Vol.2017-MBL-82 No.8
Vol.2017-UBI-53 No.8
2017/3/9

通信負荷にロバストな共同編集システムの評価

可児 潤也[†1, a)]　板倉 宏太[†1]　今井 岳[†1]　木原 英人[†1]
植木 美和[†1]　由良 淳一[†1]　武 理一郎[†1]

概要：複数の端末から同時に編集するリアルタイム共同編集システムは既にサービスとして提供されているが，従来の共同編集システムでは扱えるデータ数に限界がある．次世代の共同編集システムでは，連携する端末や機器の増加，また共有するデータの多様化と高品質化に伴いサーバの処理負荷が増加し，システムが破綻する恐れがある．本稿では次世代の共同編集システムに対し，サーバに処理を集中させない分散型構成で，通信状況やシステムの負荷に応じて通信量を抑制する，次世代共同編集システム向けの新しいアーキテクチャを提案する．筆者らが開発中の不要な通信を抑制する分散データ共有ミドルウェアを分散型の次世代共同編集システムに適用し，提案アーキテクチャの実現可能性を評価した．その結果，次世代の共同編集システムのサーバ処理負荷を軽減し，実環境でも遅延の少ない UX を提供できることを確認した．

キーワード：共同編集，分散システム，結果整合性，マルチキャスト通信

1. はじめに

クラウド技術や通信技術が発達し，ネットワーク上の様々な機器やシステムが連携・協調するサービスやシステムが実現可能になってきている．その中でも，ネットワーク上で複数人のユーザが協調して作業を行うことができる共同編集システムが提供されている．近年では Google Docs1のような, 複数人のユーザが異なる端末から同時に１つのドキュメントファイルを編集できるリアルタイム共同編集システムも提供されている．これらのサービスは，グループ会議の議事録などの小規模な単位での文書編集で利用されている．

一方，スマート端末や IoT 機器のような小型のデバイスも通信機能や計算機能力を持ち，様々なシステムの利用や連携が可能になってきており，通信されるデータ数は増加する [1]. 更に, ネットワーク上で共有するコンテンツの多様化と高品質化により，通信されるデータサイズも増加している[2]．リアルタイム共同編集システムは，端末間のデータの整合性と他端末の操作に対する即時応答性が求められる．しかし，スマート端末や IoT 機器との連携やコンテンツの高品質化によるデータ数とデータサイズの増加に伴いサーバの処理負荷が膨大に増加し，整合性と即時応答性を保てなくなる恐れがある．

そこで本稿では，サーバの処理負荷の増加に対処した新しい共同編集システムのアーキテクチャを提案する．提案アーキテクチャは，サーバに処理負荷を集中させない分散型の構成で，最終結果に影響しない情報の通知を抑制し通信量を適切に削減する．これによって，データ数とデータサイズ増加によるサーバへの処理負荷を低減させ，多様な環境で様々な情報を扱う次世代の共同編集システムを実現可能にする．

また，筆者らが開発中の通信状況やシステムの負荷に応じて通信量を抑制する分散データ共有ミドルウェア（DDSM と呼ぶ）[11]を共同編集システムに適用して提案アーキテクチャの実装を行なう．サーバ負荷制御の効果を計測するため評価用のシミュレータを開発し，提案アーキテクチャを評価する．また，提案アーキテクチャを適用した共同編集システムの負荷評価も行なう．どの程度通信量を削減させ, どの程度の通信遅延を抑えることができるか, また適用させた共同編集システムがどの程度実現可能性があるのかを検証する．

以下，２章では従来のリアルタイム共同編集システムの課題，３章で提案アーキテクチャと実装，４章で評価，５章で関連研究について述べ，６章で本論文をまとめる．

2. リアルタイム共同編集システムの課題

2.1 既存のリアルタイム共同編集サービス

複数のユーザがネットワーク上で連携し，リアルタイムに共同で作業が可能なグループウェアサービスを，多くの企業が提供している．Google Docs は，複数のユーザが同時に文書編集することが可能であり，Google Sheets2は同様に表計算が可能である．法人向けに提供されている Suite3のユーザ企業数は 300 万を超えており[3]，リアルタイム共同編集システムのニーズは高まっていると言える．その他にも，Etherpad4など，複数のサービスで同様のリアルタイム

†1 (株)富士通研究所 Fujitsu Laboratories Ltd.
a) kani.junya@jp.fujitsu.com

1 Google Docs, https://docs.google.com/
2 Google Sheets, https://spreadsheets.google.com/
3 G Suite, https://gsuite.google.co.jp/
4 Etherpad, http://etherpad.org/

ⓒ2017 Information Processing Society of Japan

共同編集機能が提供されている．しかしながら既存のサービスは，同時利用可能なユーザ数やデータの種類が限定されたものが多い．動的なコンテンツやセンサデバイスが利用可能で，ユーザ数やデータ数，データサイズの増加に対処した共同編集システムサービスはいまだ存在しない．

