RF-001

既存 Web ページ同期編集システム WFE-S における差分同期機構の試作
An Implementation of a Differential Synchronization Mechanism for Synchronous Web Editing System WFE-S

合田 拓史† 井上 良太† 加藤 雄大†
Takushi Goda Ryota Inoue Yudai Kato
白松 俊‡ 大囿 忠親‡ 新谷 虎松‡
Shun Shiramatsu Tadachika Ozono Toramatsu Shintani

1. はじめに

本研究室では，一般的に用いられている Web ブラウザを用いて，複数人で同期的に Web ページを編集するシステム WFE を開発している[1]．本システムを用いることで，既存の Web ページを Web ブラウザのみで編集可能になる．本研究では，WFE における通信効率やレスポンス速度といったパフォーマンスの向上，スケーラビリティの向上，および一貫性維持機構の改善を行うことを目的とし，WFE システム全体を WFE-S として再開発を行う．また，その開発・実装環境として，Google が提供している GoogleAppEngine サービスを用いて開発を行うこととした．近年では，様々な Web サービスにおいて差分同期の研究が盛んであり[2]，本研究でも Web ブラウザ上で HTML データの差分情報のみを交換することで効率的な同期を実現した．本論文では，差分情報のみを用いた同期型の Web ページ編集システム，およびその実装方法について述べる．また，評価実験によって，プッシュ配信機構や差分更新機構を用いる事で，HTML データの一貫性維持性能を改善できたことを示す．

2. Web ページ同期編集システム WFE の課題

本システムのベースとなった Web ページ編集システム WFE の基本的な構成や動作，問題点等について述べる．WFE とは，Web ブラウザ上に表示された Web ページの文字列を選択することにより，Web ページの HTML ファイルを編集することができるシステムである[3]．ユーザはシステムが提供するインターフェースを用いて Web ページの編集を行うことができる．本システムの利点としては次のようなものが挙げられる．まず Web ブラウザ上から編集を行えるため編集が手軽である，編集結果を確認する手間が不要である，サーバへのアップロード作業が不要といった事から編集におけるユーザの負担軽減が期待できる．また，編集のためのインターフェースをシステムが提供するため，HTML の存在を意識せずに編集を行える，すなわち HTML 言語に対する知識が必要ないという利点もある．

†名古屋工業大学 工学部 情報工学科
Department of Information Technology,
Nagoya Institute of Technology
‡名古屋工業大学 大学院 情報工学専攻
Major of Information Technology
Graduate of Nagoya Institute of Technology

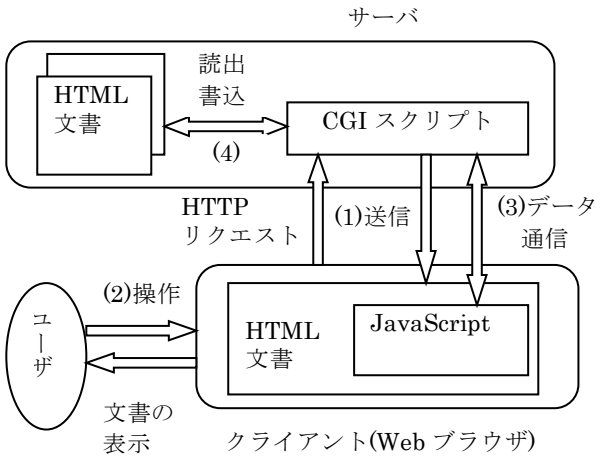

図 1. WFE システムの概観

2.1 WFE の構成および動作

WFE の基本的な構成について図１を用いて説明する．図１に示すように WFE は主に次の２つの要素から構成される．１つ目は編集可能な Web ページを提供し，HTML ファイルの管理等を行うサーバに設置された CGI スクリプトである．２つ目は実際に HTML ファイルに対する編集操作を行い，結果をサーバに渡す JavaScript である．システムはサーバ側に設置された CGI スクリプトと，編集対象となる HTML ファイルに埋め込まれた JavaScript 間で通信を行うことで実現される．また，そのために編集可能な Web ページを公開するサーバは，そのページにあらかじめ編集用の JavaScript を埋め込んでおく必要がある．しかし WFE にはそれを自動で行う機構が無く，ユーザの手間となっていた．これに関しては，システム全体の再開発の一環としてブックマークレットを用いた Web ページの登録機構が開発されており，ほぼ実装されている．WFE の基本的な動作について述べる．WFE を用いた Web ページの編集は，次の(1)～(5)の手順で行われる．

(1) クライアントの HTTP リクエストに対して編集のための JavaScript が埋め込まれた HTML ファイルを返す．
(2) クライアント側で，HTML ファイルに対して JavaScript が提供する UI を介して編集操作を行う．

(3) クライアントからサーバに対して，編集操作を行った結果（HTML ファイル）を送信する．
(4) サーバ側で(1)でクライアントに送信した HTML ファイルの内容を，(3)でクライアントから送信されたもので置き換える（上書きする）．
(5) (2)～(4)を繰り返す．

