Vol.2017-MBL-82 No.7
Vol.2017-UBI-53 No.7
2017/3/9

通信環境の変化に対してロバストな分散データ共有ミドルウェアの提案

今井岳[†1, a)]　可児潤也[†1]　木原英人[†1]　由良淳一[†1]　植木美和[†1]　武理一郎[†1]

概要：スマート端末や IoT デバイスの普及やクラウド技術・通信技術の発展により，ネットワーク上の様々な機器やシステムが連携・協調するシステムが実現可能になってきている．本稿では，人との接点である UI やロボットなどの高い応答性が要求されるフロントシステムにおいて，機器やシステムが連携する分散型のシステムを構築するための，通信ミドルウェアを提案する．フロントの分散システムは，今後，連携するデバイスやシステムの増加に伴いトラフィックが増加し，通信能力や計算能力の限界によりシステムが破綻する可能性がある．提案ミドルウェアでは，送信が滞る場合にデータの全ての更新を通知せずに，最終結果が一致する結果整合性を保証する仮想的な共有データスペースを提供する事により，フロントの分散システムの構築を容易にする．通信が不安定で帯域も限定的なモバイル環境や，動的に連携する機器が変更される場合においても，連携する機器やシステムに同じデータが配信されるため，どのような通信環境においてもシステムとして一貫した動作を行うことが出来る．

キーワード：分散システム，フロントシステム，モバイル環境，結果整合性，通信抑制

Distributed data sharing middleware for unstable communication environments

TAKASHI IMAI[†1, a)], JUNYA KANI[†1], HIDETO KIHARA[†1], JUNICHI YURA[†1], MIWA UEKI[†1], TAKE RIICHIRO[†1]

1. はじめに

スマート端末の普及や無線技術の進歩により，場所を選ばずに ICT サービスを利用できるようになり，ICT サービスは利用者の生活に欠かせないものとなってきた. さらに，センサや家電などの様々な機器がインターネットに接続される Internet of Things（IoT）という概念が注目されており，2020 年には 300 億個の電子機器やセンサがネットワークに接続されると予測されている**エラー! 参照元が見つかりません。**. 今後は，このようなネットワークで接続された様々な機器や、クラウドサービスが連携・協調して，様々な場面で人を支援するシステムが実現可能になってきた.

本稿では，人との接点である UI やロボットなどの高い応答性が要求されるフロントシステムにおいて，機器やシステムが連携する分散型のシステムを構築するための，通信ミドルウェアを提案する．フロントの分散システムにおいて，分散システムを構成する機器やシステム（ノードという）が連携するためには，複雑な通信が必要となってくる．特に，今後想定される，モバイル端末や IoT デバイスとの連携を想定すると，フロントの分散システムには次のような要件がある.

(1) **応答性**

人と接するフロントシステムの場合，ユーザに対する高い応答性が要求される．そのため，システムが動作するのに必要な情報が少ない遅延で通知される必要がある．一方で，古くなった情報は重要性が低い場合もある．今必要な情報を遅延なく通信することにより，システムが正しい応答を遅延なく返すことができる.

(2) **通信状態の変化に対するロバスト性**

モバイル端末や IoT デバイスは一般的に無線環境で通信される．無線通信は通信状態が不安定であり，状況によって利用可能な通信帯域が大きく変動し，通信が切断されることも頻繁に発生する．このような状況においても，システムとして安定した挙動が求められる.

(3) **ノード数に対するスケーラビリティ**

IoT 機器の増加によって，今後連携する機器の数が増加することが予想される．連携する機器間で単純に情報をマルチキャストする方法では，トラフィック量はノード数に対して指数関数的に増加する．ノード数が増えても破綻しないような連携方法が必要となる.

(4) **動的な構成変更に対するシステムの適応**

モバイル端末や IoT デバイスは物理的に移動することが可能である．例えば，その場に集まった端末やデバイスが動的に連携するようなケースも想定される．このように、分散システムが動的に構成される場合や，ノードの参加・離脱があった場合にも，各ノードが一貫した挙動ができるような連携の仕組みが必要である.

