gbps Meinberg 製 PTP270PEX（時刻同期ネットワーク接続）

する．

プロトタイプシステムのシステム構成を図 **10** に示す．ここでは，3 台の Exr ノードと 1 台のマネージャノードを用いた．PTPv2 による時刻同期精度を保証するため，専用のスイッチを用いた時刻同期用ネットワークと，ノード間のデータ通信用ネットワークの 2 つを用いる構成とした．数十マイクロ秒レベルでの分散ノードの動作制御を目標とし，ノードには表 **1** に示す汎用 PC を用いた．時刻同期ネットワークに接続するインタフェースは，PCI Express *1 接続の PTPv2 OC（Ordinary Clock）機能搭載 NIC（Meinberg 社製 PTP270PEX．以下，PTPNIC）を使用した．これにより，ノード間でナノ秒レベルの時刻同期精度を確保し，マイクロ秒レベルでの動作シナリオ実行には時刻同期のずれは無視できるものとした．

現在時刻の情報は，この PTPNIC から専用の API を用いて取得する．また，動作シナリオで指定された時刻まで待機をするために usleep システムコールを用いた．

ノードには，Kernel バージョン 2.6.31 に CONFIG PREEMPT RT パッチを適用した Ubuntu10.10 を用いた．プロトタイプシステムはソフトウェアとして実装し，開発には C 言語を用いた．

6. 評価

同期時刻と動作シナリオを用いた分散処理制御に影響を与える遅延について計測し，提案するモデルに従って動作シナリオの調整を行い，調整モデルの有効性を検証した．

*1 PCI Express は PCI-SIG 社の登録商標または商標です．
*2 Intel Core 2 Duo は，Intel Corp. の登録商標または商標です．
*3 Linux は，Linus Torvalds 氏の日本およびその他の国における登録商標または商標です．
*4 Ubuntu は，Canonical Ltd. の登録商標です．

表 2 時刻取得実行時間測定結果
Table 2 Evaluation result of obtaining the time information.

CPU 負荷	平均実行時間（マイクロ秒）	標準偏差（マイクロ秒）
なし	4.0	0.1
1 コア	4.76	32.2
2 コア	12.6	225.9

表 3 待機状態からの復帰時間測定結果
Table 3 Evaluation result of return from the waiting state.

CPU 負荷	平均実行時間（マイクロ秒）	標準偏差（マイクロ秒）
なし	55.8	9.8
1 コア	57.5	34.4
2 コア	982.6	258.3

6.1 ノードの性能や特性に起因する遅延の計測

6.1.1 PTPNIC からの時刻取得実行時間

本システムを構成する各ノードにおいて，タスクの処理動作の判断基準となる現在時刻を取得する必要がある．ここでは，1 台の Exr ノードを用いて，PTPNIC から時刻取得にかかる時間を計測した．計測では，PTPNIC から 2 回連続で時刻を取得し，その差分から 1 回の時刻取得処理にかかる時間を計測した．計測は 1,000 回実行し，平均実行時間と標準偏差を計測した．今回使用した Exr ノードは 2 コアの CPU を有するため，タスクを実行している状態を想定し CPU 負荷を発生させた状態での実験も実施した．CPU 負荷を発生させない場合，1 コアに負荷を発生させた場合，2 コアに負荷を発生させた場合の 3 つの実験を行った．負荷の生成には，POSIX 準拠の負荷ジェネレータである StressAPI を使用した．StressAPI は，sqrt 計算を行うことで CPU 負荷を生成する．

結果を表 **2** に示す．CPU 負荷を発生させない場合では，平均実行時間は 4.0 マイクロ秒，標準偏差は 0.1 となった．CPU 負荷を発生させた場合，平均実行時間は 1 コアのときに 4.76 マイクロ秒，2 コアのときに 12.6 マイクロ秒と増加した．CPU 負荷が大きいほど標準偏差が大きくなっていることから，CPU 負荷に比例して処理時間の変動が大きくなっていることが分かる．

6.1.2 待機状態からの復帰にかかる時間

usleep システムコールで待機可能な最短時間を計測することで，1 度待機状態に入ってから処理を再開するまでに発生する遅延を計測した．計測では，usleep システムコールを実行する前後に時刻を取得し，2 つの差分を計算する．その差分から，6.1.1 項で求めた時刻取得にかかる時間の平均を減算し，算出した．試行を 1,000 回行った．6.1.1 項の実験と同様に，CPU 負荷を発生させない場合，1 コアに負荷を発生させた場合，2 コアに負荷を発生させた場合の 3 つの実験を行った．

結果を表 **3** に示す．CPU 負荷が発生していない状態では，平均実行時間は 55.8 マイクロ秒，標準偏差は 9.8 となっ

表 4 ノード間転送遅延評価結果
Table 4 Evaluation result of the transfer delay.

