RO-004

サーバサイドイメージレンダリングによる Web UI コンポーネントフレームワークの負荷分散手法
Workload Distribution for Web User Interface Component Framework using Server Side Image Rendering

石倉 直弥† 内海 宏律† 齋藤 邦夫† 手塚 大†
Naoya Ishikura Hironori Utsumi Kunio Saito Masaru Tezuka

1. はじめに

近年，モバイルファーストやクラウドファーストといった考えが提唱され，スマートデバイスやクラウド基盤の業務活用に対するニーズが高まっている．こうした背景から，既存の Web アプリのモバイル対応を容易に実現する Web UI コンポーネントフレームワーク(Cloud UI Component : CUIC)を提案した[1].

モバイル対応の問題の一つに端末の性能の低さがある．従来の Web アプリは，高負荷なレンダリングをクライアントで実行する．しかし，PC と比較し低性能なスマートデバイスではレンダリングを含む処理の応答に時間がかかる．そこで，CUIC はレンダリングをサーバ上で実行し，その結果を画像として転送する方式を採用することで，スマートデバイスであっても PC と同程度の応答性能を実現した．サーバサイドレンダリングは，高負荷処理(レンダリング)がサーバに集中する．そのため，クライアントの接続数やリクエスト数によっては，高性能なサーバであっても負荷に耐え切れない可能性がある．CUIC では，多数のクライアントが集中アクセスした際の負荷対策が課題である．

Web サーバの負荷対策に広く用いられる手法として，スケールアウトによる負荷分散がある．スケールアウトは Web サーバを複数台用意し，ロードバランサを利用して Web サーバ間の負荷が均等になるように処理を振り分ける．しかし，CUIC の通信アーキテクチャ上，本方法によるスケールアウトでは有効に負荷分散できない．

本研究の目的は，CUIC を適用した Web アプリケーション(CUIC アプリ)について，アクセスが集中する場合であっても十分な応答性能を確保することである．研究にあたり，上記の問題を解決し CUIC でスケールアウトを実現する手法を考案した．また，考案した手法を利用して，負荷が集中した場合であっても遅延を発生させることなく CUIC アプリが応答可能か評価を行った．

2. CUIC フレームワークの特徴と課題

2.1. CUIC フレームワークの特徴

CUIC は，既存 Web アプリ製品を容易にスマートデバイスに対応させることを目的とした汎用的な Web UI コンポーネントフレームワークである．以下に CUIC の特徴を示す．

(1) サーバサイドレンダリングによるクライアントの負荷軽減

モバイル対応時に発生する問題の一つにスマートデバイスの性能の低さがある．一般的な(MVC モデルの)Web アプリは，レンダラがクライアントに存在し，画面の更新が必要になるとクライアント上でレンダリングを行う場合が多い．レンダリングは高負荷な処理であり，低性能なスマートデバイスでは，実用に耐える応答性能を確保できない．

†（株）日立ソリューションズ東日本, Hitachi Solutions East Japan, Ltd.

そこで，CUIC ではサーバサイドレンダリングによってこの問題を解決している．本方式は，従来クライアントで行っていたレンダリングをサーバ上で実行する．レンダリング結果は画像として保存され，クライアントに送信される．クライアントは送られてきた画像を表示するのみとなり，スマートデバイスの性能でも多くの計算量を必要とする画面が表示可能となる．

図 1 および図 2 に，従来の Web アプリと CUIC アプリの画面表示フローを示す．図中のデータ取得とは，レンダリング用のデータを他のアプリや RDB から取得する処理を指す．サーバサイドレンダリングにより解決する問題を踏まえると，CUIC の適用範囲は以下のような特徴を持つコンポーネントと言える[1].

- **描画に多くの計算量や大量のデータを必要とする**
- **対象クライアントの性能が低い(応答性能が悪い)**

一方，テキストボックスやリスト表示コンポーネントのような描画負荷が小さいコンポーネントや，高性能なクライアントのみを対象とする場合には従来のクライアントで描画を行うモデルが適用できる．通常，Web 画面は複数のコンポーネントから構成される．そこで，先行研究では，計算量の多いコンポーネントは CUIC で描画し，それ以外の部分には従来のコンポーネントを用いる形態を提案した．

(2) WebSocket による通信の高速化

画像の送受信は，画像をブロック単位に分割しクライアント画面の表示に必要な分だけを送信する(差分転送)．そのため，CUIC のクライアントとサーバ間の通信は，通常の Web アプリと比較し二つの特徴が存在する．一つは，頻繁に通信が発生する点である．もう一つは，スクロールや拡

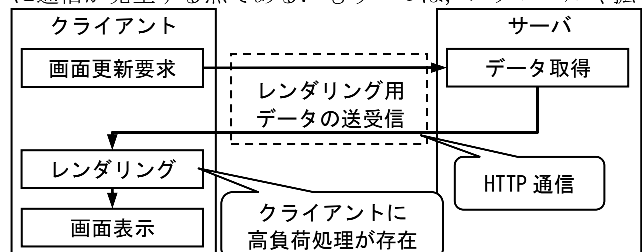


図 1 MVC モデルの画面表示フロー

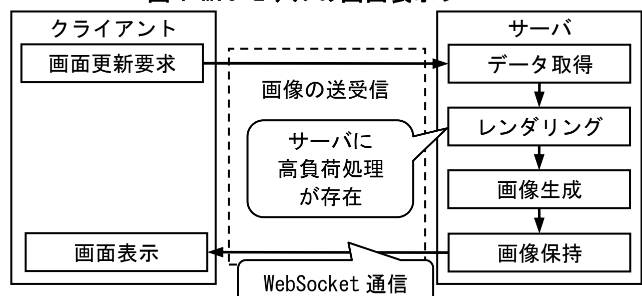


図 2 CUIC の画面表示フロー

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

大縮小(UI 操作)といった応答速度を求められる部分で通信が発生することである．これらの特徴を考慮すると，CUIC の通信はできる限り高速であることが求められる．