WFE は 1 つのファイルを複数人で編集するため，何らかの一貫性維持の仕組みが必要であるが，WFE では編集結果間で衝突が発生したときに，後の編集結果で上書きすることで一貫性を維持している．しかし，この方法での一貫性維持は，前の編集結果が消失するという問題にもなっている．また，誰かが編集した結果はできるだけ早く他のユーザに通知されないと，そのような編集結果の衝突が起きやすくなってしまうことが考えられる。これについては，編集操作を行っているクライアントがポーリングによる HTML ファイルの更新の問い合わせを数秒単位で行うことである程度解消している。

2.2 一貫性の保証

WFE は複数人で同時に 1 つの HTML ファイルの編集を行っている場合に，各ユーザから見た場合の最新の HTML ファイルに一時的に不整合が起きる可能性がある．具体的にどのような場面で不整合が発生するかを述べる．まず初期状態として，A と B という 2 つのクライアントが 1 つのファイルを編集可能な状態にある．その状態で A が先に編集操作を行い，サーバへ編集結果を送信したとする．WFE の想定する利用シナリオでは，その後，サーバが受け取った編集結果により格納されている HTML ファイルを更新，B のポーリング処理によりその編集処理は B の編集している HTML ファイルにも反映される．しかし，ここで B が A の編集をポーリングにより検知する前に編集操作を行いサーバへその結果を送信した場合について考えてみる．その場合，ここで送信した B の編集結果に A の編集内容は含まれていない．そのため，A の編集結果が先にサーバに届き HTML ファイルを更新した後，A の編集内容の反映されてない B の編集結果で上書きされてしまい，A が編集した内容は消失してしまう．これを図 2 に示す．このように実時間で先に行われた他のユーザによる編集を検知する前に，編集操作を送信する事を本システムでは編集操作の衝突，または単に衝突と呼び，衝突によって HTML データに不整合が発生する事を矛盾が発生するという．しかし，衝突が発生したとしても，B から見ると A による編集があったことを知らず，自身の編集が正しく反映されたように見えるだけであり，サーバから見ても A の編集の直後に B の編集が行われたというだけで B の編集が最新の HTML ファイルであるということで問題なく処理される．従って，A が B の編集結果に対して再び編集操作を行うことで，結果的には全ての立場から見て問題なく整合性が取れることになる．しかし，これでは A が同じ編集操作を 2 度行うことになり，ユーザにとって使いやすいシステムであるとは言い難い．

3. Web ページの同期編集のための差分同期機構

ここでは，既存の WFE に存在する問題点，つまり一貫性保証性能の改善を目的として行った実装について述べる．まず 2.2 で示した問題点について，改善手法の検討について述べる．この問題の解決には 2 つのアプローチが考えら

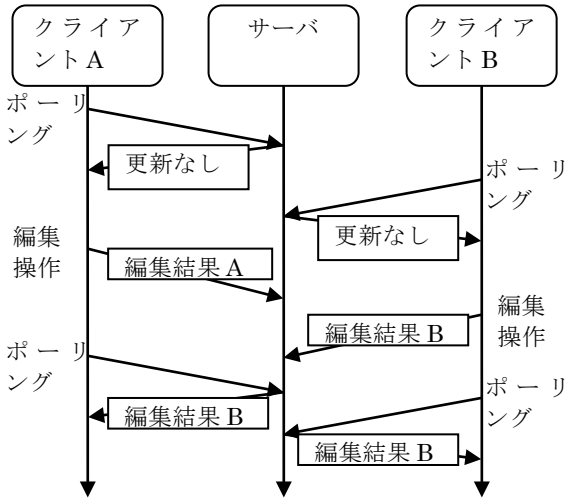

図 2. 不整合の発生ケース

れる．1 つはクライアントが常に最新の HTML データに対して編集操作を行えるようにすることであり，もう 1 つは最新でない HTML データに対して編集操作が行われた場合でも上手く整合性を保つ仕組みを用意することである．前者は衝突自体を発生させ難くすることであり，後者は衝突による矛盾を発生させ難くすることである．本システムでは衝突が発生した際の解決手段の一つとして，送受信する HTML データの内容を改良することで，一貫性維持性能を向上させることを目指す．ここではクライアントが編集を行った際に編集結果として送っていた HTML データ全体を，変更を行った箇所の HTML データ内での位置，編集後データとすることで実装した．常に最新の HTML データに対して編集操作を行うためのアプローチとしてはプッシュ配信による HTML ファイルの更新通知が考えられるが，これについては GoogleAppEngine への移植の際に行ったこととして 4.2 で述べる．