†1 株式会社富士通研究所
Fujitsu Laboratories Ltd.
a) imai.takashi@jp.fujitsu.com

Vol.2017-MBL-82 No.7
Vol.2017-UBI-53 No.7
2017/3/9

これらの要件は，フロントの分散システムの共通の課題であり，個々のシステムでこれらの要件を満たすような開発を行うことは困難であった．そこで，本ミドルウェアでは，連携する個々の対象や通信状態を意識することなく、フロントの分散システムを開発可能にすることを目指している．

提案するミドルウェアでは，上述した 4 つの要件を，アプリケーションから分離するため，分散アプリケーション間で共有される，仮想的な共有データスペースを提供する．端末・機器等の分散ノード上で動作する分散アプリケーションが情報を相互に操作・参照するような分散システムの構築に適している．本ミドルウェアの主な用途として，ドキュメントや設計図を複数の端末から共同編集するシステムや，センシング情報を共有しながら同じ目的をもって動作する群ロボットシステムなどがあげられる．

本論文の構成を以下に述べる．2 章で，提案ミドルウェアの方針と設計について詳しく述べ，3 章では，評価および考察を行う．4 章で関連研究との比較を行い，5 章でまとめと今後の課題について述べる．

2. 提案ミドルウェア

2.1 基本方針

前章で述べたように，通信上の 4 つの要件を，アプリケーションから分離するため，分散ノードで動作するアプリケーション（本稿では分散アプリケーションと呼ぶ）に対して，仮想の共有データスペースを提供する．各分散アプリケーションは，共有データスペースのデータに対する操作と，データ更新通知の受信により他のノードや通信を意識せずに連携することが可能である．逆にミドルウェアはアプリケーションに意識させずにこれらの要件に対応することができる．図 1 に仮想共有データスペースの概念図を示す．

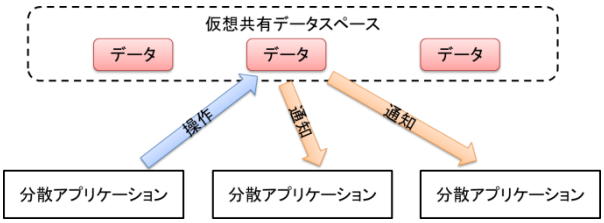

図 1　仮想共有データスペースの概念図

次項以降では，これらの要件を満たすためのミドルウェアの設計方針について述べる．

2.1.1 マスターレス

センサや操作の情報の共有など、高頻度の通信を行う場合、特定のノードやサーバが共有データスペースを提供する方式では，単一障害点となるサーバに処理や通信が集中し，システム全体のボトルネックとなるため，トラフィックの破綻や遅延を引き起こす．(1) と (3) の要件を考慮して，本ミドルウェアでは，特定のサーバを必要としない構成をとる．すなわち，各ノードがデータスペースを保持し，各ノードにおけるデータの操作を必要に応じてノード間で相互に通知し，同期することにより，仮想的な共有データスペースを提供する．

2.1.2 通信状態に応じたトラフィックの抑制

本ミドルウェアでは，各分散アプリケーションによるデータの操作を通知しあう．しかしながら、前章でも述べた通り，通信状態が悪化した場合や，ノード数が増加した場合に，トラフィック量が通信能力の限界を超えてしまい，通信が破綻することが想定される．(2) と (3) の要件を考慮して，通信状態を良好に保ち，システムを維持することを優先し，通信状態によっては全てのデータ操作を通知しない．

2.1.3 結果整合性

共有データスペースに基づき動作する分散システムにおいて，システムが一貫した動作を行うには、分散ノードに対して一貫性のある情報を提供する必要がある．しかし，通信状態の悪化や一時的な切断などのトラブルや，ノードの途中参加・離脱など，システムの構成変更があった場合には，前項で述べたような通信の抑制や，失敗が起こるため，各ノードが，保持する情報にばらつきがでる．(4) の要件を考慮して，そのような状況においても，分散システムに一貫した挙動を可能とするため，本ミドルウェアが提供する共有データスペースは，結果整合性[2]を保証する．すなわち，各瞬間では各ノードが異なるデータに基づき動作することを許容するが，一定時間操作がなかった場合に，最終的には各ノードがもつデータが一致している事を保証する．また，このようなポリシーを持つことにより，トラフィックの抑制に関しても，最終結果に影響しないデータの配信を抑制するなど，効率的に実施できる．