従来，クライアントとサーバ間の通信には HTTP がよく利用される(図 1)．しかし，通信のたびに接続を確立しなければならない HTTP は CUIC の通信プロトコルには不向きである．そこで，CUIC は WebSocket[2]を利用して通信を行う(図 2)．WebSocket は HTTP の拡張プロトコルであり，一度接続を確立すると，以降ユーザが切断するまでコネクションを維持する．CUIC では初期接続時にクライアントとサーバ間の通信経路を確保し，画像の送受信ではその経路を利用することで，画像表示の遅延を最小限にしている．

2.2. CUIC サーバの負荷問題

従来の Web アプリ(MVC モデル)では，レンダリングを各クライアントで行っていた．しかし，CUIC アプリはサーバに接続した全クライアントのレンダリングをサーバで行う必要があり，負荷がサーバに集中しやすい設計となっている．接続クライアント数が増加しレンダリングの頻度が高くなると，高性能なサーバであってもその負荷に耐え切れなくなる可能性がある．CUIC アプリがデプロイされたサーバ(CUIC サーバ)に多数のクライアントが集中アクセスする場合の負荷対策が CUIC の課題である．

3. CUIC スケーリングシステム

Web サーバの負荷対策としていくつかの方法があるが，コストや拡張性を考慮するとスケールアウトによって対策することが多い[3]．スケールアウトはクラウドとの親和性が高い[4][5]といった点でも有利である．以上を考慮し，本負荷対策でも，スケールアウトを前提として検討する．

CUIC サーバのスケールアウトについて，一般的なスケールアウトをそのまま適用すると負荷が均一に分散されないといった状況が発生する．そのため，上記問題を解決し CUIC でも適用可能なスケールアウト手法を考案した．以下，詳細について述べる．

3.1. 一般的なスケールアウトと CUIC への適用

一般的なスケールアウトによる負荷対策の例を図 3 に示す．本図では負荷対策が必要な Web アプリを二台(複数台)のサーバにデプロイする．クライアントは，ロードバランサ(LB)を経由してサーバにアクセスする．LB はクライアントから接続要求があると，各サーバの負荷が均一になるように要求を振り分ける．こうした負荷分散の例は，ステートレス接続を前提とした Web アプリでは有効である[6]．

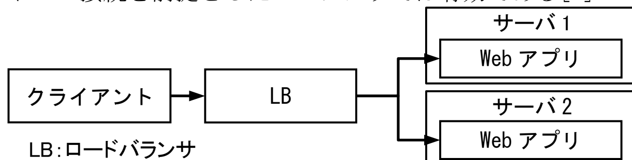


図 3 スケールアウトによる負荷対策の例

一方で，図 3 の構成を CUIC サーバに適用すると，サーバ間の負荷が偏る場合がある．その原因は，CUIC が通信に WebSocket を利用するためである．WebSocket は一度コネクションを確立するとその状態を維持する．つまり，初期接続時に通信するサーバが決定すると，以降の通信で別のサーバを使うことができない．そのため，レンダリングを伴う操作を，高い頻度で行うユーザ(高負荷ユーザ)と低い頻度で行うユーザ(低負荷ユーザ)が混在するような状況では，特定のユーザ(サーバ)のみ応答性能が悪化するといった現象が起こり得る(図 4)．

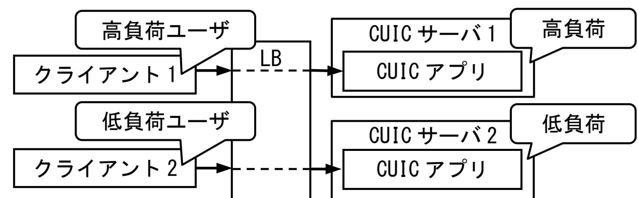


図 4 CUIC のスケールアウトの例

3.2. スケーリングシステムの考案

3.1 で示したように，CUIC が WebSocket を利用する関係上，一般的なスケールアウトによる CUIC サーバの負荷対策は困難である．そこで，CUIC にも適用可能なスケールアウト手法(スケーリングシステム)を考案した．スケーリングシステムの考案にあたり，画像更新の一連の処理のうち，スケールアウトが必要な処理はレンダリングのみという点に着目した．レンダリングを CUIC サーバから分離してしまえば，CUIC サーバ自体のスケールアウトは必要なくなる．

上記を踏まえ設計したスケーリングシステムの概要を図 5 に示す．スケーリングシステムは，高負荷処理であるレンダリングを専用のサーバ(レンダラサーバ)に分離し，レンダラサーバをスケールアウトする．CUIC サーバとレンダラサーバの通信にはスケーリングシステムが提供する独自の Web API(HTTP)を用いる．CUIC サーバに残る処理は低負荷処理のみであるためクライアントが複数台接続してきた場合でも，CUIC サーバの負荷が問題とはならない．

CUIC サーバのスケールアウトが不要となるため，クライアントと CUIC サーバ間の通信は従来通り WebSocket が利用できる．また，CUIC サーバとレンダラサーバ間の通信は HTTP であり，画面更新のたびに接続するレンダラサーバが変わる．つまり，UI 操作に伴う画像の送受信は高速なまま，画面更新が発生した場合のみスケールアウトによる負荷分散が可能となる．なお，スケーリングシステムによる画面更新は HTTP の介在により WebSocket のような高速な応答はできない．しかし，画面更新自体が UI 操作ほどの応答性能が求められないため，これによる弊害は生じない．

レンダラサーバの分離について注意すべき点がある．CUIC は多くの Web アプリに対して汎用的に適用可能なフレームワークである．CUIC サーバとレンダラサーバの通信は Web API を使って行うが，この Web API が特定(CUIC 以外)の Web アプリに依存するような設計は避けなければならない．そこで，CUIC は Web アプリによらず全てのレンダリング結果が画像になる点に着目し，この間の通信を画像生成要求とそのレスポンス(画像)とした．そのため，レンダラサーバではレンダリングの他に画像生成も行っている．

以上をまとめると，提案手法の要点は下記 3 点である．

(1) 高負荷処理(レンダリング)の負荷分散が可能
(2) CUIC の特徴である UI 操作の高い応答性能は従来通り
(3) 特定のアプリに依存しない汎用的な設計

4. スケーリングシステム評価

スケーリングシステムの評価では，スケーリングシステム自体の応答性能評価と既存 Web アプリへの適用評価および従来手法との比較評価を行った．以下，各評価の詳細とその結果について述べる．