一貫性の保証のための具体的な手法として今回行ったのは，差分更新による編集結果の矛盾回避である．1 つの HTML ファイルを複数人で編集したとしても，編集部分が重なっていなければ両方の更新を適用することは可能である．本システムでは HTML の DOM 要素単位での更新を行うことで一貫性保証性能の向上を目指す．差分情報のみを用いてクライアント間で Web ページをリアルタイムで同期するための課題は，(1) Web ページ上での編集の差分を取得する方法，(2) 差分を管理する方法，および，(3) 差分を適用する方法，の 3 点である．以降，これらの課題の解決方法について論じる．

3.1 差分情報の取得

本システムは Web ページの編集操作を行う際に，まず HTML のタグ情報から編集部分を含む DOM 要素を検出する．そして，検出した DOM 要素の HTML を書き換えることで，実際に HTML ファイルに対する編集を行っている．今回実装する差分更新では，編集を行う際に対象となる DOM 要素までの XPath を差分情報として用いている．XPath とは XML に準拠した文書の特定の部分を指定する言語構文である[4]．この XPath により WFE での編集部分を指定することで，他クライアントは編集部分を知

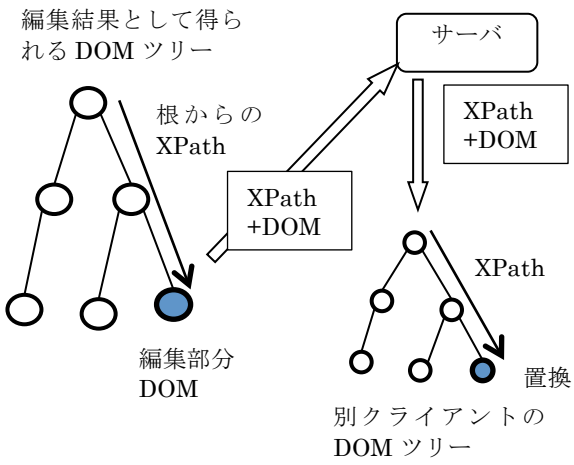

図 3. XPath による差分更新イメージ

ることができる．この XPath とそれが指す DOM 要素の HTML を差分情報として，これまで編集結果として送信していた HTML 全体の代わりにサーバに送信する．XPath を用いた差分更新のイメージを図 3 に示す．

3.2 差分情報の管理

クライアントからサーバへ編集結果として送信された差分データは，サーバに用意した差分情報格納用のデータストアに蓄積される．この差分データはクライアントからのアクセスがあった際，必要に応じて適宜送信される．また，この方法では差分データが編集の度に増えていくため，編集を重ねると処理効率は悪くなり，この差分更新機構はいずれ破綻する．従って適当なタイミングで，ある時点までの差分データを全て適用した HTML データをスナップショットとして作成し，HTML データ自体を更新する必要がある．

3.3 差分情報の適用

差分情報の適用には２つの場合が考えられる．１つは新規のクライアントから Web ページへのアクセスがあった場合，もう１つは既に Web ページへアクセス中のクライアントから編集操作があった場合である．まず前者の場合について述べる．新規のクライアントから Web ページへのアクセスがあった場合には，Web ページの HTML データが読み出されるが，このとき読み出される HTML データにはそれまで行われた編集操作が一切適用されていない．そのため，Web ページに対して行われた編集操作の差分データをまとめて読み出し，クライアントへ送信する．古いものから順に全ての差分データを適用すれば最新の HTML データが得られる．また，この際得られた最新の HTML データをスナップショットとしてサーバに送信する．次に編集操作が行われた際の差分情報の適用について述べる．複数のクライアントが同一の Web ページにアクセスしている際に，あるクライアントが編集操作を行った場合は，その編集操作を他のクライアントの HTML データにも適用する必要がある．この場合には全ての差分情報は必要無く，その時行われた編集による差分情報があれば十分である．そのため，編集操作が行われた場合は差分データストアへ差分データの保存は行うが，新たに読み出しは行わず，クライアントから送信されたものをそのまま各クライアントへ送信する．それぞれの場合での HTML データ，差分データの流れを図 4，図 5 に示す．