結果整合性を保証するためには以下が必要である．

- ノード間の同時データ操作による競合の解決
- 一時的な切断や送信エラーに対する再送
- ノードの離脱・参加を考慮した同期管理の仕組み

2.2 ミドルウェアの構成

図 2 に前節で述べた基本方針に基づく，本ミドルウェアの概念図を示す．本ミドルウェアは各ノードに配備され，マスターレスで動作する．ミドルウェア層でローカルのデータスペースを同期する事により，分散アプリケーションに対して，仮想的な共有データスペースを提供する．同期は通信状態などにより，滞ることがあるが，各ノードが最終的に一致した情報を参照できることを保証する．

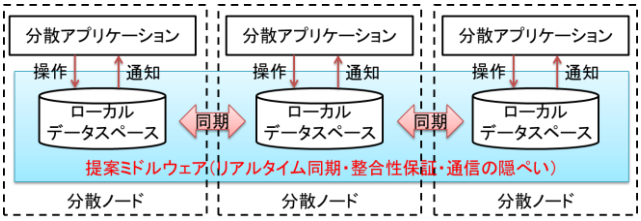

図 2　提案ミドルウェアの概念図

本ミドルウェアは図 3 に示す通り，データ管理，同期管理，トラフィック制御の主に 3 つの機能ブロックで構成される．

データ管理機能は，同期されたローカルデータと，同期のためにノード間で共有されるシステムデータを管理し，アプリケーションに対するインターフェースを提供する．同期管理機能は，現在のデータの更新状況と，他のノードからの同期の要求に応じて，どのデータをどのノードに送信すべきかを決定する．同期管理機能による送信要求は，いったんトラフィック制御機能により保持され，通信状態などに基づき、適切なタイミングで送信される．

通信層は本ミドルウェアから切り離す構成としている．ノード間の実際の通信にメッセージブローカを用いるか，ピアツーピア（P2P）による通信を行うかは，分散システムの設計者によって選択することができる．

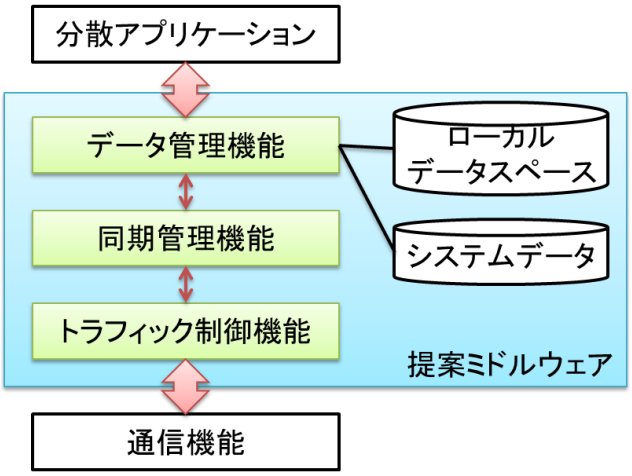

図 3　提案ミドルウェアの構成

2.2.1 アプリケーションからのインターフェース

本ミドルウェアがアプリケーションに対して提供する主要なインターフェースを表 1 に示す．本ミドルウェアは、目的や用途に応じて異なる単位で共有データスペースを構築できるよう、チャネルという共有単位を設けた。チャネルを用いることで、各ノードは複数の独立したデータスペースを扱うことができる。

仮想共有データスペースはアプリケーションからキーバリューストア（KVS）のように利用することができる．一般的な KVS と同じく値の設定、値の取得ができることに加えて、本ミドルウェアは，相互のやり取りを行う分散アプリケーションに必要となる，値の更新の通知を受け取る機能を提供する．変更検知機能では，階層構造を持ったデータをより扱いやすくするため，キーにドメイン名のような階層構造をもたせ、上位のキーで配下のキーを subscribe することができるようにしている．