2.2 空間 UI

筆者らは，次世代の共同編集システムとして空間 UI システム [4,5]の研究開発を進めている．空間 UI システムは，壁や机などの広い共有空間での情報の表示や共有を可能にする．例えば，ワークショップに参加した人のスマート端末と会場にある表示機器が連携しスマート端末の画面を壁や机に大きく映すことができる．空間 UI システム上の操作は逐次スマート端末に伝えられ，スマート端末間の情報交換が直感的な操作で簡単に実現できる．

空間 UI システムは様々なコンテンツや Web アプリケーションを複数の端末上でシームレスに動作させるために，複数の端末間で連携する仮想的なウィンドウシステムとして動作する．文章や手書きのメモを残すノートアプリや，デジタルな付箋紙を複数端末から共同で編集する模造紙アプリなど，様々なアプリを空間 UI システム上で動作させることができる．そのため空間 UI システムでは，文書編集情報だけでなく，動画像や Web コンテンツ，機器のセンサデータなど様々なコンテンツ情報に加えて，ユーザ操作に基づくウィンドウの描画情報などコンテンツの周囲情報も共有する．このように次世代の共同編集システムは，多量で多様なデータを共有する必要がある．

2.3 課題

既存のリアルタイム共同編集サービスでは，扱えるユーザ数が限定されている．Google Docs の公式サポート[6]では同時に編集可能なユーザ数は上限 50 ユーザまでと示されている．また Dang らは，Google Docs のパフォーマンス評価を行った結果，1 秒間に 2 文字の入力頻度でも，10 ユーザ数の同時利用で遅延が発生しはじめ，38 以上のユーザ数の同時利用でサービス不能状態になったことを報告[7]している．

既存のリアルタイム共同編集サービスに比べ，次世代の共同編集システムではコンテンツの中身はもちろん，周囲情報を含めて共有することになる．データ数やデータサイズが増加する上に，サーバ側での処理が複雑化するため，より一層同時利用可能なノード数は減少する．よって，共同編集システムのサーバ処理負荷を軽減させる新しいアーキテクチャが必要である．

3. 提案アーキテクチャと実装

本稿では上記課題に対し，通信量の増加によるサーバへの処理負荷の増加に対処した新しい共同編集システムのアーキテクチャを提案する．

3.1 提案アーキテクチャ

次世代の共同編集システムでは，コンテンツの中身はもちろん，コンテンツの周囲情報やセンサデータを含めて共有するため，コンテンツの中身と周囲情報をそれぞれ適切に同期する必要がある．コンテンツの中身は，アプリやアプリ内の機能に応じて整合性への要求が異なる．共有しているアプリケーションのウィンドウサイズ, 座標情報など，コンテンツの中身に依存しない周囲情報は，最終的に同期されていれば操作途中のデータは必ずしも必要のないデータであり，サーバ側で一括して集中処理する必要はない．コンテンツの中身のうち整合性への要求が低くても良い情報やコンテンツの周囲情報に対して以下の特徴を持つ共同編集システムを提案する．

（1） 整合性の要求が低くても良い情報に対する処理をクライアント側のノードで分散的に処理する．これにより，サーバに集中していた処理負荷を分散させる．

（2） 整合性の要求が低くても良い情報の通信をシステムの負荷に応じて抑制する．これにより，メッセージ通信の増加によるサーバ処理負荷を低減する．

3.2 実装

提案アーキテクチャを，次世代の共同編集システムである空間 UI システムに適用して実装を行う．空間 UI システムは，ユーザ操作に基づく空間構成情報をノード間で共有する．このデータは，最終的に同期されていれば操作途中のデータは必ずしも必要ではない．よって，空間 UI システム上で共有される空間構成情報を対象にして提案アーキテクチャを適用する．また，データ配信制御には，DDSM を利用する．

3.2.1 空間 UI への実装

空間 UI への適用イメージを図 1 に示す．サーバ側ではコンテンツごとに用意されたアプリケーションサーバが存在する．クライアント側では，DDSM とウィンドウシステムを組み込んだ，画面描画を行うクライアントノードが存在する．ウィンドウシステムは移動操作などの処理を検出し，画面の再描画を行う．コンテンツごとの通信はアプリケーションサーバを通じて行われ，画面描画に関するコンテンツ周囲情報はノード間で DDSM を用いて適切に送信抑制しながら共有する．