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

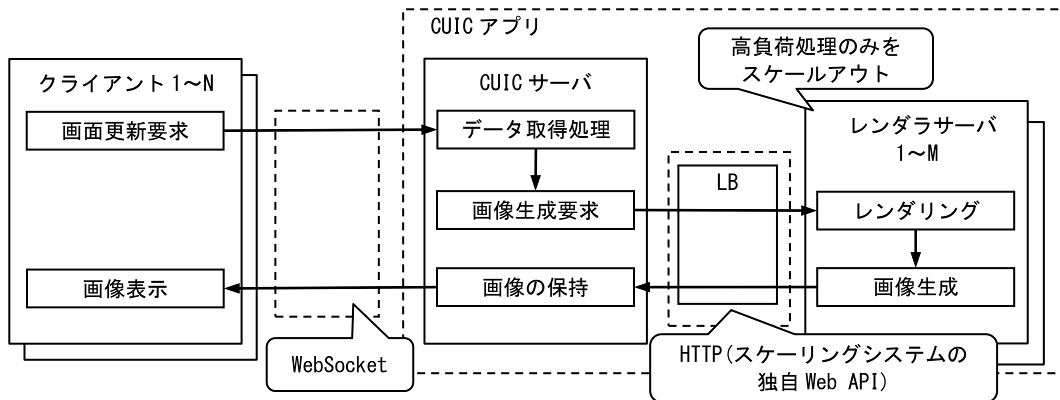


図 5 スケーリングシステムの構成

4.1. スケーリングシステムの応答性能評価

本評価では，考案したスケーリングシステムによって負荷問題が解決可能であるか評価する．評価では，CUIC アプリの動作をシミュレーションするテスト環境を構築し，負荷が上昇した場合でも適切な応答性能が確保可能か確認する．適切な応答性能が確保できている状態とは，負荷による遅延なしで処理できるリクエスト数(許容リクエスト数)が理論値と実測値で同様の推移を示すこととする．

評価手順として，初めにテスト環境の構築のため CUIC アプリのモデル化を行った．続いて，構築(モデル化)したテスト環境に基づいて許容リクエスト数の理論値を算出する．最後に，テスト環境を用いて許容リクエスト数の実測値を計測し理論値との比較を行った．

4.1.1. CUIC アプリのモデル化

CUIC アプリの動作をシミュレーションするテスト環境構築の前段階として，一般的な CUIC アプリのモデル化を行う．ここで一般的な CUIC アプリの動作とは，画像更新の際，データストアからデータを取得した後，そのデータを用いてレンダリングするといったものである．

図 6 に本評価で想定する典型的な CUIC アプリの例を示す．なお，図中では便宜上スケーリングシステム適用時と非適用時の二つのパスを同時に示したが，一度の画面更新要求で呼ばれるパスはどちらか一方である．CUIC アプリ動作のシミュレーションは，本モデルのボトルネックとなりえる処理について行う．該当する処理として，今回スケールアウトの対象となっているレンダリング(CUIC サーバ，レンダラサーバ)の他，データ取得応答(既存 Web サーバ)が考えられる．データ取得応答は，単にデータを返すだけならば低負荷な処理である．ただし，Web アプリによってはデータ取得の都度データ生成が必要となるものがある．データ生成は処理負荷が高いことが多いため，このような Web アプリではデータ取得応答がボトルネックとなることもある．その他の処理については，負荷が非常に小さくボトルネックとなる可能性は少ない．

以上より，負荷の高い処理としてデータ取得応答およびレンダリングがある(図 6)．両処理を高負荷処理 1 および 2 として定義する．

高負荷処理 1： データ取得応答
高負荷処理 2： レンダリング

図 7 は各高負荷処理に関するモデル図である．テスト環境はこのモデル図を元に構築した．テスト環境での各高負荷処理は，高負荷な状態を再現するために，CPU を占有する計算処理を指定した時間だけ実行する．ただし，ここで指定する時間は他の処理による負荷がかかっていないと仮定した状態での時間である．つまり，複数の高負荷処理が同時に実行されている状態では，指定時間以上に時間がかかることもある．評価では，クライアントから一定の頻度でリクエストを送信し，各サーバの高負荷処理について実際の処理時間を計測する．レンダラサーバにある高負荷処理 2 は，必要に応じてスケールアウトされるものとする．