表 1　提案ミドルウェアの主要なインターフェース

メソッド	説明
open	接続する仮想共有データスペースのチャネル名や利用する通信機能に必要なパラメタを指定して，仮想共有データスペースの利用を開始する
close	仮想共有データスペースの利用を終了する
put	キーと値を指定して仮想共有データスペースに対して値を書き込む
get	キーを指定して仮想共有データスペースにから値を読み出す
subscribe	key を指定し，key に対する情報が更新されると通知を受け取ることができる
unsubscribe	subscribe 時に返される id を指定し，subscribe を解除する

2.2.2 通信機能の要件

本ミドルウェアはチャネルに参加するノードの管理は通信機能に依存している．通信機能として，必要な機能は，以下である．

- 特定のチャネルに参加するノードの一覧を取得
- 特定のチャネルへの参加・離脱
- 特定のチャネルに参加・離脱するノードを通知
- ノードを指定してデータを送信

上記の機能を持っている通信手段であれば，簡単なラッパーを実装するだけで本ミドルウェアから利用することができる．また，結果整合性の為，チャネルへのノードの参加や離脱の通知について，タイミングのばらつきは許容されるが，最終的に全ノードに正しく通知されることが必要である．

2.3 データ同期の基本動作

基本的な動きは pub/sub 型のメッセージと同様である．すなわち，データの更新があったノードは，subscribe しているノードに対してデータを配布する．

自ノードでデータ操作が行われ，最新データを保持するノードは、そのデータに関して配布責任を負う．配布責任を負ったノードは，そのデータを subscribe する全ノードに確実にデータを配布する責任を負う．その為，送信に失敗した場合には再送を行う．また，新たに subscribe するノー

ドが現れたら，データを配布しなければいけない（図 4）．他のノードでデータの操作が発生した場合には，一時的に複数のノードが配布責任を負う状態になる（図 5）．配布責任を負うノードは，該当のデータが他ノードにより更新された場合には，全てのノードへの配布が完了していなくても配布責任を終了する（図 6）．