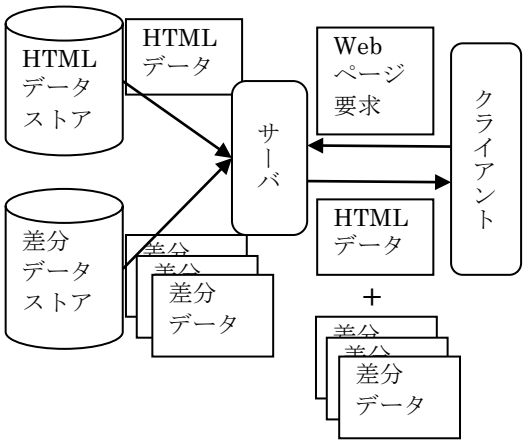

図 4. 差分情報を用いた Web ページアクセス

4. 差分情報プッシュ機能の実装

GoogleAppEngine を用いた WFE（以下 WFE-S と呼ぶ）の開発について述べる．ここでは，WFE-S の基本的な構成，動作，また既存の WFE（以下，旧 WFE）を GoogleAppEngine 上に移植する際に必要となった変更点について述べる．WFE-S と旧 WFE の違いについて概要を述べる．両者ではサーバ側の環境が変わるだけであるため，基本的な構成要素も大きく変化するわけではない．しかし，自前のサーバではなく Google の提供するものを用いるため，サーバの CGI スクリプトから静的ファイルの作成，変更，削除といった操作を行うことができないということや，CPU リソースの利用時間制限等，いくつかの制限がある．また， WFE-S では，旧 WFE で他のクライアントの編集操作を検知するために行っていたポーリング処理の代わりに，GoogleAppEngine が提供する API を用いたプッシュ配信機構を新たに実装し，他のクライアントによる編集操作の検知に利用している．

4.1 HTML データの保管について

旧 WFE は，Web サーバに設置された Perl スクリプト内でサーバのファイルシステム内に存在する HTML ファイルに対して読み出し，書き込み，またその生成を行うことで Web ページの管理を行っている．しかし，GoogleAppEngine では静的なファイルに対する操作を行うことが出来ないため，新しく HTML データや差分データを管理する機構が必要となった。今回実装する WFE では，GoogleAppEngine で提供されている機能の一つであるデータストアを用いて Web ページの保管や更新操作の管理を実装することとした．データストアは GoogleAppEngine がアプリケーションに提供するストレージであり，インターフェースとして SQL とよく似た GQL という問い合わせ言語を備えている．これを用いた HTML データの管理機構の概観を図 6 に示す．

ここで，編集対象 Web ページの登録について述べる．Web ページの登録には JavaScript のブックマークレットを用いており，登録の際の手続きは次のようになる．まず

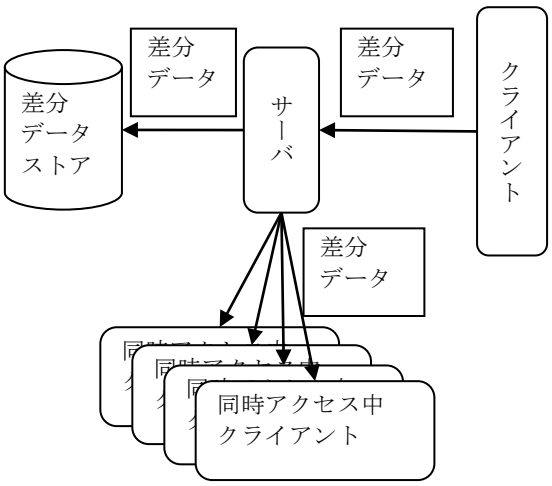

図 5. 差分情報を用いた編集検知時の処理

ユーザは編集したい Web ページをブラウザで開き，ブックマークレットを実行する．ブックマークレットは WFE-S の URL にアクセスし，登録用のスクリプトを読み込む．登録用スクリプトはそのページの URL をキーとして，ページが WFE-S のデータストアに存在するかを問い合わせる．サーバは対応する HTML データがデータストアに存在すればサーバに格納されている HTML データの ID を返し，無ければ存在しないという旨の応答を返す．クライアントは，HTML データの ID を受け取ればそれを用いて HTML データへアクセスし，HTML データが存在しないという応答を受け取れば HTML データの送信を行う．

HTML データへのアクセスにおいて，クライアントが HTML データの ID を取得し，具体的な HTML データの送信要求を行った際には，サーバは以下のような手順で応答する．まず，要求された URL のパス名末尾のファイル名に当たる部分の拡張子から HTML ファイルや画像ファイルなどのファイルタイプを判別し，それに応じてデータストアから該当するデータを読み出す．結果として得られたデータにコンテンツタイプを示すヘッダを付加し，クライアントに送信する．

HTML データの編集操作に関して，クライアント側での処理は既存のものと異なり，差分データをクライアント側に渡し，クライアント側で編集を行うことは先に述べた．これは，その際に述べたスナップショットをサーバに送信する際の処理についてである．編集結果の HTML データを受け取ったサーバ側での処理は次のようになる．

(1) 編集を行った Web ページの ID と，編集結果の HTML データを受け取る．
(2) 元の HTML データには無かった，JavaScript での編集中に付加された編集のための DOM 要素を，HTML データとして保存する前に取り除く．
(3) データストア内の HTML データを編集された結果で置き換える．