ⓒ2017 Information Processing Society of Japan

Vol.2017-MBL-82 No.8
Vol.2017-UBI-53 No.8
2017/3/9

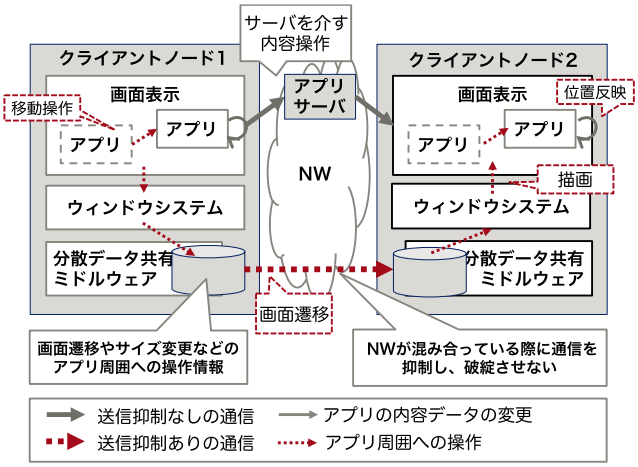

図 1　空間 UI への適用イメージ

3.2.2 分散データ共有ミドルウェア

筆者らが開発を進めている分散データ共有ミドルウェア（DDSM）[11]は，通信の頻発や瞬断など，不安定な通信環境や通信状態でも，不要な通信を適切に抑制しながら整合性を保ってデータを同期することができる．ノードごとの通信負荷や処理負荷の差を考慮し，それぞれの負荷を動的に判断する．システムの負荷全体を考慮しながらデータの送信を適切に遅延させ，システムの過負荷状態を回避する．今回適用した空間 UI システムでは，各クライアントノードが DDSM を利用し，描画情報などの空間構成情報を本 KVS の読み書きを行い描画処理する．これにより，空間構成情報は適切に送信抑制を行いながらノード間で同期される．

4. 評価

提案アーキテクチャのサーバ負荷に対する効果を検証するため，データ同期における遅延時間を測定する評価シミュレータを開発した．評価シミュレータを用いて，Google Docs と空間 UI システムに提案アーキテクチャを適用することを想定したシミュレーション評価を行った．また，提案アーキテクチャを適用した次世代共同編集システムの実現可能性を評価するため，空間 UI システムに実適用してサーバ負荷制御の評価を行った．

4.1 シミュレーション評価

4.1.1 評価シミュレータ

指定されたパラメータに応じて，複数ノードで相互にデータ書き込みとデータ配信を行い，動作結果を可視化する評価シミュレータを開発した．次世代共同編集システムが Web を基盤としたシステムとして発展することを想定し，評価シミュレータは，サーバ側で Web サーバとして起動する管理コンソール UI と，分散データ共有ミドルウェアを組み込んだツールクライアント（node.js[5]アプリ）が連携して動作するシステムとした．評価シミュレータの全体構成を図 2 に示す．

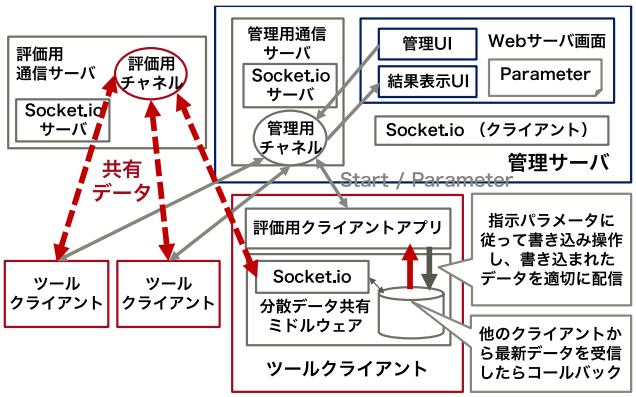

図 2　評価シミュレータの全体構成

通信には WebSocket 通信[8]の Socket.io[6]ライブラリを利用した．Socket.io サーバには，管理用チャネルと評価用チャネルを分けて用意することで，評価時の通信負荷がシステムに影響しないようにした．管理用チャネルは管理コンソールとツールクライアント間で，評価試行の開始指示や評価用のパラメータを送信するためのチャネルである．また，評価用チャネルはツールクライアント間で相互にデータ通信を行うためのチャネルである．評価シミュレータで利用可能なパラメータを表 1 に示す．

表 1　評価シミュレータで利用可能なパラメータ一覧

項目名	説明	単位
ノード数	データの相互書込を行うノード数	個
key 数	相互書込の対象となる key の数	個
データサイズ	key に対して読み書きするデータサイズ	Byte
書込間隔	データ書込処理を行う間隔．7	ミリ秒
書込 key 数	一回に書き込む key の数	個
処理時間	クライアントが書込処理を行い続ける時間	秒
送信抑制機能	DDSM の送信抑制機能の有無	有無

管理コンソールの UI 上でパラメータを指定し開始ボタンを押下すると，パラメータがツールクライアントに送信され，各クライアントで書き込み処理が並列に行われる．一定時間が過ぎた段階で，クライアントは動作を停止し，管理コンソールに自分が持つ動作結果を送信する．管理コンソールは各クライアントの動作結果から測定結果を算出し画面に表示する．評価シミュレータで得られる測定結果

5 Node.js https://nodejs.org/
6 Socket.io http://socket.io/
7 各ノードが定期的に同じタイミングで書き込み処理することを避けるため，指定された時間内のランダムな時間で書き込み処理を行う．

ⓒ2017 Information Processing Society of Japan

の項目をまとめたものを表 2 に示す.