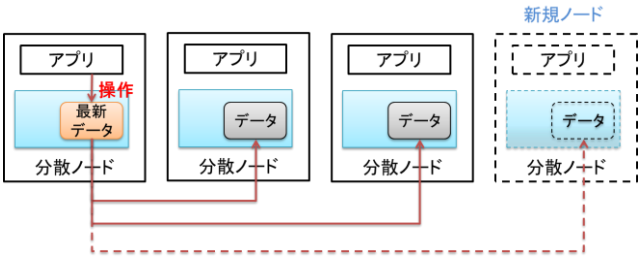

図 4　配布責任の概念図

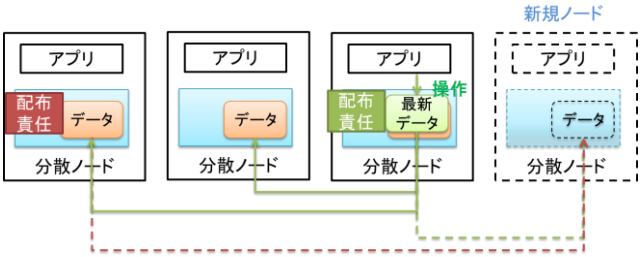

図 5　配布責任の重複状態

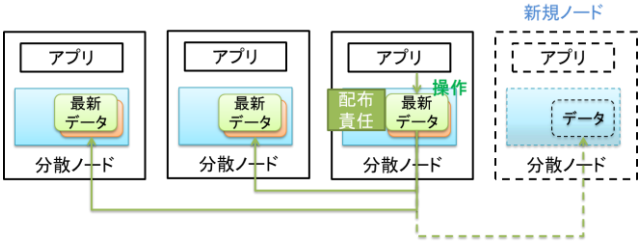

図 6　配布責任の終了

このように一時的な配布責任の重複を許しながら，配布責任を次々に変更する事により最新の情報を全てのノードに配信する．

2.3.1 システムデータの配布

配布責任ノードが subscribe しているノードに対してデータを配布するには，どのノードが該当の key を subscribe しているかを知っている必要がある．そのため，この subscribe 情報を予め全ノードで共有しておく必要がある．このような全ノードが共有するデータを本ミドルウェアではシステムデータと呼んでいる．システムデータは各ノードに属する情報であり，システムデータが更新された場合，各ノードはチャネルに参加する全てのノードにシステムデータの更新を配布する．送信に失敗した場合には成功するまで再送し，新規参加ノードに対しても送信する事により，全てのノードが情報を共有できる．配布されたシステムデータは，受信した各ノードで送信元のノードごとに管理される．このようにして配布されるデータは，各瞬間ではノード間で異なる値を保持している可能性があるが，更新がない状態で，十分に時間が経過すると全てのノードで同じ値になる．

subscribe 情報として共有されるのは以下の情報である．

- ノード ID
- Subscribe 対象 key

2.3.2 共有データの競合解決

ここまでで述べたデータの配布の方式では，各ノードが自身の最終更新を通知するが，更新は各ノードで非同期に行われるため，同じタイミングで更新が発生した場合や，通信状態により，更新の通知が遅延する場合に正しい順序で更新データが届くとは限らない．そのため，各データには更新時刻を表すクロックを付与し，受信したノードが，自身の持つデータとクロックを比較するだけで最新のデータを判断可能にする．これにより全てのノードからの情報が届いた時点で，最終結果を一致させることができる．更新時刻を表すクロックは，論理クロックとしてよく利用されるランポートクロック[2]と OS のシステムクロックから選択して利用できる．システムクロックを用いる場合には，連携するノード間で予め時刻を合わせる必要がある．クロックが同じ値の場合には，更新したノード ID の値を比較するなど，各ノードが同じ基準で判断する方法をもつことにより最終結果を一致させる．

2.4 ノードの離脱時の整合性

前節で説明した同期方法は，配布責任を負ったノードが再送や新規ノードへの配布の役割を果たすことを前提としている．そのため，配布責任ノードが離脱した場合や切断された場合には，最終結果を一致させることができない．そこで，ノードの離脱や切断が発生した場合には，該当ノードの配布責任を他のノードが引き継ぐ必要がある．

2.4.1 基本方針

ノードの離脱を検出された際に，残された分散ノード間で新たな配布責任ノードを決定する必要がある．本ミドルウェアは，各ノードは，チャネルに参加するノードの正しいリストを最終的には知ることができることを前提としている．そのため，ノードリストが決まれば一意にノードを選択する手段を持つことにより，最終的には，各ノードが同じ引継ぎノードを決定することができる．しかしながら，例えばハートビートによりノードの存在を確認する場合など，通信機能の性質によっては，ノードの離脱の検出にかかる時間が大きくばらつき，引継ぎノードが最終的に一致するまでに時間がかかる可能性がある．ここで，配布ノードや引継ぎノードが整合しているかどうかはアプリケーシ

ョンにとっては重要ではない．引継ぎノードが最終的な一致に至らない場合においても，できるだけ早く共有データの整合させたい．そこで，一時的な冗長な配信を許容し，各ノードが把握する参加ノードのリストが一致しない場合においても，配布責任が機能するようする．配布責任ノードの役割は多くの場合，新規参加ノードにデータを配布する事であり，重複配信によるトラフィックの増加の影響は限定的である．

このような方針により引継ぎ処理を行う場合，以下を考慮する必要がある．

(1) **誰も引継ぎを開始しないケース**

各ノードが自身のもつノードリストから引継ぎ先を判断した場合，全てのノードが引き継ぐ必要がないと判断する可能性がある．

(2) **引継ぎノードが配布すべきデータの存在を知らないケース**

各ノードは自身が subscribe しているデータのみを他ノードから受け取り保持する．その為，subscribe していない場合や，subscribe していたとしても，まだデータが到達していない場合に，引き継ぐべきデータの存在を知らない可能性がある．

(3) **引継ぎノードが最新データを保持していないケース**

離脱したノードが全てのノードに最新データを送信済みとは限らない．引継ぎノードが，データの存在を正しく把握できたとしても，保持するデータが最新ではない可能性がある．