以上により，旧 WFE で行っていたクライアントでの編集操作を，サーバに存在する HTML データに適用するまでの処理を WFE-S として実装した．

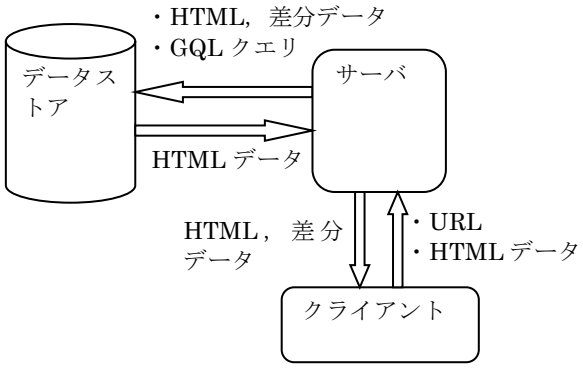

図 6. データストアを用いた HTML データの保管

4.2 プッシュ配信機構について

WFE-S では，GoogleAppEngine が提供する API の一つ，ChannelAPI を用いて Comet によるプッシュ配信機構を実装した．ChannelAPI はアプリケーションと Google サーバ間で持続的なコネクションを作成し，クライアントの JavaScript に対してメッセージを送信することで，ポーリングに依らないリアルタイム通信を実現することができる[5]．プッシュ配信を実装するにあたり必要となる処理を次に示す．

(1) プッシュ配信を行うための，サーバでのソケット開設処理．
(2) サーバからクライアントにプッシュ配信を実際に行う処理．
(3) クライアントでサーバからのプッシュ配信を受信した際の処理．

(1)をクライアントからサーバへの Web ページのリクエストの際の追加処理として，(2)をサーバがクライアントから編集結果を受信しデータストアに編集結果を格納した後の処理として，そして(3)をソケットが何かしらのメッセージを受信した際の処理として，それぞれ実装した．

5. Web 同期編集システムの実装

ここではシステムの機能として直接の関係はないが，システムとして必要な Web ページ編集者管理機能の実装や，既存の WFE に存在していた問題点の解消等も行った． 旧 WFE に対して，別ページへのリンクの埋め込み，ページのタイトル，背景色等の編集機能の拡張，またログインによる編集制限等の機能実装を行ったバージョン（以下拡張版 WFE）がある．この拡張版 WFE を基にして編集機能拡張，編集者管理機能等の実装を WFE-S に対して行った．以下で，その作業中に必要となった変更点などについて述べる．旧 WFE から拡張版 WFE で実装されたのは主に次の３つである．

(1) 各種編集機能の追加
(2) パスワードによる編集制限機構
(3) スクリプトの読み込み手続きの変更

これらの変更の内，(1)については JavaScript 内で閉じた変更であり，編集を行っていた JavaScript を拡張版 WFE のものに置き換えることでほぼ実装することが可能であったため，(2)，(3)について述べる．また 2.1 節で触れた，ブックマークレットを用いた Web ページ登録機構についても簡単に述べる．

5.1 パスワードによる編集制限機構

拡張版 WFE では，Web ページにパスワードを設定し，Web ページを編集できるユーザを制限する機能が追加されている．この機能は，次のような手続きで行われる．

(1) クライアントが編集する Web ページへアクセスする．
(2) サーバからクライアントへ，ログイン用の JavaScript を埋め込んだ HTML データを送信する．
(3) 編集操作を行おうとした際に，ログインフォームを表示，ユーザにパスワードを要求する．
(4) ユーザがパスワードを入力する．
(5) クライアントからサーバへ，編集対象の Web ページの URL，(4)で入力されたパスワードを送信する．
(6) サーバでパスワードの照合を行う．
(6.1)成功すれば，編集用の JavaScript を送信する．
(6.2)失敗すれば，失敗したと応答し，(3)に戻る．

ここで，(6)で認証が成功すれば，サーバは編集用の JavaScript をクライアントへ送信し，それを受け取ったクライアントは編集操作を行えるようになる．また編集後にサーバへ編集結果を送信する際に，パスワードも POST パラメータとして付加し送信する．(6)で認証が失敗すれば認証に失敗したという応答を返し，再びログイン要求があるまで待機する．

ログイン機能を WFE-S に適用する際に必要になったものは，サーバ側での認証処理と Web ページに設定するパスワードの管理機構である．パスワードは，HTML データのパラメータの一つとしてデータストアに実装した．

5.2 スクリプト読み込み手続きの変更

旧 WFE では，必要な JavaScript ファイルを全て HTML から直接読み込んでいたが，拡張版 WFE では利用する JavaScript ファイルの内，あるファイルを最初に一つだけ読み込み，そこから残りの JavaScript ファイル連鎖的に読み込むように変更されている．理由は，WFE で利用している JavaScript ファイル間で依存関係があり，読み込む順番を制御する必要があるためである．Web ブラウザは HTML の解析中に Script タグが存在した場合，HTML の解析を一時中断し，Script の読み込み，実行が終了するまで待機するものが多いが，そうなっていないものもある．そのような場合に，JavaScript の読み込まれるタイミングがずれ，途中で未定義の関数を呼び出そうとした等の理由でエラーとなってしまう可能性がある．読み込み手続きの変更は，これに対応したものである．