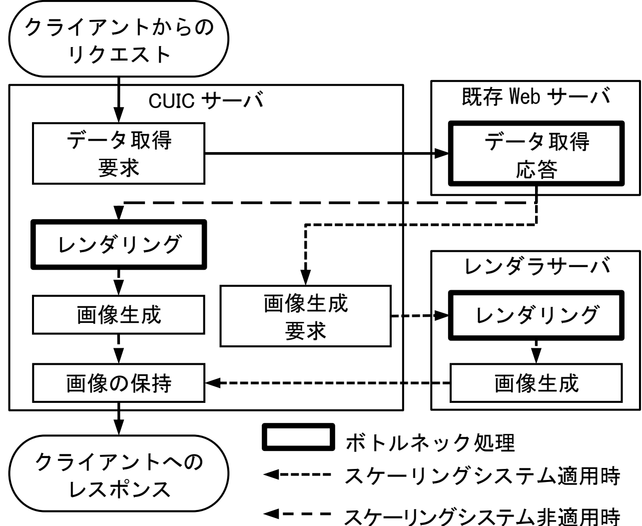


図 6 典型的な CUIC アプリの画像更新フロー

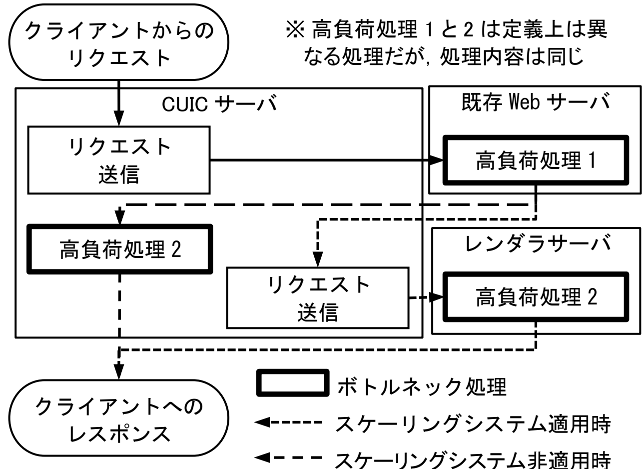


図 7 評価用モデル図

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

4.1.2. 許容リクエスト数理論値

許容リクエストの理論値の算出には待ち行列を用いる．高負荷処理は，先の処理(リクエスト)が終わるまで後続の処理が待たされることから，待ち行列の系と捉えることができる．図 7 より CUIC は，高負荷処理 1 と 2 からなる 2 段直列型待ち行列である[7][8]．直列型の待ち行列では，各系のうち一つでも遅延が発生すると，それがシステム全体の遅延に繋がる．そのため，CUIC アプリの許容リクエスト数は，高負荷処理 1 と 2 のうち許容リクエスト数の少ない方となる．CUIC アプリの許容リクエスト数λを式(1)に示す．ここで，λ_xは高負荷処理xの許容リクエスト数である．

$$\lambda = \min(\lambda_x | x \in \{1,2\}) \cdots (1)$$

高負荷処理の待ち行列について，各処理は m/m/s の待ち行列に従うものとする．許容リクエスト数を超えた状態とは，つまり，リクエストの平均到着率が各高負荷処理の平均サービス率を上回っている状態である．したがって，λ_xは式(2)を超えない最大値である．ここで，s_xとμ_xはxの窓口数と窓口一つあたりの平均サービス率である．また，s_xは高負荷処理xを実行するサーバ数n_xと各サーバのコア数Cの積となる．

$$\lambda_x < s_x \mu_x,\ s_x = C n_x |\ x \in \{1,2\} \cdots (2)$$

4.1.3. 実験設定

評価は，スケーリングシステム非適用時と適用時に分けて行う．スケーリングシステム非適用時の評価では，高負荷処理 1 および 2 の処理時間を変化させることでボトルネックとなる処理が変化することを確認する．適用時の評価では，高負荷処理 2 が常にボトルネックとなるように処理時間を調整し，レンダラサーバを増加させることで許容リクエスト数が増加することを確認する．

以上の条件で構築した評価環境に対し，クライアントから一分間あたりのリクエスト数を変化させて送信する．ここで，リクエスト数の変化量は 10 とする．リクエスト数が許容内であれば処理時間は全てのリクエストで一定であり，そうでなければ処理時間は徐々にあるいはバースト的に増加するはずである[9]．処理時間が一定となる最大のリクエスト数をその条件下の許容リクエスト数とする．

4.1.4. 評価環境

本評価は AWS[10]上の仮想環境で行った．評価に使用した仮想マシン(EC2)の性能について，表 1 に示す．本評価で用いるサーバの性能は全て表 1 で統一している．また，クライアントから送信するリクエストは JMeter[11]を利用して生成する．

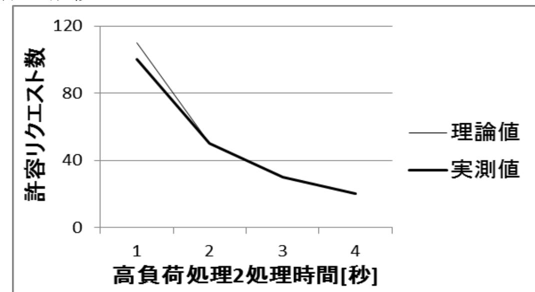
表 1 EC2 の実行環境

共通	リージョン		ap-northeast-1b (東京リージョン)
EC2	インスタンスタイプ		m3.large
		Virtual CPU	2.4 GHz Intel Xeon® E5-2676 v3 (Haswell)
		コア数	2
		メモリ(GB)	7.5
	OS		Windows Server 2012 R2
	Web サーバ		Internet Information Service 8.5

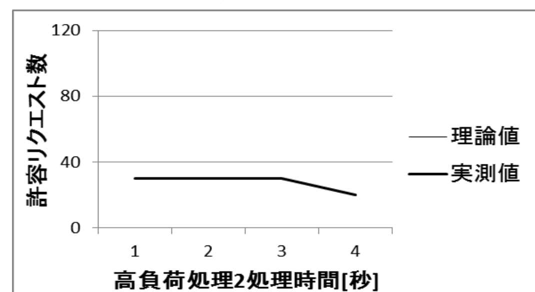
4.1.5. 評価結果

スケーリングシステム非適用時の評価結果について図 8 に示す．初めに，高負荷処理 2 が常にボトルネックとなる状態で，高負荷処理 2 の処理時間の変化が許容リクエスト数にどのように影響するか確認した．図 8(a)は高負荷処理 1 を 0.1 秒(非ボトルネック)に固定し高負荷処理 2 を 1 秒から 4 秒まで 1 秒間隔で変化させた際の許容リクエスト数である．ここで，横軸は高負荷処理 2 の処理時間，縦軸は一分間あたりの許容リクエスト数を表す．この場合，高負荷処理 2 が常にボトルネックとなるため，理論値が示すように高負荷処理 2 の処理時間増加にあわせて許容リクエスト数も少なくなる．実測値も理論値と同様の推移を示した．

続いて，あるタイミングまでは高負荷処理 1 がボトルネックとなる状態での許容リクエスト数の変化を確認する．図 8(b)は高負荷処理 1 を 3 秒に固定し，その他の条件については(a)に準拠させた場合である．この場合，高負荷処理 2 が 3 秒になるまでは，高負荷処理 1 がボトルネックとなるため許容リクエスト数は変わらない．ただし，4 秒になると，ボトルネックが高負荷処理 2 となり許容リクエスト数が少なくなる．こちらについても，理論値と実測値で同様の推移をしている．