(4) **引継ぎの引継ぎ**

配布責任を引き継いだノードが離脱した場合にも再引継ぎを実施する必要がある．

2.4.2 ノード離脱時の配布責任引継ぎ処理

前項で述べた方針に基づくノード離脱時の引継ぎ処理について述べる．以下の機能を実装する事により，各ノードが持つデータを最終的に一致させることができる．

(1) **引継ぎ情報を共有**

前項で述べた，様々な不整合を解消するために，各ノードは，引継ぎにより配布を実施している旨を全ノードで共有する．これにより，各ノードは自身が保持するデータについて，誰からも引き継がれていない状態や，重複して引き継がれている状態を把握することができる．共有する情報は，引継ぎ情報は以下である．

- 引継ぎノードの ID
- 引継ぎ対象のデータの key
- 引継ぎ対象のデータのクロック
- 本来の配布責任ノードの ID

これらの情報は，2.3.1 で説明したシステムデータとして配布する．

(2) **引継ぎによる配布の開始と終了**

まず，引継ぎ開始の判断について述べる．基本的には各ノードが同じ仕組みでノードのリストから引継ぎノードを選択するため，ノードの離脱の検出時には，各ノードが通信することなく，引継ぎによる配布の開始を判断することが出来る．また，保持するデータから離脱ノードが配布責任を持つデータ（引継ぎを行った場合を含む）を抽出する事により，配布を引き継ぐべきデータの key を知ることが出来る．ノードの復帰や，別のノードによるデータ更新が発生したら引継ぎによる配布を終了する．それに加え、前項で議論した考慮点に対応するため，自身が配布しない場合には，引き継ぐべきノードに対して、引き継ぐべき key を通知して引継ぎを依頼し，依頼されたノードは配信を開始する．依頼したノードは，引継ぎ情報の共有により，依頼が実行されたかどうかを知ることができるため，定期的に状態を確認し，整合した状態になるまで，一連の処理を繰り返し実施する．

次に，引継ぎによる配布を終了するタイミングについて述べる．ノードの参加・離脱や引継ぎ情報の受信により引継ぎの状況が変化したタイミングで，ノードリストから再度引継ぎ対象を判断し，他に引き継ぐべきノードが存在し，かつ共有された引継ぎ情報により，正しいノードが引き継いだ事が確認できている場合に，引継ぎによる配信を終了する．また，離脱したノードが復帰した場合や、該当の key のデータが他のノードにより更新された場合にも引継ぎの役割を終了する．引継ぎの役割を終了する際にはシステム情報から引継ぎ情報を削除する．

このような仕組みにより，引継ぎノードが一意に定まらない期間も配布を継続して，ノード間のデータの整合性を保つことが出来る．

(3) **最新値の同期**

引き継いだノードが最新のデータを保持していない場合の対処として以下の処理を行う．

① (1)で述べた通り，引継ぎ情報を受け取った際に，引き継いだノードが持つ各データのクロックを同時に受け取る．この時，自身がこのクロックより更に新しいデータを保持していた場合に，引継ぎノードに対して最新情報を通知する．

② (2)で述べた，引継ぎの依頼を受信した際に，通知された key の情報を保持していない場合には，依頼元に該当のデータを送ることを要求する．

2.5 トラフィック状態に応じた通信の抑制

本ミドルウェアにおいて，通信量の抑制は，データの更新時に発生する同期のための通信を一時的に保留する事により行われる．保留している間に，自他のノードによりデータが更新された場合に，保留していたデータの送信の必要はなくなる．どの様な場合にどのようなデータの同期を保留するかの判断を適切にすることにより，通信の効率を

向上することができる．

この判断をより適切に行う事は今後の研究課題であるが，まずは，通信状態，更新頻度，データサイズなどの基本的な情報に基づく通信の抑制を考える．本ミドルウェアでは，現状以下のような方法で通信を抑制している．