パケットサイズ	CPU 負荷無し		CPU 負荷 1 コア		CPU 負荷 2 コア	
	平均転送時間（マイクロ秒）	標準偏差（マイクロ秒）	平均転送時間（マイクロ秒）	標準偏差（マイクロ秒）	平均転送時間（マイクロ秒）	標準偏差（マイクロ秒）
64 Bytes	143.8	4.6	305.6	494.7	1,620.9	57,625.9
128 Bytes	145.6	4.5	318.3	500.4	2,904.8	3,760.1
256 Bytes	149.0	4.0	320.5	479.5	2,936.9	57,566.2
512 Bytes	154.8	4.6	330.1	441.0	557.5	4,483.2
1,024 Bytes	168.1	4.3	573.7	1,292.9	5,438.0	4,615.8
All	152.3	9.8	369.6	761.7	3,695.6	36,734.1

た．この結果から，本プロトタイプシステムでは，usleep システムコールで待機可能な最短時間よりも短い時間待機処理を行う場合，たとえば 20 マイクロ秒の待機処理を行う動作シナリオがあった場合，約 55.8 – 20 マイクロ秒の遅延が発生すると考えられる．そこで，このような動作シナリオが実行された場合，次の動作が前倒し可能ならば 20 マイクロ秒前倒しするか，不可能ならば 35.8 マイクロ秒後ろ倒しする調整が必要になる．また，1 コアのみに CPU 負荷を発生させた場合は，平均実行時間は 57.5 マイクロ秒であり，負荷がかかっていない状態とほぼ変わらないことが分かる．標準偏差が 34.4 マイクロ秒と大きくなっていることから，処理時間の変動は負荷をかけていない状態に比べて大きくなっていることが分かる．さらに，2 コアに CPU 負荷を発生させた場合，平均処理時間が 982.6 マイクロ秒と大幅に増加した．標準偏差も 258.3 マイクロ秒と大きくなり，処理時間の変動も大きいことが分かる．

6.2 ネットワーク的要因に起因する遅延の計測

6.2.1 ノード間の通信遅延

ノード間の 1 ホップの転送にかかる通信遅延の計測を，UDP 通信を用いて行った．パケットサイズは 64，128，256，512，1,024 Bytes とし，それぞれ 300 回，合計 1,500 回試行した．試行は，CPU 負荷を発生させない場合，1 コアに負荷を発生させた場合，2 コアに負荷を発生させた場合の 3 つの状況で実施した．

結果を**表 4** に示す．CPU 負荷を発生させない場合，64 Bytes のときは平均転送時間 143.8 マイクロ秒，標準偏差 4.6，1,024 Bytes のときでは平均転送時間は 168.1 マイクロ秒，標準偏差は 4.3 となっており，パケットサイズによって標準偏差はほぼ変わらないが，平均転送時間は変動することが分かる．また，CPU 負荷がかかっている状態では，1 コアに CPU 負荷を発生させた場合，64 Byte のパケットを転送する平均転送時間は 305.6 マイクロ秒，標準偏差は 494.7 となった．2 コアに発生させた場合は平均転送時間 1,620.9 マイクロ秒，標準偏差 57,625.9 マイクロ秒と，負荷の増加とともに非常に長い処理時間がかかり，処理時間の変動も大きいことが分かる．

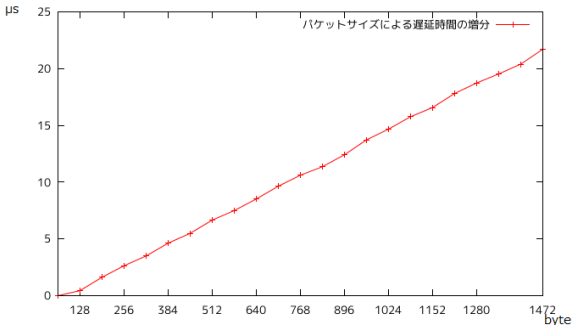


図 11 パケットサイズによる遅延時間の増分
Fig. 11 Difference of delay time due to packet size.