(a)高負荷処理 1：0.1 秒

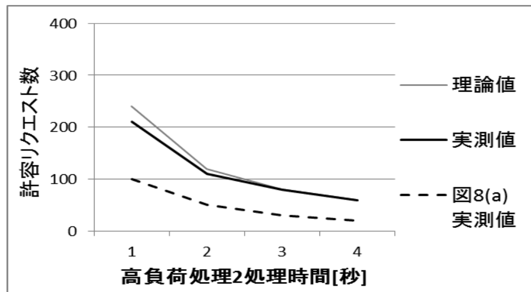


(b)高負荷処理 1：3 秒

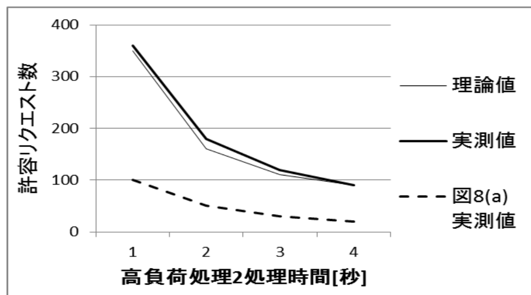
図 8 評価結果(スケーリングシステム非適用)

スケーリングシステム適用時の評価結果を図 9 に示す．図 9 (a)は図 8 (a)の条件に対し，レンダラサーバを 2 台にスケールアウトした際の評価結果である．評価の結果，理論値と実測値が同様の推移を示した．また，レンダラサーバ 1 台の場合と比較すると，許容リクエスト数が概ね 2 倍となっていることが分かる．図 9(b)はレンダラサーバの数を 3 台とした場合の結果である．こちらについても，理論値と実測値で同様の推移かつ，レンダラサーバ 1 台の場合と比較し許容リクエスト数が 3 倍以上になる．リクエスト数が等倍で増加しない理由は，リクエスト数の変化量を 10 としたためである．例えば，図 8(a)で高負荷処理 2 の処理時間が 4 秒の場合の許容リクエスト数は 20 であるが，より正確には 20 以上 30 未満である．この誤差の分がレンダラサーバ増加によって大きくなっていると考えられる．

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.



(a) レンダラサーバ数 2 台，高負荷処理 1：0.1 秒



(b) レンダラサーバ数 3 台．高負荷処理 1：0.1 秒

図 9 評価結果(スケーリングシステム適用)

4.2. 既存 Web アプリへの適用評価

4.1 の評価結果より，スケーリングシステムを活用することで，CUIC サーバの応答性能確保が可能となることが分かった．しかし，上記の評価では，実際の CUIC アプリでどの程度のリクエストが処理可能であるかが不明である．本評価では，工程管理などの現場で実際に利用されているプロジェクト管理用 Web アプリ製品への CUIC の適用を題材としてスケーリングシステムの応答性能を評価した．

4.2.1. 既存 Web アプリ概要

モバイル対応の対象となるプロジェクト管理用 Web アプリ製品(対象アプリ)の特徴として，次の 3 点がある．

(1) ガントチャートの閲覧が可能
(2) モバイル端末には未対応
(3) クライアントとサーバ間の通信は独自の Web API を利用

対象アプリは，プロジェクトの計画や進捗状況をガントチャートとして表示する機能を持っている．ただし，クライアントには PC を前提としており，モバイル端末には未対応である．また，対象アプリは独自の Web API を持って(公開して)おり，クライアントとサーバ間のデータの送受信は全てこの Web API を経由して行われる．

本評価では，CUIC を利用して上記のガントチャートをモバイル端末で閲覧するガントチャートアプリ(モバイルガント)を開発し(図 10)，スケーリングシステムによって許容リクエスト数がどのように変化するか確認する．

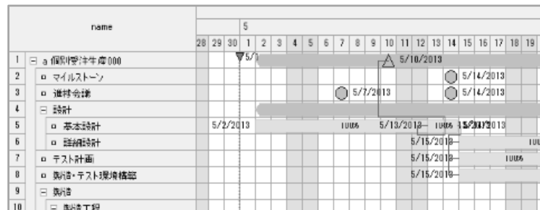


図 10 ガントチャートの画面例

4.2.2. モバイルガントの画像更新フロー

モバイルガントで画面更新が実行される操作は初期表示時とデータ更新時の 2 種類があり，それぞれ実行パスの一部が異なっている．データ更新としては進捗状況の入力や各作業項目を表す行(サマリタスク)の開閉処理などがある．データ更新が実行されるとそれに合わせて画面を再描画する必要がある．また，図 6 で示した典型的な CUIC アプリとの違いとして，モバイルガントはレンダラサーバで生成した画像を RDB に保存する．これは，CUIC サーバのメモリ上に全ての画像を保持する場合，CUIC サーバのメモリが枯渇する可能性があるためである．図 11 に画面更新の実行パスを示す．

初期表示の実行パスでは，クライアントから画面表示要求があると対象アプリから Web API 経由でデータ取得を行う．対象アプリはリクエスト時点での最新の進捗状況を返す仕様でありデータ取得応答のたびにデータ生成処理が実行される．そのため，この部分は高負荷処理となる．データ取得後は，取得したデータを RDB に保存(キャッシュ)しておく．データを保存した後は，レンダラサーバに処理が移る．レンダラサーバでは RDB に保存されたデータを取得しレンダリングを行う．レンダリングの際，レンダラサーバには対象アプリが持つガントチャートレンダラと同様のレンダラがデプロイされており，レンダリングの際はそれを用いる．その後，レンダリング結果から画像を生成し，RDB に保存する．