① 送信対象ノードごとに送信中のデータが一定量を超えた場合に通信を保留する
② Subscribe するアプリの要求に応じて優先度の高い要求に対して優先的に送信する
③ 自他ノードからの更新頻度の高いデータは送信してもすぐに古くなる可能性高いため，送信の優先度を下げる．
④ 通信状態が悪化した場合には，優先度の低い通信を保留する

3. 実装と評価

3.1 実装

web ベースの分散アプリケーションを想定し，本ミドルウェアを，javascript のライブラリとして開発した．通信機能として，以下の 3 つの方式を試作した．

- websocket / long-polling：socket.io[3]を利用
- webRTC：PeerJS[4]を利用
- P2P（tcp）: cordova[5]プラグインから利用可能（android のみ）

3.2 整合性評価

実装したミドルウェアが意図通り動作していることを確認するため，通信が不安定な状況での利用を想定した実験を行い，最終結果が整合する事、および整合するまでの時間を測定した．評価には本ミドルウェアの評価用に開発した専用の評価ツール[6]を利用した．図 7 に評価実験の構成を示す。

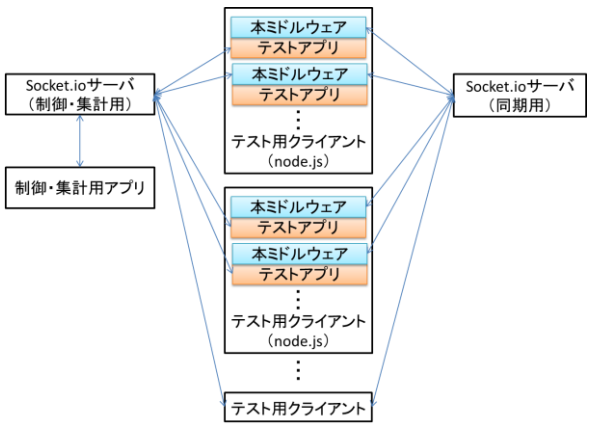

図 7　評価実験の構成

評価システムでは，Node.js[7]上で動作させるテストクライアントを複数プロセス動作させる．各テストクライアントでは，複数のテストアプリが動作し，各テストアプリは本ミドルウェアを利用して情報を共有する．各テストアプリはテストが開始されると，定期的に本ミドルウェアを利用して共有する情報を書き込む．制御・集計用アプリは，各テストアプリと通信し，テストの開始などのコントロールや，結果の集計を行う．

不安定な通信環境を再現するために，各テストクライアントは socket.io サーバとの接続をランダムに強制的切断する．また，key に対する subscribe/unsubscribe もランダムに実施し，最終整合性は各 key に対して、その時点で subscribe しているノード間で一致しているかどうかを確認する．

表 2 に実験条件の詳細を示す．すなわち，20 ノードが 10key を操作・参照しており，各ノードは 10ms 毎にランダムに 5 つの key に 1000byte のデータを書き込む．また，0~5 秒,0~1 秒の間隔でランダムな切断／再接続を繰り返し，0~3 秒の間隔で subscribe と unsubscribe を繰り返す．20 秒経過後にテストアプリは操作をやめ，集計アプリが各ノードから通知される情報に基づき，整合性の確認と，整合時間の集約を行う．実際にはこのような条件で動作するユースケースはないと考えられるが、整合性を確認する上で厳しい条件を設定した．また，今回の実験では，4 つのテスト用クライアント，2 つの socket.io サーバ，制御・集計用アプリ共に一つの PC（Windows 8.1(64bit), Intel® Xeon CPU E5-1650v3 @ 3.50GHz 3.50GHz 16.0GB RAM）上で動作させた．

表 2　評価条件

クライアント数	4
テストアプリ数（ノード数）	20
Key 数	10
データサイズ	1000byte
書き込み間隔	10ms
書き込み key 数	5
ランダム切断	0~5 秒に一度切断 0~1 秒で再接続
ランダム subscribe	0~3 秒後に subscribe 0~3 秒後に unsubscribe
テスト時間	20 秒

同じ条件で 10 回の測定を行い，10 回とも整合しているころを確認した．また，整合にかかる時間は，平均 611ms，最大 701ms であった．実験した複雑な条件においても，短い時間で最終結果を整合できることを確認した．

3.3 空間 UI システムへの適用

本ミドルウェアは，当研究所で研究している空間 UI システム[8]に採用され，ディスプレイ間でのリアルタイムな

連携に活用されている[9]. 本ミドルウェアのような通信状態の変化に対してロバストな分散データ共有の仕組みが，空間 UI のような共同編集システムには有用である事が示された[6].