表 2 シミュレータで得られる測定結果項目

項目	詳細
整合結果	全ノード間で全 key のデータが整合していたか
整合時間	全ノード間での整合までにかかった時間（処理終了後からの経過時間）
書込み数	全ノードの key データ書込み回数の累計
送信数	全ノードの key データ送信回数の累計
送信成功率	書き込みに対する送信要求回数（書込回数×ノード数）のうち，実際に送信された回数の割合. 書き込みを行う間隔が送信処理を行う間隔より短い場合や，書き込んだデータを送信する前に他のノードから新しいデータを受信した場合に間引かれる.
受信数	全ノードの key データ受信回数の累計
受信成功率	受信要求回数（書込ノードから受信ノードに対して送信が行われた回数）のうち，実際に受信された回数の割合. socket.io サーバに到達しないなど，通信層でエラーが発生した場合に割合が変化する.
コールバック回数	全ノードの key コールバック回数の累計
コールバック率	受信したデータのうちコールバックの発火が行われた回数の割合. 複数のノードから同じ key に対するデータを受信した場合や自分の書き込んだデータの方が新しかった場合に間引かれる.

コールバックは，共同編集システムが最新データの受信時にアプリが受信データに対して描画処理などの処理を行うことを想定しており，データを受信するたびにコールバック処理が発火する．ただし，データのタイムスタンプがすでに所持しているデータよりも過去のものであった場合，コールバック処理は発火しない.

また，送信抑制を行わない場合には，送信の割合，受信の割合，コールバックの割合は 100%となり，送信数，受信数，コールバック数は全て一致する結果になる.

4.1.2 シミュレーション評価環境

シミュレータを動作させた評価環境を表 3 に示す.

表 3 評価環境

種別	プラットフォーム	実行環境
サーバ	Windows 8(64bit) Intel® Core™ i5-4300U CPU 1.90GHz 2.50GHz 10.0GB RAM	Node.js v4.4.7
クライアント	macOS Sierra v10.12.3 Intel® Core™ i7 @ 2GHz 8.0GB RAM	Node.js v4.2.1

4.1.3 Google Docs のシミュレーション評価

はじめに，一般的な共同編集システムである Google Docs を対象に，評価シミュレータを利用して提案アーキテクチャのサーバ負荷制御の効果を評価した．シミュレータに指定するパラメータは，既存の一般的な共同編集システムである Google Docs の限界性能と同程度の負荷を想定し，Dang らの報告[7]を参考にして表 4 のようにした．なお，データサイズに関しては，Google Docs で採用されている OT アルゴリズム[12][17]を参考にする．OT アルゴリズムでは，{ InsertText: ‘A’ @1 }といった形式の操作種別，対象文字，変更箇所を示すデータをサーバに送信する．実装に応じて識別子や筆者情報なども送信するが，今回は最低限必要と思われる 10byte のサイズで評価を行なった.

表 4 Google Docs シミュレーション評価のパラメータ

項目	数値	想定
ノード数	10〜50	10 ノードごとにクライアントアプリを~50 まで増やして評価
key 数	1	1 つのドキュメントに対する同時編集を想定
データサイズ	10Byte	OT アルゴリズムの編集時のデータ量の想定
書込間隔	500ms	1 秒間に 2 文字程度の書き込み頻度を想定
書込 key 数	1	key 数と同様
処理時間	10 秒	20 文字程度の 1 つの文章を書き込むことを想定
送信抑制機能	有,無	送信抑制ありの場合, なしの場合それぞれで評価

上記パラメータの条件で 5 回ずつ試行し，それぞれの結果を平均値にして示す．データの内容に関しては，送信抑制ありの場合には全ての試行で最終的に整合が取れていることを確認している.

整合時間の測定結果を図 3 に示す．本グラフから，整合時間は送信抑制の有無に関わらず，10 ノード程度では数秒もかからないことがわかる．しかし送信抑制なしの場合，20 ノードで 5 秒，30 ノード以上では 20 秒を超えるか，もしくは通信が利用できない状態になった.

一方，送信抑制ありの場合では，ノード数が増えるごとに整合時間が緩やかに増加しており，50 ノードでは 4.5 秒程度の遅延が発生することがわかった．この結果は，従来や送信抑制なしの場合ではサービスが利用不能になってしまっていた状態に比べれば, 優位な結果になったと言える.

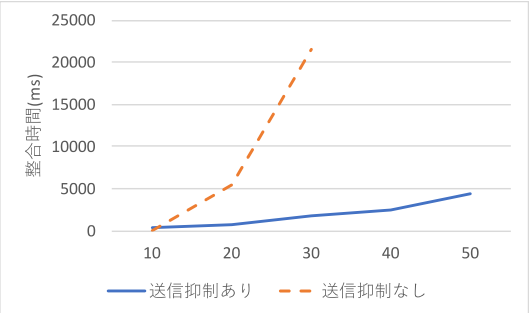

図 3 整合時間評価結果

次に，メッセージの送信数と送信成功率（送信抑制ありの場合）の測定結果を図 4 に示す．結果を見ると，送信抑制なしの場合の 30 ノードでは, メッセージ送信数は 30,000