ここでの WFE-S への適用処理は次の通りである．まず JavaScript 側の変更点である．これに関してはほぼそのまま使うことができ，変更点は HTML リクエストを行う先の URL と，更新確認のためのポーリング処理の停止である．WFE-S ではプッシュ配信によるリアルタイム通信を実装するため，クライアント側からポーリング処理を行う必要が無くなったためである．二つ目がサーバ側での編集用 JavaScript の送信処理である．拡張版 WFE では，パスワード認証の際にサーバの CGI 内で編集用 JavaScript のロード用プログラムを生成し，クライアントへ送信，クライアントがそれを実行して編集用 JavaScript をロードしていた．しかし，編集用 JavaScript のロードプログラムの取得にサーバを介する必要が無いのではないかということから，次のような仕様変更を行った．まず，JavaScript のロード用プログラムを生成していた Perl スクリプトから，編集用 JavaScript のロード用 JavaScript ファイルを作成する．それを最初の HTTP リクエストで得たログイン用 JavaScript からロードし，そこから編集用 JavaScript のロードスクリプトを適宜実行することでシステムに必要な JavaScript を全てロードする．

5.3 ブックマークレットによる Web ページ登録機構

ブックマークレットを用いた Web ページ登録機構について簡単に述べる．本システムを用いて Web ページを編集する際，Web ページに編集用の JavaScript を埋め込む必要がある．本研究と平行して，これを容易に行うための手段として Web ページのシステムへの登録，JavaScript の埋め込みをブックマークレットにより自動化する機構が開発されている[6]．動作の流れは次のようになる．

(1) 編集用 Web ページの存在確認
(2) パスワードの登録
(3) 編集ページの生成
(4) 編集ページへの遷移

ユーザがブックマークレットを呼び出すと，閲覧中の Web ページが本システムに登録されているかを確認する．登録されていなければ，ユーザに編集用パスワードの設定要求を行う．ユーザがパスワードを入力すると，ブックマークレットは Web ページの URL，パスワード等をシステムの動作しているサーバへ送信する．サーバではそれを用いて編集用 JavaScript を埋め込んだ Web ページを生成し，編集用ページのアドレスをクライアントに送信する．そしてクライアント側ではそのアドレスのページに遷移する，というものである．最初の編集用 Web ページの存在確認の段階で閲覧中のページが既に登録されていた場合は，パスワードの入力要求を行い，ユーザの入力したパスワードをサーバ上のデータと照合し，一致すれば編集ページへの遷移を行う．

6. 評価実験

最後に，新たに実装した WFE-S の評価実験を行った．評価する性能はシステムのリアルタイム性と 2.2 で示したような衝突，矛盾の回避性能である．

6.1 リアルタイム性評価実験

まず，リアルタイム性評価のために行った実験について述べる．本実験ではリアルタイム性を次のように定めた．「複数人が同一 Web ページに対して編集を行っている際に，あるクライアントの行った編集が，別のクライアントのブラウザに表示されている HTML データに反映されるまでの時間が短いほどリアルタイム性が高い」である．

リアルタイム性評価のために行った実験の環境について述べる．まず同時編集の評価基準であるが，1 クライアントは大体 30 秒に 1 回の編集操作を行うとする．つまり 30 秒に 1 回の編集が行われれば 1 人による同時編集が行われており，15 秒に 1 回の編集が行われれば 2 人による同時編集，2 秒に 1 回で 15 人といった具合である．本システムの利用対象としては，研究室程度の小規模グループによる利用を想定しており，30 人での同時編集に耐えられる程度，すなわち 1 秒に 1 回の編集操作がサーバに送られる状況でも，一定時間以内にレスポンスを返すことを目標とする．また目標となるレスポンス時間であるが，1000 ミ