6.2.2 パケットサイズによる通信遅延への影響

イーサネット経由でノード間の通信を行う場合，転送するパケットが NIC で処理され，送出され初めてから最後のビットが送出されるまでの時間は，パケットサイズによって異なる．6.2.1 項の結果からも分かるように CPU 負荷の小さい場合は，同じサイズのパケットを送信した場合の転送時間の標準偏差は 4 マイクロ秒程度と小さく，平均転送時間はパケットサイズに比例して長くなっている．そのため，CPU 負荷が小さい場合，パケットサイズの違いによって発生する遅延についても，この評価結果をふまえた動作シナリオの微調整が可能であると考えられる．ここでは，パケットサイズを 64 Bytes から 1,472 Bytes まで 64 Bytes 刻みで通信にかかる時間を計測した．計測結果から，各パケットサイズのときの通信時間と 64 Bytes のときの通信時間との差分をとった結果を**図 11** に示す．この結果から，ノード間の通信遅延はパケットサイズに比例して増加し，64 Bytes 増加するごとに通信時間が約 1 マイクロ秒増加していることが分かる．最大のパケットサイズ 1,472 Bytes のときは，最小のパケットサイズ 64 Bytes の場合に比べて約 22 マイクロ秒長くなっている．

そこで，式 (3)，(6) を用いて動作シナリオの調整を行うとき，ネットワーク的な要因に起因する遅延 dn の値を，パケットサイズによる影響 pd と 64 Bytes のパケットを送ったときの通信遅延 td に分け，調整を行うこととする．pd は送信するパケットサイズを S，64 Bytes ごとに増加する遅

表 5 文字列編集距離計算タスクの動作シナリオ
Table 5 Scenario of the calculation of levenshtein distance.

ノード ID	データを供給するノード	次にデータを渡すノード	Tex(μs)	Tf(μs)	Ts(μs)	処理内容
Exr ノード 1	なし	Exr ノード 2	0	3,700	4,000	文字列生成・編集距離計算
Exr ノード 2	Exr ノード 1	Exr ノード 3	8,000	9,000	9,500	文字列編集距離計算
Exr ノード 3	Exr ノード 2	マネージャノード	10,000	11,000	11,300	計算結果を出力

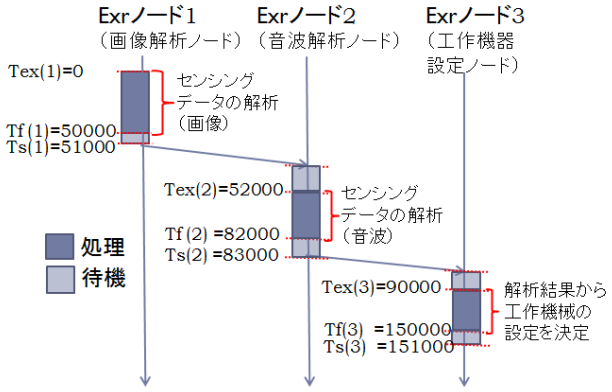


図 12 産業オートメーションにおけるシーケンス例
Fig. 12 Example of sequence of the automation control.

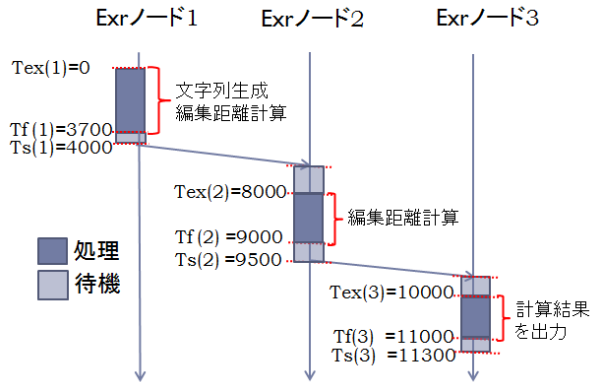


図 13 文字列編集距離計算タスクの処理シーケンス図
Fig. 13 Experiment sequence of the calculation of levenshtein distance.