回を超えており，理論的にも 30 ノードを超えれば 50,000 回を超えるメッセージ数が送受信されることになる．この数値はメッセージングサーバである Socket.io の限界スループット[9]である 9,000～10,000 を大きく超えてしまっているため，動作が不安定になってしまったものと考えられる．

送信抑制ありの場合では，送信するメッセージ数が増加に伴って送信抑制機能が働き，50 ノードでは送信数を 1 割程度に抑制していることがわかる．具体的には 50 ノードで，実際には 90,000 を超える送信要求のうち，その 10 分の 1 の 9,000 程度のメッセージ数に抑えることで，通信サーバへの負荷を抑えている．これによって，サーバが利用不能になることを回避している．

また，送信を抑制されたメッセージが妥当な制御であったのかを評価するため，コールバック回数とコールバック率を測定した．コールバック回数は，評価クライアントが DDSM に要求している key の更新通知要求に対し，他ノードから受信した更新メッセージのうち key への最新の更新のみを通知した回数である．つまり，受信したデータが全て最新のデータでアプリにとって有意義なものであれば，コールバック率は 100%となる．しかし 50 ノード時の結果では，コールバック率は 4.64%でメッセージ受信数に対して 1 割にも満たない非常に低い値であることがわかった．これは受信したデータがすでに所持しているデータよりも過去のタイムスタンプのデータであったために手元で破棄されていることを示し，不要なデータ送信を発生させていることになる．これに対しては，現時点の実装では通信負荷のみを考慮した送信抑制となっているため，受信ノードにとって必要かどうかも動的に推測することで対処可能であると検討しており，ロジックの設計と実装は今後の課題とする．

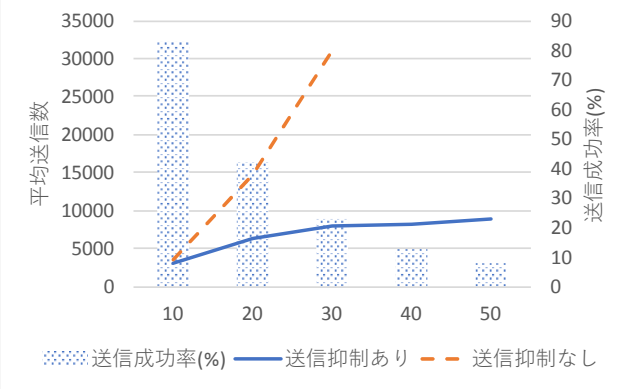

図 4　平均送信数の評価結果

4.1.4 空間 UI のシミュレーション評価

次に，次世代の共同編集システムとして空間 UI を対象に，評価シミュレータを利用して提案アーキテクチャが次世代の共同編集システムにおいてどの程度サーバ負荷制御に効果があるか評価を行った．シミュレータに指定するパラメータには，部屋全体を UI とした最大で 20 人規模程度のワークショップを行う環境を想定し，表 5 のようにした．

表 5　空間 UI シミュレーション評価のパラメータ

項目	数値	想定
ノード数	2～10	2 ノードごとにクライアントノードを 10 まで増やして評価．最大で前方左右に 2 面，机に 4 面の画面を想定．
key 数	50	1key が 1 つのウィンドウで，最大で 1 つの画面に 4 つ程度のウィンドウが表示されることを想定．
データサイズ	100Byte	空間 UI ウィンドウ管理用のコンテンツ周囲情報のおおよその値．
書込間隔	50ms	ブラウザのサンプリングレートを想定．
書込 key 数	2	1 つのディスプレイ端末に対し 2 人程度のユーザが同時操作を行うことを想定．
処理時間	10 秒	1 回の操作で 10 秒程度のウィンドウ操作を行うことを想定．
送信抑制機能	有,無	送信抑制ありの場合，なしの場合それぞれで評価．

上記パラメータの条件で 5 回ずつ試行し，それぞれの結果を平均値にして示す．データの内容に関しては，送信抑制ありの場合には全ての試行で最終的に整合が取れていることを確認している．

整合時間の評価結果を図 5 に示す．本グラフから，送信抑制の有無に関わらず 2～4 ノードであれば，1 秒以内の遅延で整合することがわかった．しかし送信抑制なしの場合には，6 ノードを超えると負荷が一気に増大し，18 秒程度の整合時間がかかることがわかった．8 ノード以上では負荷が高くなってしまい，通信サーバの動作が不安定になり最終データが整合せず正確な測定結果が得られなかった．一方，送信抑制ありの場合では，ノード数が増えるごとに平均整合時間が増加し，10 ノードでは 4.5 秒程度の遅延が発生することがわかった．この結果は Google Docs シミュレーション同様，送信抑制を行わない場合にサービスが利用不能になってしまっていた状態に比べれば優位な結果になったと言える．しかし，4.5 秒程度の遅延は近年の Web サービスにおけるコンテンツ表示の遅延時間としては，更なる改善が求められる結果となった．