4. 関連研究

ノード間でデータを共有する方法として，よく利用されるのが，出版・購読型（Pub/Sub 型）のメッセージ指向ミドルウェアである．Pub/Sub 型データ配信モデルのプロトコルの仕様である MQ Telemetry Transport（MQTT）[10]や，OMG によって標準化されているミドルウェアの仕様である Data Distribution Service（DDS）[11]では，様々な通信状態に対応するため，QoS（Quality of Service)のパラメータを調整する手段を提供している．特に DDS は，メッセージブローカが不要な分散型のミドルウェアであり，提案ミドルウェアと近い構成をとる. pub/sub 型の汎用性が高い仕様であるが，配布者と受信者の役割が分かれている pub/sub 型の仕組みでは, パブリッシャが離脱してしまった場合など，動的な構成変更時の整合性を保つことは難しいと思われる．一方で, DDS は多様な QoS の仕組みを持っており[12],今後，通信の抑制機能を高度化するにあたり，参考になると考えている．

5. まとめと今後の課題

本稿では以下の特徴を持つ通信環境の変化に対してロバストなデータ共有ミドルウェアを提案した．

- マスターレス
- 通信抑制
- 結果整合性の保証

本稿では特に，分散アプリケーション間で同じ情報を相互に書き込むケースを想定し，結果整合性を実現するための方法について詳しく述べた．また，簡単な評価により、短い時間で最終的な結果が一致する事を確認した．

今後は以下の改善に取り組んでいく予定である．

(1) 階層化

本ミドルウェアは通信機能により，各ノードが全てのノードの存在を知ることが出来ることが前提となっているが，連携するノードが増加した際に，各ノードの処理が増加する．その為，データ共有するノードを複数のグループに分割し, さらにグループ間で同期する仕組みを検討している．

(2) 通信抑制機能の高度化

通信抑制機能については, 現状は通信状況が悪化した際に，優先度の低い通信を抑制する機能のみ実現している．更新頻度を考慮した最適な抑制方法の考案や，データの属性等も考慮した通信の抑制機能が今後の研究課題である．

参考文献

[1] 総務省: インターネットに接続する様々なモノの拡大，http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h28/html/nc121100.html (参照 2017-02-01)
[2] アンドリュー・S 他．水野他（訳）：分散システム 原理とパラダイム 第 2 版．ピアソン・エデュケーション, 2009
[3] Socket.io: Socket.io, http://socket.io/
[4] PeerJS: PeerJS, http://peerjs.com/
[5] Apache Cordova: Apache Cordova, https://cordova.apache.org/
[6] 可児，他: 通信負荷にロバストな共同編集システムの評価，情報処理学会研究報告，情報処理学会 (2017)
[7] Node.js: Node.js, https://nodejs.org/
[8] 富士通研究所: 部屋全体をまるごとデジタル化する UI 技術を開発し、ICT による共創支援の実証実験を開始，http://pr.fujitsu.com/jp/news/2015/07/27.html, (参照 2017-02-01)
[9] 板倉，他: 複数ディスプレイをシームレスに統合するウェブブラウザベース分散システム，情報処理学会研究報告，情報処理学会 (2017)
[10] IBM: MQ Telemetry Transport (MQTT) V3.1 プロトコル仕様，http://www.ibm.com/developerworks/jp/websphere/library/wmq/mqtt31_spec/ (参照 2017-02-01)
[11] OMG: DDS Portal, http://portals.omg.org/dds/, (参照 2017-02-01)
[12] HéctorPérez, and J.JavierGutiérrez, Modeling the QoS parameters of DDS for event-driven real-time applications, The Journal of Systems and Software 104 (2015) 126-160, 2015