延時間の増分を ad とすると，式 (9) で表すことができる．

$$pd = (S/64) \times ad \tag{9}$$

6.3 動作シナリオの調整

本論文で提案する動作シナリオの動的な調整を評価するにあたり，各 Exr ノードが実行する処理の完了までの時間が一定時間以内に収まるものと変動するものの 2 種類の処理を対象に，提案方式を評価した．

まず，提案方式の適用先の 1 つである，産業オートメーションを想定した処理フローの例を**図 12** に示す．この例では，タスクは 3 台の Exr ノード（画像解析ノード，音波解析ノード，工作機器設定ノード）で処理される．まず Exr ノード 1 が画像のセンシング結果から外形の解析を行い，Exr ノード 2 へ転送する．Exr ノード 2 は，Exr ノード 1 の解析結果と音波のセンシング結果から内部の解析を行い，解析した結果を Exr ノード 3 へ転送する．Exr ノード 3 は解析結果を用いて，工作時の設定を決定する．

この処理フローと同様の環境下で提案方式がどのように動作するかを評価するため，産業オートメーションの例で述べたシステム構成と同様の 3 台の Exr ノードを用いた分散処理制御実験を実施した．実験で用いるタスクは，文字列編集距離計算と AES 暗号化タスクである．これらのタスクは，本論文で提案する分散処理制御システムの適用先とは異なるが，タスクの処理特性として，各 Exr ノードにおける処理が一定時間以内に完了しやすいものと，変動をともなうものという，異なる処理特性のものとして選択している．従来の制御処理は，一定時間以内に完了するものが多いが，今後処理時間の変動をともない高度なデータ処理も制御処理として組み込まれてくることが予測される．そこで，処理時間が一定の従来型タスクと，処理時間変動をともなうタスクの 2 つについて，分散制御処理システム構成として評価を行うべく，上記のような構成とした．この 2 種類の処理特性のタスクをそれぞれ実行した場合の動作シナリオの動的な調整機構を評価することで，汎用的な有効性を示すことができると考える．なお，評価における初期動作シナリオの生成や，提案するモデルに従った動作シナリオの調整時に必要な Exr ノードの性能・特性は，6.1 および 6.2 節で述べた CPU 負荷をかけていない場合の実験結果を用いている．

6.3.1 文字列編集距離計算タスク実行実験

文字列編集距離の計算は，2 つの文字列の類似度を求めるものである．ここでは，一般的な動的計画法を用いた計算アルゴリズムを使用した．このアルゴリズムでは，文字数 i の文字列と文字数 j の文字列に対して計算を行う場合，$(i+1) \times (j+1)$ のサイズのスコアテーブルを順番に埋める．一般的に高い並列性を持つアルゴリズムとして知られており，画像認識のためのパターンマッチングや遺伝子解析を目的としたデータマイニングなどの分野において利用されている．

評価実験では，アルファベット 26 文字と数字 10 文字の組合せで，512 文字の 2 つの文字列を生成し，それら 2 つに対して編集距離の計算を行った．3 台の Exr ノードの動作シナリオは**表 5** のとおりとした．時刻は $Tex(1)$ 開始からの経過時間とし，単位はマイクロ秒である．この動作シナリオに基づいて，各 Exr ノードが動作した場合，**図 13** に示すような処理フローとなる．Exr ノード間で転送され

表 6 AES 暗号化タスクの動作シナリオ
Table 6 Scenario of the AES enciphered.

ノード ID	データを供給するノード	次にデータを渡すノード	Tex(μs)	Tf(μs)	Ts(μs)	処理内容
Exr ノード 1	なし	Exr ノード 2	0	3,000	4,000	暗号化用データ生成
Exr ノード 2	Exr ノード 1	Exr ノード 3	5,000	6,000	6,500	AES 暗号化
Exr ノード 3	Exr ノード 2	マネージャノード	7,000	8,000	8,300	計算結果を出力

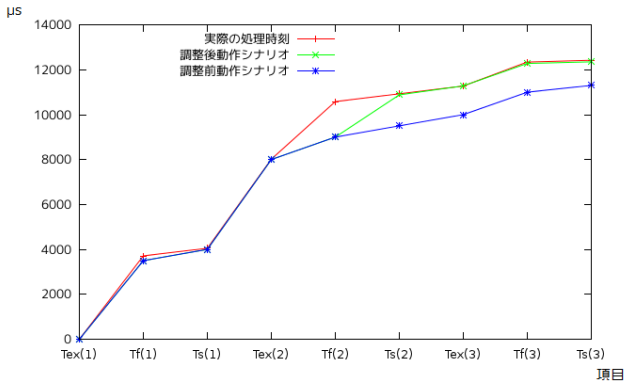


図 14 文字列編集距離計算タスクの実行結果
Fig. 14 Comparison figure of scenario and processing time of levenshtein distance calculation.