ⓒ2017 Information Processing Society of Japan

Vol.2017-MBL-82 No.8
Vol.2017-UBI-53 No.8
2017/3/9

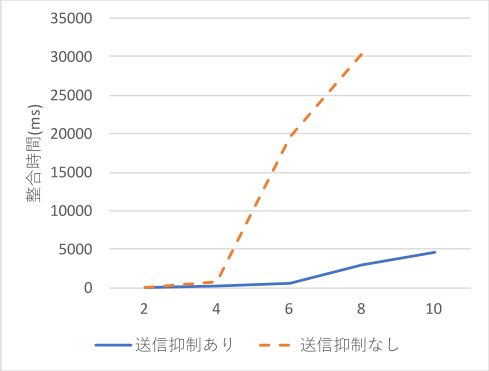

図 5　整合時間評価結果

次に，メッセージの送信数と送信成功率（送信抑制ありの場合）の測定結果を図 6 に示す．結果を見ると，送信抑制がなしの場合，8 ノードでは通信しているメッセージ数が，28,000 程度である．整合性が取れていなかったことから，この値以上のメッセージを通信しようとしていたことになる．8 ノード以上で通信サーバの動作が不安定になった理由は，メッセージ数が Socket.io の限界性能を大きく超える値であるため，遅延の発生や動作停止が生じたと推測できる．一方，送信抑制ありの場合では，4 ノードで既に 9,000 程度の送信メッセージ数であり，10 ノードでは 6 割の通信を抑制しているにも関わらず 25,000 に近いメッセージ数が送受信されていることがわかる．これは Socket.io の限界スループットを大幅に越える値であり，10 ノード程度であれば利用不能になることは回避しているものの，通信サーバには高い負荷がかかっており，より一層の通信抑制が必要であることがわかった．

また，送信を抑制されたメッセージが妥当な制御であったのかを評価するため，コールバック回数とコールバック率を測定した．Google Docs シミュレーション時と同様に，コールバック率は緩やかに下がり，10 ノード時には 28.5% となっており，送信抑制には更なる改善が求められる．

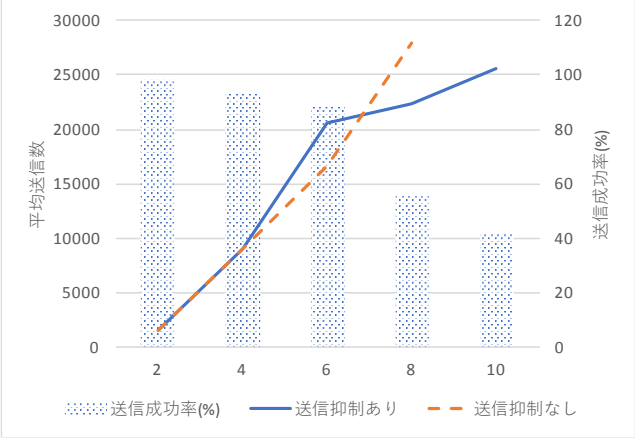

図 6　平均送信数の評価結果

4.2　空間 UI への適用評価

シミュレータによる評価結果から，空間 UI システムでは 6 ノードまでの相互利用であればほとんど遅延を感じることがなく，8 ノードを超えるあたりから遅延が発生することがわかった．それを実証するため，DDSM を実際の空間 UI システムに適用し，実システムでの評価を行った．

4.2.1 評価環境

評価環境を表 6 に示す．

表 6　評価環境

種別	プラットフォーム	実行環境
サーバ	Windows 8.1 Pro Intel® Core™ i7-4765T 2.00GHz 8.0GB RAM	Node.js v6.2.2
ディスプレイ端末	Windows 8.1 Pro Intel® Core™ i7-4785T 2.20GHz 8.0GB RAM	Chromium 50

4.2.2 評価概要

評価には，管理サーバ 1 台と，壁表示用 PC 2 台，机表示用 PC 2 台の合計 5 台で空間 UI システムを動作させた．壁表示用 2 台で壁一面，机表示用 2 台で 4 人掛けの机 2 台での利用が可能であり，おおよそ 10 人程度の規模のワークショップを想定している．管理サーバはウィンドウシステム上で動くアプリサーバやメッセージングサーバとして動作する．管理サーバ端末上に 2 ノードとディスプレイ用端末に 4 ノード，合計 6 ノードで動作する．管理サーバ上の 2 ノードはデータを書き込むことはないが変更通知は受信するため，その分の通信が発生する．送信抑制の有無のそれぞれの条件で評価を行った．

システムに一定の負荷を加えるため，3 つのディスプレイ用端末からウィンドウに対して定期的に操作する．ウィンドウシステム上には 10 のウィンドウを表示しておき，それぞれ 3 端末から 10 のウィンドウに対して常にウィンドウの描画変更が必要な操作を行う．

測定対象の評価は，上記操作を行っていない 1 つの端末上で，ウィンドウの作成とウィンドウの移動操作を行う．それらの操作が残りのノード上で反映するまでの遅延時間を測定する．これにより，空間 UI 上で一定の負荷がかかっている状態でどの程度の遅延が発生するかを測定する．