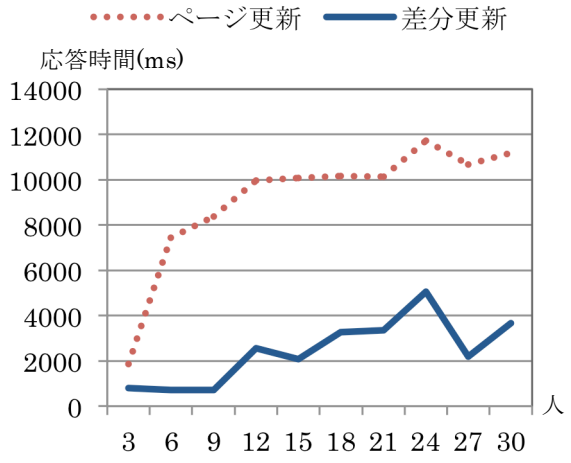

図 7. 同時接続数の変化に伴う応答時間の変化

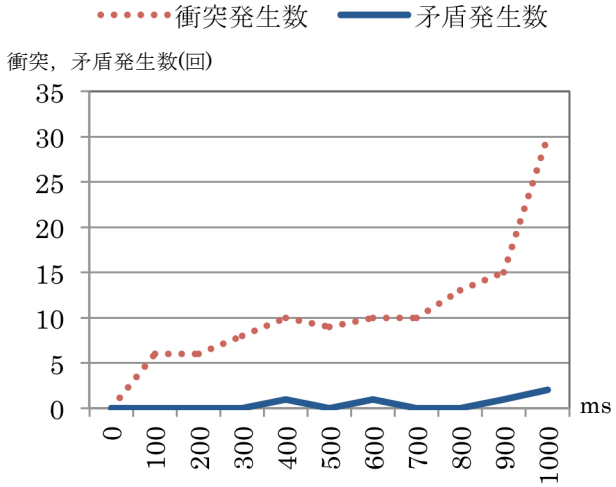

図 8. 遅延時間の変化に伴う衝突，矛盾発生数の変化

リ秒以内に応答が帰ってくれば複数人で同期編集を行うための十分なリアルタイム性があるとした．また，システム改良による性能の変化を調べるため，ページ単位での更新を行うバージョンでも同様の実験を行い，結果を差分更新版でのものと比較する．

実験は次のような方法で行った．まずクライアント数に応じたアクセス間隔を基準とした，ある程度ランダムな時間間隔で HTML データのランダムな場所に対して編集操作を行う．ここで，サーバに編集操作を送信する直前の時間を開始時間として記録する．その後サーバは編集操作を受信すると各クライアントに対してプッシュ操作を行い，それを受信したクライアントが編集を適用する．ここで編集を適用した直後の時間を終了時間として記録する．このとき，終了時間から開始時間を引いたものをレスポンスにかかった時間として記録する．これを数十回繰り返してレスポンスにかかった時間の平均を平均応答時間として記録した．

今回の実験の結果を図 7 に示す．結果を見ると，本システムでは同時接続数が 9 までなら目標の 1000 ミリ秒を達成できているが，それ以降は目標を達成できていない．また 9 と 12 の間で応答時間が急激に増加しているが，それ以降はそれほど大きく変化していない．単純な増加となっていないのは，同時人数が増えても編集操作の間隔がある程度ランダムになっていることから，編集タイミングの偏りが生じたためと考えられる．従来の WFE のようにページ更新を行うものと比較すると，レスポンスにかかる時間を大きく抑えられているのがわかる．

実験の手法について考察する．今回の実験は，編集間隔が一定ではシステムへ複数人による同時編集時のリアルタイム性を正しく評価できないのではないかということから，編集操作の間隔にある程度のばらつきを持たせた．しかし，結果を見ると人数が増加したときの応答時間の変化が予測しにくいものとなった．編集操作間隔のばらつきはもう少し抑えた方が良いと考えられる．

実験結果について，今回は 1 クライアントが 30 秒に 1 回の編集を続けて行い，小規模なグループの全員で一斉に編集を行うという条件での評価実験を行った．目標の 1000 ミリ秒を達成できたのは 10 人程度までであったが，ページ単位での更新を行う場合と比較するとリアルタイム性は大きく向上させることができた． 実験を行う前は，同時接続数がある程度増えると，レスポンス時間が爆発的に増加することを予想したが，どちらも同時接続数に比例して大きくなるわけではなかったことについても調査する必要がある．

6.2 矛盾回避性能評価実験環境

次に矛盾回避性能の評価のために行った実験について述べる．具体的な実験方法，結果を述べ，実験についての考察を行う．

この衝突回避性能評価実験では，理想的には実時間でのタイムラグ無しで同時接続中クライアント間での編集情報の交換を行う事ができれば，2.2 で述べたような矛盾は発生しない，という事に基づいて行う．実験では，あるクライアントの編集操作を他クライアントにプッシュ配信させる際にサーバ側で意図的に遅延を生じさせる．そして遅延を変化させた際の衝突，矛盾の発生数を記録する．それにより，本システムにおける衝突，矛盾の発生がどの程度問題となるかを検証する．

まず同時接続人数を 5 人と仮定する．ユーザの同時接続の基準についてはリアルタイム性評価実験と同じものを用いる．そしてリアルタイム性の評価実験と同じように，ある程度ランダムな時間間隔でランダムな場所に対して編集操作を行う．この際，各編集操作には操作をサーバに送信した時間と編集操作識別情報を付加する．そして，各クライアントが別クライアントからの編集操作をサーバからのプッシュ配信により受信した際に，その情報を記録する．このランダム編集操作を各クライアントで 10 回ずつ，同時に開始し，そのログを取る．あるクライアントが行った編集操作 A を，同クライアントが受け取った任意の別クライアントからの編集操作 B と比較を行う．A の送信時間が B の送信時間より後，かつ B の受信時間より先であった場合，編集操作の衝突が起きていることになる．これにより衝突の検知を行う．次に矛盾の検知であるが，矛盾が発生するのは衝突した編集操作間で編集領域が重複した場合である．これは一方の編集領域，すなわち DOM 要素がもう一方の DOM 要素を内包する場合である．これについては衝突した編集操作の XPath を比較し，一方がもう一方の部分文字列になっているかどうかを調べる事で検出可能である．