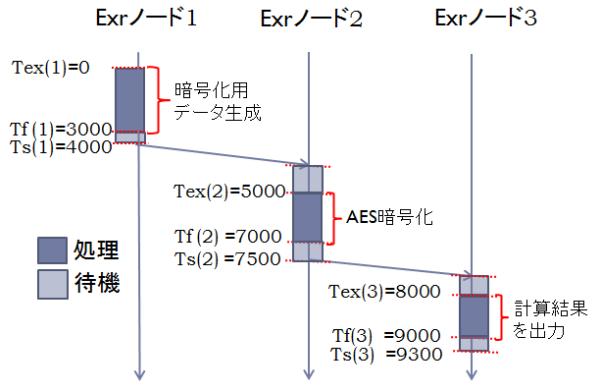


図 15 AES 暗号化タスクの処理シーケンス図
Fig. 15 Experiment sequence of AES encryption.

るパケットサイズは 1,024 Bytes とし，Exr ノード 1 の処理終了後に，Exr ノード 2 へ転送されるパケット数は 256 パケット，Exr ノード 2 の処理終了後に，Exr ノード 3 へ転送されるパケット数は 1 パケットとした．時刻取得にかかる時間 dl と，待機状態からの復帰にかかる時間 dt は 6.1.1，6.1.2 項の結果を使用し，dl は 4.0 マイクロ秒，dt は 55.8 マイクロ秒とした．また，Exr ノード 1 と Exr ノード 2 間のネットワーク適要因に起因する遅延 dn は 4,000 マイクロ秒とし，Exr ノード 2 と Exr ノード 3 間の遅延 dn は 500 マイクロ秒として初期動作シナリオを生成した．各 Exr ノードからマネージャノードを介して新しい動作シナリオを配布，変更するのにかかる時間 $M + mx$ は 300 マイクロ秒とした．

調整前後の動作シナリオと実際に Exr ノードで処理が実行された時刻の変化を**図 14** に示す．この結果から，タスク実行中に $Ts(2)$ 以降の動作シナリオが調整されたことが分かる．動作中に Exr ノード 2 が実行した処理が 1,581 マイクロ秒遅延し，動作シナリオに記述された時刻 9,000 マイクロ秒よりも遅い，10,581 マイクロ秒に終了している．そのため Exr ノード 2 は，処理が終了したタイミングで Ts の後ろ倒し調整モデルに従い，まず，$Ts(2)$ に対し後ろ倒しが行われる．Exr ノード 2 の処理が終了した時刻が 10,581 マイクロ秒であり，動作シナリオの配布と変更にかかる時間が 300 マイクロ秒であるため，$Ts(2)$ が，式 (1) を満たす 10,881 マイクロ秒に調整される．次に Tex の調整モデルに従い，$Tex(3)$ が式 (6) を満たさなくなる時刻 11,285 マイクロ秒に調整される．それにともない $Tf(3)$，$Ts(3)$ が後ろ倒しにされている．

6.3.2 AES 暗号化タスク実行実験

3 台の Exr ノードを用いて，生成されたデータを，AES 暗号化アルゴリズムを使用し暗号化するタスクを行った．3 台の Exr ノードの動作シナリオは**表 6** のとおりとした．このタスクの処理フローを**図 15** に示す．Exr ノード 1 で生成されたデータが，Exr ノード 2 へ転送され暗号化が行われる．その後 Exr ノード 3 へ転送され出力される．

AES 暗号化タスクでは，1,024 Bytes のデータに対して，256 bits の鍵を用いて暗号化を行った．時刻は $Tex(1)$ 開始からの経過時間とし，単位はマイクロ秒である．Exr ノード間で転送されるパケットサイズは 1,024 Bytes とし，Exr ノードの処理終了後に，次の Exr ノードへ転送されるパケット数は 1 パケットとした．また，各 Exr ノード 1 と Exr ノード 2 間のネットワーク適要因に起因する遅延 dn は 1,000 マイクロ秒，Exr ノード 2 と Exr ノード 3 間の dn は 500 マイクロ秒として初期動作シナリオを生成した．時刻取得にかかる時間 dl，待機状態からの復帰にかかる時間 dt，および動作シナリオの配布と変更にかかる時間 M は 6.3.1 項と同様の値を使用した．