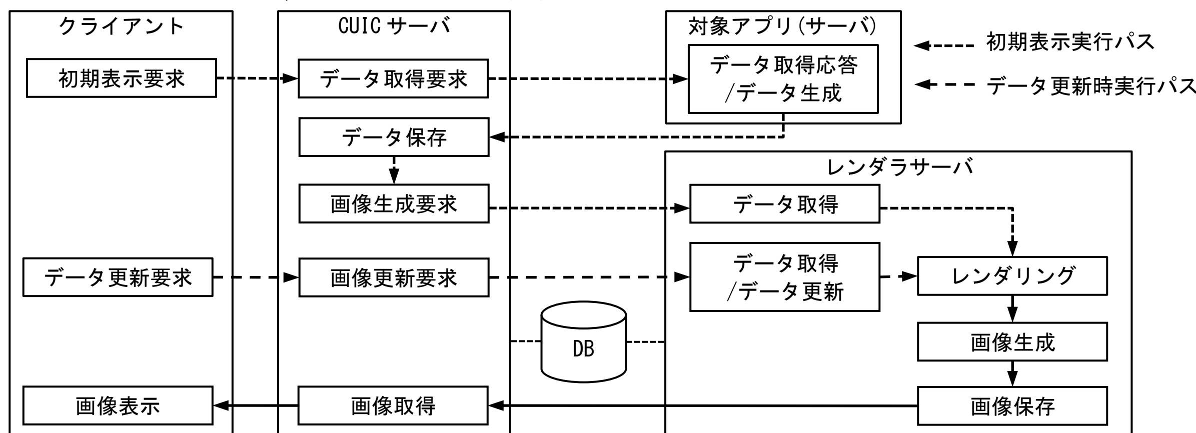


図 11 モバイルガントの画像更新フロー

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

データ更新要求では，サマリタスクの開閉による再描画を対象として評価を行う．この場合は初期表示時に保存しておいたデータに対して更新処理を行うこととなる．つまり，本実行パスでは対象アプリの処理自体が省略される．処理フローとしては，初めに CUIC サーバからレンダラサーバに対して画像更新要求を行う．この際，開閉するサマリタスクの情報も同時に渡す．続いて，レンダラサーバでは RDB に保存されたデータの更新を行い，更新後のデータを利用してレンダリングを行う．

4.2.3. 実験設定

対象アプリのデータ取得応答はデータ生成が実行される高負荷処理である．したがって，初期表示の画像更新では，本処理がボトルネックとなってしまい，スケーリングシステムの正確な評価ができない可能性がある．一方で，データ更新は対象アプリの処理が存在しないため，スケーリングシステム単体での性能評価が可能である．そこで，本評価では，データ更新時の画像更新を評価対象とする．

評価では，あるデータ規模に対して，レンダラサーバ数を変化させた場合の許容リクエスト数を確認する．また，評価データとして大規模プロジェクトに相当するデータ(大規模データ)と小規模プロジェクトに相当するデータ(小規模データ)の二つを用意し，それぞれについて評価を行う．各データ規模の詳細について表 2 に示す．併せて，サーバに負荷がかかっていない状況で計測した処理時間(標準処理時間)についても記載する．

評価手順は，初めに任意台数のレンダラサーバで構築されたモバイルガント環境に対し，クライアントからある期間内に指定回数のリクエストを送信する．全リクエストの処理完了後，各リクエスト間の処理時間が一定となっているか確認する．一定であった場合，送信するリクエスト数を増やして再度評価を行う．処理時間が一定でなかった場合，その評価の直前に行った評価のリクエスト数を許容リクエスト数とする．送信するリクエスト数の変化量は，標準処理時間を考慮し大規模データで 10，小規模データで 100 とした．

なお，ブラウザの仕様により同時に接続できる WebSocket の本数には制限がある．そこで，同時接続数の上限を 250 とし，それを超えるリクエストを送信する際は，全リクエストを送信するまでの間隔を狭めることで対応する．例えば，一分間に 500 リクエストを処理可能か調べる場合，実際には 30 秒間で 250 リクエストを送信する．

表 2 ガントチャートのデータ規模

データ種別	大規模	小規模
タスク数	600	25
プロジェクト期間	20 ヶ月	3 ヶ月
アクティビティ数/タスク	1	1
標準処理時間	2300(ms)	400(ms)

4.2.4. 評価環境

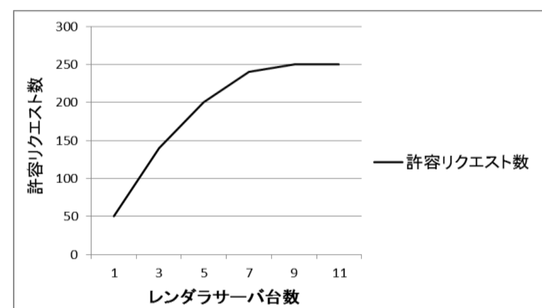
本評価は AWS 上の仮想環境で行った．評価に使用したリージョンおよび仮想マシン(EC2)の性能は表 1 と同じである．仮想 DB(RDS)の性能について表 3 に示す．本評価で用いる RDB の性能は対象アプリが使用している RDB も含めて全て表 3 で統一している．

表 3 RDS の実行環境

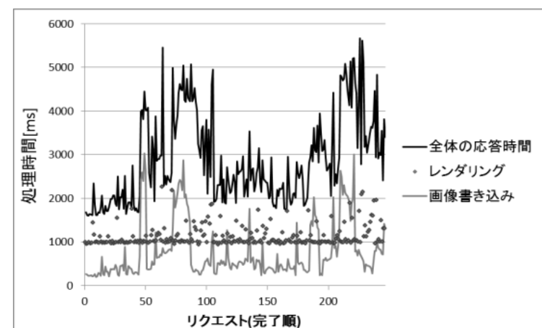
RDS	インスタンスタイプ		db.m3.large
		コア数	2
		メモリ(GB)	7.5
		ネットワークパフォーマンス	中
	DB エンジン		Oracle SE One 11.2.0.4.v6

4.2.5. 評価結果

大規模データの評価結果について，図 12 に示す．図 12 (a)は，レンダラサーバを 1 台から 11 台まで 2 台ずつ変化させた場合の許容リクエスト数である．横軸がレンダラサーバ数，縦軸が許容リクエスト数を表す．評価の結果，レンダラサーバ 1 台(スケールアウトなし)の場合の許容リクエスト数は 50 リクエストであった．また，レンダラサーバをスケールアウトすることによって 9 台目までは許容リクエスト数の増加が確認できる．一方で 9 台目から 11 台目にかけては，許容リクエスト数の増加は確認できなかった．この原因を明らかにするため，レンダラサーバ内の各処理の処理時間について調査した．図 12 (b)は，レンダラサーバ数 11 台，リクエスト数 260(許容リクエスト数以上)での各処理の内訳である．横軸が CUIC サーバで処理が完了したリクエストの順番，縦軸が各リクエストの処理時間である．本図によると画像書き込みの処理時間が不安定になっている．画像書き込みは RDB の性能に依存しており，この部分がボトルネックである場合，レンダラサーバの台数を増やしても許容リクエスト数は増加しない．したがって，許容リクエスト数が増加しなかった原因は RDB の性能限界に達したためと考えられる．RDB の並列化対応などで，モバイルガント全体の許容リクエスト数を更に引き上げることが可能である．