本実験の結果を図 8 に示す．衝突発生数，矛盾発生数は各クライアントで発生したそれぞれの回数の合計を示す．従来のページ単位での更新では，本実験における衝突がほぼ全て矛盾となっていたことを考えると，差分更新によって衝突後の矛盾が発生する確率を大きく低減させる事ができたと言える．また，サーバ側からの編集通知に遅延を持たせると，衝突の発生確率が増大する事も確認できる．すなわち編集操作を素早く検知する事ができれば衝突の発生確率を減少させる事ができ，本システムで実装したプッシュ配信機構によって衝突を減少させる事ができたと言える．従来のポーリングによる手法では 3 秒毎にポーリングによる編集検知を行っていたため，平均 1.5 秒の遅延が生じていた事になる．本実験では 1.0 秒の遅延までしか測定していないが，それでも，衝突数を大きく減少させることは確認できる．本実験の結果から，サーバからの応答遅延の増大により衝突の発生回数が増加しても，それによる矛盾はほとんど増加しておらず，矛盾の発生は実用上無視できる程度のものであると考える．

7. 考察

本システムでは，Web ページの同期編集において一貫性を保持するための仕組みとして，ロック機能および編集領域の可視化機能を備えている．ロック機能では，ユーザによる編集対象領域のロックが可能である．これにより，ロックを取得したユーザは，他のユーザに対して排他的な編集が可能であるため，一貫性が保証される．ロックの問題点としては，編集の前にロックを取得する作業が手間であることや，ロックの解除忘れが課題となり，運用の上では煩雑である．本システムでは，リアルタイムな同期編集を円滑にするために，一貫性の保持よりも運用上での煩雑さを減らす事を重要視し，ロックしない同期編集も可能とした．

編集の可視化機能とは，あるユーザが編集を開始したことを他のユーザに通知する機能である．具体的には，あるユーザが編集を開始したときに，その編集に依る衝突の可能性がある範囲を他のユーザに可視化する機能である．また，衝突が発生した場合には，衝突した領域を表示する事も可能である．可視化のためには，追加のプッシュ配信が必要となり，レスポンスが悪化する．レスポンスの悪化により，衝突が増える可能性があるため，経験的には可視化しない方がよいと考えているが，より詳細な検証が必要である．

8. おわりに

本研究では，既存の Web ページを複数人で同期編集するためのシステム WFE-S を実装し，同期通信の効率を改善するために部分的な DOM 木および XPath を用いた差分更新の仕組みを実現した．実験により，プッシュ配信や差分更新を用いる事で，実用上では衝突による問題はほとんど発生しない事が確認できた．仮に派生したとしても，ユーザ間の協調で容易に解決可能なものであると考える．本手法により，通信負荷を軽減できたことから，スケーラビリティの向上にも期待できる．本システムにより,Web サーバの運用形態に関わらず，既存ページを複数人で同期編集することが可能になり，既存 Web ページを用いた協調作業を支援することが可能になる．

現状ではスタイルシートや JavaScript を利用する Web ページには対応しておらず，今後の課題として残る．しかし，スタイルシートについては Web ページをシステムに登録する際に HTML を検査し，CSS の利用を検知，別途読み込み，CSS 指定の書き換え機能を編集用 JavaScript に実装することで比較的容易に実装できると考えている．スクリプトを用いた Web ページに関して，本システムを実装するためには具体的な実装手法を研究する必要がある．しかし，スクリプトを利用し Web ページに加える変更はユーザに適用させるような一時的なものである事が多く，スクリプトによる編集を共有編集ページとして同期することの意義は薄いと考えており，スクリプトを用いた Web ページへの対応は検討していない．

参考文献

[1] 土井 達也, 白松 俊, 大囿 忠親, 新谷 虎松,“ブックマークレットを用いた既存 Web ページのリアルタイム編集機構”, 情報処理学会全国大会, (2012).
[2] Neil Fraser,”Differential synchronization”, DocEng ‘09,(2009)
[3] 田代慎治，大囿忠親，伊藤孝行，新谷虎松，“WFE における協調的な Web ページの編集について”，第 67 回情報処理学会全国大会論文集, Vol.2, pp.419-420 (2005).
[4] “XMLPathLanguage(XPath)”,<http://www.w3.org/TR/1999/REC-xpath-19991116/>(2012/06/29)
[5] Google,“ChannelAPIOverview(Python)”,<https://developers.google.com/appengine/docs/python/channel/overview>(2012/04/16)
[6] 井上良太，加藤雄大，合田拓史，白松俊，大囿忠親，新谷虎松，”既存 Web ページ同期編集機構 WFE-S のクラウド環境への適用について”，FIT2012，(2012)（掲載予定）