調整前後の動作シナリオと実際に Exr ノードで処理が実行された時刻の変化を**図 16** に示す．この結果から，タスク実行中に $Ts(1)$ 以降の動作シナリオが調整されたことが分かる．まず，Exr ノード 1 の処理が動作シナリオに記述された時刻よりも早く終了したため，Ts の前倒し調整モデルに従い，式 (4)，式 (1)，および式 (6) を満たす時刻 3,163 マイクロ秒まで前倒しされ，それにともない $Tex(2)$，$Tf(2)$ も前倒しされている．

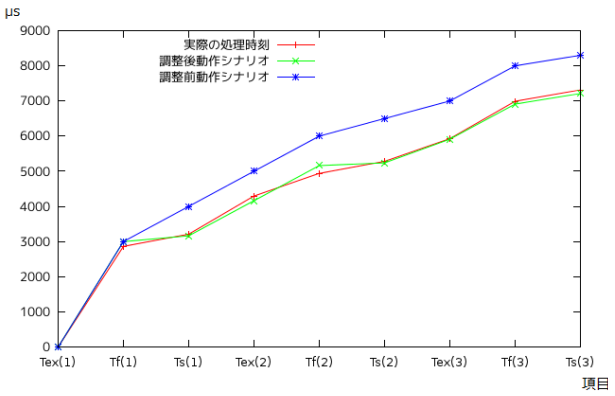


図 16 AES 暗号化タスクの実行結果

Fig. 16 Comparison figure of scenario and AES encryption processing time.

6.4 考察

6.4.1 遅延の計測に基づく考察

6.1，6.2 節の結果から，Exr ノードの負荷状況に応じて，割り当てられたタスクの実行や他の Exr ノードとの通信にかかる時間は大きく変動することが分かる．このことから，複数のタスク処理やデータの受信を同時に実行しないよう時刻によって制御することで，Exr ノードに与える外乱を取り除くことが可能であるといえる．一方，動作シナリオの変更処理では Exr ノードとの通信が必要となる．そのため複数のタスクを同時に処理している場合，タスクを処理している Exr ノードは，処理していない Exr ノードが動作シナリオの変更を行う場合よりも，変更にかかる時間を長く見積もる必要があると考えられる．

処理頻度と調整時間幅という点から考察すると，タスクが短い時間で完了する大量の処理から構成される場合，動作シナリオ調整にかかる遅延が問題になると考えられる．特に，ノード間の通信にかかる時間よりも短い時間で完了する処理から構成される場合，処理 1 つ 1 つに対して動作シナリオの調整を行うと，タスクの実行にかかる時間よりも，動作シナリオの調整にかかる時間のほうが長くなる．この問題に対する解決策として，初期動作シナリオを生成した段階で，処理時間が短い処理が続けてスケジューリングされた場合，1 処理ごとに動作シナリオの調整を行うのではなく，複数処理を実行した後に調整を行うことで，動作シナリオ調整にかかる時間を削減する方法が考えられる．

6.4.2 動作シナリオの調整実験結果に基づく考察

6.3.1 項の結果では，タスクの実行中に Exr ノード 2 で予期しない遅延が発生している．動作シナリオの調整を行わなかった場合，遅延の影響を受ける $Tf(2)$ 以降の処理は，動作シナリオから 1 ミリ秒以上ずれた時刻に実行されることが図 14 から確認できる．しかし，Exr ノード 2 に割り振られた処理が終了したタイミングで，動作シナリオの調整方式が動作することで，$Ts(2)$ 以降の動作は，調整後の動作シナリオに従って処理され，動作シナリオとの誤差数十マイクロ秒で処理を実行できていることが分かる．

タスク実行中の Exr ノード間で，動作シナリオから他の Exr ノードの動作状況を確認した場合，その結果が実際の処理状況とどの程度ずれているかという点から考えると，Exr ノード 2 の処理が終了したタイミングでは，調整前の動作シナリオを使用するため，遅延により 1 ミリ秒以上処理状況がずれていることになる．しかし，それ以外は，動作シナリオが適切に調整されているため，数十マイクロ秒単位のずれに収まっている．

さらに，Exr ノード 1 のデータ生成が終了したタイミングで，以降の動作シナリオが前倒しされている．前倒し処理の場合は，調整前の動作シナリオで，当該 Exr ノードが処理中である時刻に動作シナリオの調整を実行し，新しい動作シナリオが適用される．6.3.2 項の結果では，Exr ノード 1 から Exr ノード 2 へ通信を開始する時刻より前に，動作シナリオの調整が実行され，その後の処理も動作シナリオから大きくずれることなく処理されている．そのため，動作シナリオから他の Exr ノードの動作状況を確認した場合，動作シナリオから確認した動作状況と，実際の処理状況が大きくずれることはないと考えられる．