4.2.3 評価結果

評価結果には 3 回行った平均値を以下の表 7 に示す．

表 7　適用評価結果

	整合時間(ms)	送信数	送信率(%)
送信抑制なし	8290.33	10923	100
送信抑制あり	747.67	20280.25	26.86

表 7 の結果を見ると，整合時間は送信抑制のない場合には 8,290ms かかっていたが，送信抑制機能を利用することで 747ms 程度の遅延で整合することがわかる．文献[18]で

ⓒ2017 Information Processing Society of Japan

は，人間が感じる認知機能に基づけば，0.1 秒でソフトウェアが反応することが理想的であり，1 秒の遅延でユーザがシステムのインタラクションに何かしらの問題が発生したと考えると述べられている．また SmartBear の調査[19]によれば，57%のユーザは Web 表示の許容できる画面描画の待ち時間は 3 秒と報告されている．今回の結果では 1 秒以内で整合していることからも，この程度の負荷と環境であれば，送信抑制がない実適用環境に比べれば，ある程度十分な UX を提供できることが確認できたと考えられる．しかしながら理想的には，0.1 秒程度での同期が理想と言えるため，より一層の改善が求められる．

メッセージ送信数は，送信抑制なしの場合に比べて送信抑制ありの場合には 20,280 回と倍以上の差が出ていたが，送信率を見ると 7 割以上の通信が抑制されていることがわかる．その理由は，送信抑制がない場合では送受信が連続して発生することで描画処理が重なり，ディスプレイ端末に対して負荷がかかってしまったため，ブラウザへの入力操作のサンプリングに遅延が発生し，書き込み処理そのものが減ってしまったことによるものと推測できる．また，評価中のウィンドウシステム画面は，見た目にも滑らかであるとは言い難いものであったことからも，描画処理が減っていたと推測できる．

5. 関連研究

我々の提案アーキテクチャは，サーバ処理負荷を減らすことを目的に通信の抑制を行なっている．そのため，共同編集システムのような複数の端末でデータを同期するシステムに関する多数の研究の中でも，サーバ処理負荷や通信負荷に対処する研究について述べる．

5.1 サーバの処理負荷軽減に関する研究

Dang ら[7]は， Google Docs や Etherpad を例に挙げ，大規模なユーザ数での同時利用では動作が安定しないことを報告している．Dang らはそれに対して，Google Docs で利用されている編集整合方式である，OT アルゴリズム[12][17]を問題点に挙げ，より軽量な CRDT 方式[15]を利用すべきであると主張している．しかし，たとえ CRDT 方式を利用したとしてもデータ数が増加すればサーバの処理負荷も増加することに変わりはなく，いずれシステムは破綻する．我々の提案では，サーバの処理負荷を一定の負荷で推移するように，分散型の構成で通信を抑制する点が異なる．

また，我々の方式と同様にサーバへの集中処理を問題視し，スケーラビリティの問題に対処するために，分散システムや P2P ネットワークを共同編集システムに生かす研究が提案されている[13,14]．Duplex[13]は，共同編集システムを大規模環境で利用するために，ヘテロな通信環境を含む大規模なネットワーク環境で，分散型の共同編集システムを構築するためのアーキテクチャを提案している．分散型にすることによって生じる問題を解決することにアプローチしており，複雑な整合性解消の問題には，ドキュメントを独立的に分解して管理することで，整合させるパーツを分けて通信可能とし，不整合を避ける仕組みである．データを分割している点は似ているが，我々は整合性への要求に応じて分割することで適切なデータにのみ送信抑制を可能にしている点が異なる．

Oster ら[14]は，P2P 型の共同編集システムを提案し，P2P 型の共同編集システムでの整合性保証のアルゴリズムを検討している．本アーキテクチャの構成に近いが，ノードや通信量の増加に対する検討まではされていない．

5.2 通信負荷に対する送信抑制に関する研究

Perkins らが提案している Simba は[10]，クライアント・サーバ間でデータ連携を行うモバイル向けのアプリケーションにおいて，モバイルアプリ内で高い整合性が求められるデータと低い整合性で問題のないデータを予めアプリ開発者が分けて定義しておく．それにより，データ同期のための通信に優先度をつけ，モバイル環境のような不安定な通信環境においてデータ通信を効率化させるアプローチである．しかし，クラウドサーバを必要とした構成を前提にしており，整合性を保つことに焦点をあてているため，データ量がより一層増加していった際に，通信負荷に応じて積極的なデータの通信抑制を行わなければシステムが破綻する恐れがある．

Katayama ら[16]は，Web コンポーネントの Canvas を用いてデータを同期する協調型 Web アプリにおいて，同期する Canvas 上のデータに対する無駄な通信を減らすことで，同期遅延を減らす方式を提案している．既存の Canvas をラップした，更新頻度の高さの違う複数のレイヤーを持つ新しい Canvas を定義する．アプリは要素の更新頻度の高さに応じて描画するレイヤーを変える．これにより，再描画が必要な場合に，更新頻度の高いデータから順に通信することが可能になる．本方式も，通信を削減することに貢献できるものの，Web 上のコンポーネントである Canvas にしか適用できない．