(a)許容リクエスト数の推移

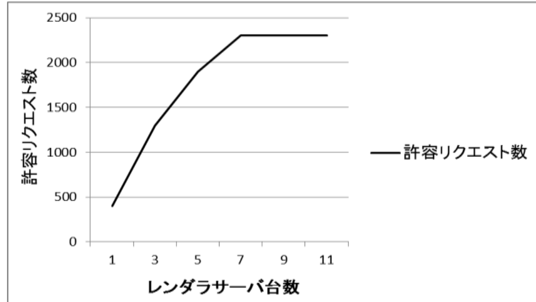


(b)処理時間の内訳

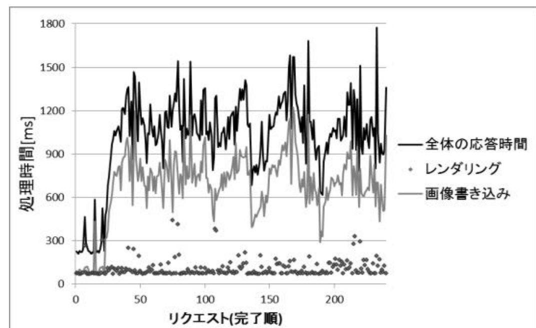
図 12 評価結果(大規模データ)

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

小規模データのサマリタスク開閉について，評価結果を図 13(a)に示す．結果として，7 台目までは許容リクエスト数が増加したが，以降は増加しなかった．これについて，レンダラサーバ数を 11 台，リクエスト数を 2400(許容リクエスト数以上)として，各レンダラサーバ内の処理時間を計測した(図 13(b))．大規模データと同様に画像書き込みの処理時間が不安定になっている．したがって，RDB がボトルネックとなっており，RDB を性能向上させることで許容リクエスト数が増加すると考えられる．



(a)許容リクエスト数の推移



(b) 処理時間の内訳

図 13 評価結果(小規模データ)

本評価では，モバイルガントのデータ更新時の許容リクエスト数について評価を行った．その結果，レンダラサーバのスケールアウトによって，許容リクエスト数が増加することを確認できた．ただし，レンダラサーバを増加させていくと，あるタイミングで RDB の性能限界に達し，それ以上の許容リクエスト数増加は見込めなくなる．これについては，RDB の性能を上げることで対応可能であり，クラウド環境であれば容易に実現できる．

4.3. 従来手法との比較評価

上述の評価より，レンダラサーバを適切にスケールアウトすることで許容リクエスト数が増加することを確認した．本評価では，同時接続可能なユーザ数について，提案手法(スケーリングシステム)と図 3 に示す従来手法を比較する．

4.3.1. 従来手法によるモバイルガントの負荷分散

図 14 に従来手法でモバイルガントを負荷分散した例を示す．提案手法との違いとして，LB がクライアントと CUIC サーバの間に存在する．またレンダリングを含めた全ての更新処理を CUIC サーバ内で行う．3.1 でも述べたとおり，本手法は，初期接続時に接続先のサーバが固定されてしまうため，ユーザ間で CUIC アプリの利用方法に差があるような(高負荷，低負荷ユーザが存在する)状況では効率的な負荷分散が難しい．

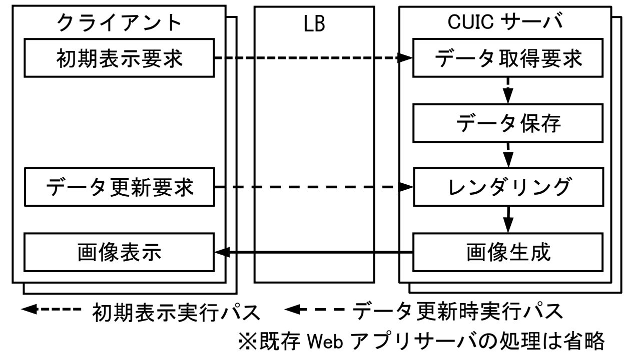


図 14 従来手法によるモバイルガントの負荷分散

4.3.2. 実験設定

本評価では，提案手法と従来手法のそれぞれで構築したモバイルガントに対して，どの程度のユーザが接続可能であるか調査する．ここで，接続可能であるとは，モバイルガントに接続したユーザが行った全ての画面更新要求に対して負荷による遅延なしで処理できることを指す．

接続するユーザは，高負荷ユーザもしくは低負荷ユーザのどちらかに属するものとする．また，両ユーザは以下に示す間隔で更新を行うものとする．

高負荷ユーザの更新間隔(頻度)：6 秒/回(10 回/分)
低負荷ユーザの更新間隔(頻度)：20 秒/回(3 回/分)

ただし，更新要求の発生タイミングは更新間隔の間でランダムである．例えば，高負荷ユーザの最初の更新は 0 秒から 6 秒までの間でランダムに発生する．

評価手順は，始めに任意数のクライアント(ユーザ)が CUIC サーバに初期接続を行う．この時，提案手法であれば接続する CUIC サーバが一台，レンダラサーバが複数台である．従来手法では，CUIC サーバが複数台用意されており LB が接続先を決定する．初期接続完了後，各クライアントからデータ更新要求を送信し，その処理時間を計測する．全てのデータ更新要求を遅延無く処理できている場合，そのクライアント数は同時接続可能であるとする．

本評価で使用するサーバの台数は，提案手法でレンダラサーバ 5 台とし，レンダラサーバの 1 台が CUIC サーバを兼ねるものとする．従来手法は CUIC サーバ 5 台構成とする．また，あるサーバに接続したユーザが高負荷ユーザである確率と低負荷ユーザの確率はどちらも 50%とする．ただし，システム全体では，高負荷ユーザと低負荷ユーザが常に同数となるよう調整する．使用する評価データは大規模データとし，評価環境は表 1 と表 3 に準拠する．