以上により，Exr ノードでの処理実行やデータ転送にかかる遅延が変動する環境においても，Exr ノードの処理状況を動的に動作シナリオに反映できていることを確認した．また，動作シナリオと同期時刻に基づいて，数十マイクロ秒単位で同期のとれた分散ノードの制御が実現できることを確認した．複数ノードに分散して実行されるタスクを，あらかじめ設定された時間以内に完了させることを可能とする分散制御が可能であるといえる．したがって，事前に決めた動作シナリオどおりに分散ノードで処理を実行していくことが難しかった従来システムに対しても，同期時刻と動作シナリオを用いた制御を適用可能であると考えられる．

7. まとめと今後の課題

本論文では，動的に調整される動作シナリオと同期時刻を用いたタイムアウェア型分散処理制御方式を提案した．本方式は，実際の動作状況に応じて動作シナリオを動的に調整することで処理中に遅延が発生，変動する環境においても，分散ノード群で行われるタスクの処理時間を一定時間に収めることが可能である．提案方式実現のために，タスクを現在時刻に準拠して実行するための動作シナリオを定義し，実際の処理状況に応じて動作シナリオを動的に調整する方式を開発した．さらに分散システム内で生じる遅延が同期時刻と動作シナリオに基づいた分散処理に及ぼす影響を検証し，動作シナリオの調整モデルを提案した．方式の有効性を検証するため，原理実験用のプロトタイプシステムを開発し，評価実験を行った．評価の結果，動作シナリオ調整方式が正常に稼働していることを確認し，処理

遅延が生じる環境においても数十マイクロ秒単位で同期のとれた分散制御が実現できることを確認した．

今後は，複数のタスクが同時に実行される場合における，動作シナリオ調整方式の性能評価が必要である．また，近年の一般的な PC のプロセッサはマルチコアであり，マルチスレッドで動作可能である．そのため，1 つの Exr ノードに，同時刻に複数の処理を割り振る場合，実行する処理ごとに割り当てるコア数を規定するなどの制御が必要であると考えられる．今回 1 タスクにつき 1 つの動作シナリオとして管理したが，複数のタスクを 1 つの動作シナリオにまとめ，システムが実行するタスクを一元管理する方法についても検討が必要である．さらに，動作シナリオの調整時にマネージャノードを用いずに新しい動作シナリオを共有する方式の検討や，高精度な時刻情報を用いた精度の高い処理負荷の推定方式を開発し，効率的な初期動作シナリオ生成手法の開発を行う．また提案方式を適用したシステムの開発も予定している．

謝辞 本研究の一部は，文部科学省特別経費「持続可能社会にむけた知的情報空間技術の創出」および JSPS 科研費基盤研究（C）25330067，若手研究（B）24700060 による支援を得た．ここに記して謝意を表する．