6. まとめと今後の課題

従来の共同編集システムでは扱えるデータ数に限界がある．次世代の共同編集システムでは，連携する端末や機器の増加，また連携するデータの多様化と高品質化に伴い，通信が増加し，サーバの負荷が高まることでシステムが破綻する恐れがある．それに対して本稿では，次世代の共同編集システムに対し，サーバに処理を集中させない分散型構成で，通信状況やシステムの負荷に応じて通信量を抑制

ⓒ2017 Information Processing Society of Japan

する，新しいアーキテクチャを提案する．また，筆者らが開発中の不要な通信を抑制する DDSM を次世代の共同編集システムである空間 UI システムに適用し，評価を行った．評価結果から，既存の集中管理型の構成や送信抑制のない仕組みではデータ数の増加に耐えきれず，システムが動作停止となるが，提案アーキテクチャでは動作可能であることを確認した．また，実際に実適用したシステムで，送信抑制がない実適用環境に比べれば，ある程度十分な UX を確認することができた．

しかし，評価結果ではデータ数の増加に応じて処理負荷が依然として増加していた．またコールバック率が非常に低く，いまだ不要な通信を行っていた．今後は，受信ノードにとって必要かどうかも動的に推測する仕組みのロジック設計と実装を行う予定である．また共同編集システムで利用される様々なデータ種別にも対応させ，適用範囲を広げる．最後に今回の評価では，筆者らが開発している分散データ共有ミドルウェア，空間 UI システムを一例としたものでしかないため，そのほかの共同編集システムや通信抑制モジュールに対して適用させ，異なるパラメータ，環境での評価を行なっていく．

謝辞 富士通研究所で活発にご議論頂いた方々と，評価シミュレータの開発を支援して頂いた富士通ソフトウェアテクノロージズの伊藤様，神田様に感謝致します．

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]総務省: 我が国のインターネットにおけるトラヒックの集計・試算, http://www.soumu.go.jp/menu_news/s-news/01kiban04_02000107.html (参照 2017-02-01)
[3]Techcrunch: Google G Suite の有料ユーザー企業が 300 万社を超えた, http://jp.techcrunch.com/2017/01/27/20170126more-than-3m-businesses-now-pay-for-googles-g-suite/ (参照 2017-02-01)
[4]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)
[5]板倉，可児，由良，武: 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)
[6]Google: Google Docs 公式サポート, https://support.google.com/docs/answer/2494822?hl=en (参照 2017-02-01)
[7]Dang, Q., Ignat, C.: Performance of real-time collaborative editors at large scale: user perspective, Proc. IoP Workshop co-located with Networking, IFIP (2016)
[8]World Wide Web Consortium (W3C): The WebSocket API, https://www.w3.org/TR/2011/WD-websockets-20110929/ (参照 2017-02-01)
[9]Drew Harry: Practical socket.io Benchmarking, http://drewww.github.io/socket.io-benchmarking/ (参照 2017-02-01)
[10]Perkins, D., Agrawal, N., Ayanya, A., Yu, C., Go, Y., Madhyastha, H., Ungureanu. C.: Simba: Tunable End-to-End Data Consistency for Mobile Apps, Proc. EuroSys'15 ACM (2015)
[11]今井，可児，木原，由良，植木，武: 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，情報処理学会研究報告，，情報処理学会 (2017)
[12]Sun, C., Ellis, C.: Operational Transformation in Real-Time Group Editors: Issues, Algorithms and Achievements. Proc. Conference on Computer-Supported Cooperative Work (CSCW'98), pp. 59-68 ACM (1998)
[13]Pacull, F., Sandoz, A., Schiper, A.: Duplex: A Distributed Collaborative Editing Environment in Large Scale, Proc. Conference on Computer-Supported Cooperative Work (CSCW'94), ACM (1994)
[14]Oster, G., Urso, P., Molli, P., Imine, A.: Data Consistency for P2P Collaborative Editing, Conference on Computer-Supported Cooperative Work (CSCW'06), ACM (2006)
[15]Preguica, N., Marques, M.S., Shapiro, M. :, Letia, M. : A Commutative Replicated Data Type for Cooperative Editing. Proc. the 29th International Conference on Distributed Computing Systems (ICDCS), pp. 404-412 IEEE (2009)
[16]Katayama, S., Shiramatsu S., Ozono, T., Shintani, T.: Generational Layered Canvas Mechanism for Collaborative Web Applications, Proc. Advanced Applied Informatics (IIAIAAI), IEEE (2014)
[17]Google: What's different about the new Google Docs: Making collaboration fast, https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs.html (参照 2017-02-01)
[18]Jeff Johnson，武舎広幸，武舎るみ: UI デザインの心理学, pp 251-260，インプレス (2015)
[19]SmartBear: The Cost of Poor Web Performance, http://blog.smartbear.com/web-performance/the-cost-of-poor-web-performance-infographic/ (参照 2017-02-01)

ⓒ2017 Information Processing Society of Japan