参考文献

[1] IEEE Standard 1588-2008: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Mesurement and Control Systems (2008).
[2] IEEE Standard 802.1AS-2011: IEEE Standard for Local and Metropolitan Area Networks (2011).
[3] 岡本 聡，荒川 豊，山中直明：ユビキタスグリッドネットワーキング環境（uGrid）の提案，電子情報通信学会ソサエティ大会，Vol.2007-2, No.B-7-15, p.75 (2007).
[4] 岡崎裕介，須佐雄輝，碓井亮太，荒川 豊，岡本 聡，山中直明：uGrid におけるダイナミック光パスを用いた映像サービスパーツ選択，電子情報通信学会技術研究報告 PN，フォトニックネットワーク，Vol.108, No.476, pp.29–34 (2009).
[5] 中原健太，菊田 洸，山田翔太，石井大介，岡本 聡，山中直明：ユビキタスネットワーキング環境（uGrid）におけるスケーラブルなサービス提供の実現へ向けたルーチングプロトコルの拡張，電子情報通信学会技術研究報告，ネットワークシステム，Vol.110, No.190, pp.49–54 (2010).
[6] Colombo, W.A.: IMC-AESOP project, IMC-AESOP project (online), available from ⟨http://www.imc-aesop.eu/⟩ (accessed 2014-09-22).
[7] Colombo, A.W., Bangemann, T. and Karnouskos, S.: IMC-AESOP Outcomes: Paving the way to Collaborative Manufacturing Systems, *INDIN2014* (2014).
[8] 特許庁：「プラントの制御・監視技術」の技術概要，特許庁（オンライン），入手先 ⟨https://www.jpo.go.jp/shiryou/s_sonota/hyoujun_gijutsu/plant/gaiyou.pdf⟩（参照 2014-09-22）.
[9] Ferrari, P., Flammini, A., Marioli, D. and Taroni, A.: IEEE 1588-Based Synchronization System for a Displacement Sensor Network, *IEEE Trans. Instrumentation and Measurement*, Vol.57, No.2, pp.254–260 (2008).
[10] del Rio, J., Toma, D., Shariat-Panahi, S., Manuel, A. and Ramos, H.G.: Precision timing in ocean sensor systems, *Measurement Science and Technology*, Vol.23, No.2, p.025801 (2012).
[11] 藤川冬樹：電力用通信網における高精度時刻同期方式の適用検討：IEEE1588 に基づく広域時刻同期網の評価，電力中央研究所報告 R（11029），pp.1–3 (2012).
[12] Harris, K.R., Balasubramanian, S. and Moldovansky, A.: The Application of IEEE 1588 to a Distributed Motion Control System, *ODVA CIP Networks Conference*, pp.16–18 (2004).
[13] Harris, K.: An Application of IEEE 1588 to Industrial Automation, *ISPCS2008*, pp.71–76 (2008).
[14] 鈴木 誠，猿渡俊介，南 正輝，森川博之：無線センサネットワークにおける時刻同期技術の研究動向，森川研究室技術研究報告書 (2008).
[15] 小泉 稔，江端智一，堤 智昭，大島浩太，寺田松昭：高精度時刻同期を特徴とする分散型モバイルネットワークエミュレータ，情報処理学会論文誌，Vol.53, No.2, pp.754–769 (2012).
[16] 堤 智昭，大島浩太，寺田松昭：IEEE1588 による高精度時刻同期を特徴とした分散型モバイルネットワークエミュレータの設計と実装，マルチメディア，分散，協調とモバイル（DICOMO2012）シンポジウム，pp.914–920 (2012).
[17] Tsutsumi, T., Koizumi, M., Ebata, T., Ohshima, K. and Terada, M.: Performance Evaluation of Synchronous Distributed Wireless Network Emulator for High-Speed Mobility, *ICOIN2013*, pp.151–156 (2013).

堤 智昭 （学生会員）

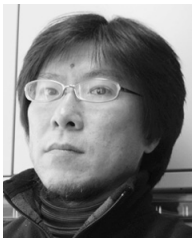
2010 年東京農工大学工学部情報工学科卒業．2012 年同大学大学院工学府博士前期課程修了．現在，同大学院工学府博士後期課程在学．アドホックネットワーク，モバイルネットワークエミュレータ，時刻情報応用システムに関する研究に従事．

大島 浩太 （正会員）

2003 年東京農工大学大学院工学研究科電子情報工学専攻博士前期課程修了．2006 年同大学院工学教育部電子情報工学専攻博士後期課程修了．博士（工学）．2006 年東京農工大学大学院工学研究院助教．現在，埼玉工業大学工学部講師．無線センサネットワーク，異種ネットワーク連携，オーバレイネットワーク等の研究に従事．電子情報通信学会会員．

小泉 稔 （正会員）

1981 年東京大学大学院工学系研究科計数工学専攻（修士）修了，同年（株）日立製作所システム開発研究所入所．自律分散システム，情報制御ネットワークシステムの研究開発に従事．2006 年 10 月より Hitachi Europe Ltd. 勤務，2013 年 4 月より同社横浜研究所主管研究員．著書『ポリシーベースによる QoS 制御』（共著，オーム社），博士（工学）．計測自動制御学会，電気学会，電子情報通信学会各会員．



中條 拓伯 （正会員）

1961 年生まれ．1985 年神戸大学工学部電気工学科卒業．1987 年同大学大学院工学研究科修了．1989 年同大学工学部助手の後，1998 年より 1 年間 Illinois 大学 Urbana-Champaign 校 Center for Supercomputing Research and Development（CSRD）にて Visiting Research Assistant Professor を経て，現在，東京農工大学大学院工学研究院准教授．プロセッサアーキテクチャ，並列処理，リコンフィギャラブルコンピューティングに関する研究に従事．電子情報通信学会，IEEE CS，ACM 各会員．博士（工学）．