4.3.3. 評価結果

提案手法について，評価手順に基づき最大接続クライアント数を調査した．その結果，最大接続クライアント数は 24(高負荷ユーザ 12，低負荷ユーザ 12)であった．図 15 は上記条件下での各レンダラサーバの処理時間(レンダリングと画像生成)である．一時的に処理時間が増加する場合があるものの，その後すぐに収束することから，処理時間は安定していると言える．続いて，従来手法について，提案手法の最大接続数であるクライアント数 24 が接続可能であるか調査した．図 16 は各 CUIC サーバの処理時間の推移である．結果より，(CUIC)サーバ 5 で他のサーバに比べ大幅な処理時間の遅延が見られた．従来手法では，提案手法で処理可能であったクライアント数を処理できなかったと言える．

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

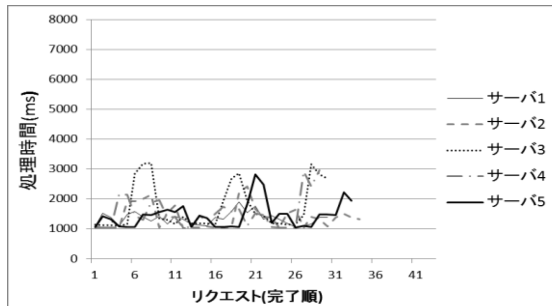


図 15 提案手法の評価結果

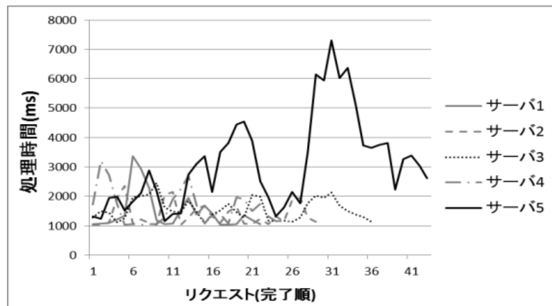


図 16 従来手法の評価結果

同じ接続数にも関わらず，従来手法でのみ処理時間にばらつきが見られた原因を明らかにするため，各手法で実行されたデータ更新回数を調査した．表 4 は提案手法の各レンダラサーバで行われた更新回数である．本結果から各サーバの処理回数はほぼ均一であることが分かる．表 5 は，従来手法で，各 CUIC サーバに接続したユーザ数とデータ更新処理の実行回数である．この時，各ユーザの接続先は LB が自動的に振り分けている．結果として，サーバ間で接続ユーザの合計数は同程度である．しかし，高負荷ユーザと低負荷ユーザの内訳はサーバごとに異なっており，それに伴い，処理の実行回数にも差が出ている．特に，大幅な処理遅延が見られたサーバ 5 には高負荷ユーザが集中している．特定のサーバに負荷が集中してしまったことが，従来手法の評価で処理時間が不安定となった原因であると言える．

表 4 提案手法での各サーバの更新回数

レンダラサーバ	1	2	3	4	5
データ更新回数	30	34	30	29	33

表 5 従来手法での各サーバの接続人数と更新回数

CUIC サーバ		1	2	3	4	5
ユーザ数	高負荷	1	2	3	2	4
	低負荷	4	3	2	2	1
	計	5	5	5	4	5
データ更新回数		22	29	36	26	43

本評価より，提案手法は，従来手法よりも効率的な負荷分散が可能であり，それによって，より多くのユーザが接続可能となることを確認した．ただし，サーバへの接続数と処理負荷が比例関係にある場合であれば，従来手法であっても提案手法と同程度のリクエストが処理可能になると考えられる．例えば，ユーザ間でアプリの使用方法に大きな差がないような場合や，実行する処理の負荷が非常に小さいアプリが該当する．

5. おわりに

CUIC アプリに複数ユーザが接続した場合であっても，十分な応答性能を確保するためには，画像更新時の負荷対策が必須である．本研究では CUIC アプリを適切に負荷分散する手法(スケーリングシステム)を考案した．スケーリングシステムは，サーバが実行する処理のうちレンダリングのみをスケールアウトすることで，WebSocket による高速な画像転送とレンダリング時の負荷対策を両立する．また，サーバ間の通信を画像の生成要求とその応答とすることで汎用的に利用可能である．スケーリングシステムの評価では応答性能評価と既存 Web アプリへの適用評価を実施した．その結果，レンダラサーバを適切にスケールアウトすることで許容リクエスト数が増加することを確認した．また，従来手法との比較評価では，スケーリングシステムによって従来手法より多くのユーザが接続可能となることが分かった．以上より，画面更新の頻度が異なる複数のユーザが接続された場合であっても，スケーリングシステムを利用することで，十分な応答性能が確保可能と言える．今後，この評価結果をもとに CUIC の実用化に向けて研究開発を進めていく予定である．

参考文献

[1] 内海 宏律， 黄 双全， 菊地 大介， 齋藤 邦夫，手塚 大， “サーバサイドレンダリングによる Web UI コンポーネントの試作と評価”， 情報科学技術フォーラム講演論文集 14(4)， pp.77-84, 2015

[2] RFC 6455 - The WebSocket Protocol , http://tools.ietf.org/html/rfc6455, Accessed 2016/2

[3] Valeria Cardellini, Michele Colajanni, Philip S. Yu, ”Dynamic Load Balancing on Web-server Systems”,IEEE Internet Computing 3(3), pp.28-39, 1999

[4]“NIST によるクラウドコンピューティングの定義”, https://www.ipa.go.jp/files/000025366.pdf, 2011

[5] Jinesh Varia , Amazon Web Services - Architecting for the Cloud: Best Practices , http://www.lifted-llc.com/docs/AWS_Cloud_Architecture_Best_Practices.pdf, 2011

[6] 土居 幸一郎， 後藤 滋樹，“HTTP セッションハンドオーバによる Web サーバのロードバランス”，情報処理学会研究報告インターネットと運用技術（IOT）2008(15(2008-DSM-048))，pp.25-29, 2008

[7] 竹田 仁， “待ち行列理論”， 文教大学大学院情報学研究科 IT News Letter vol.7, no.1, pp. 1～2, 2011

[8] 塩田 茂雄， 河西 憲一， 豊泉 洋， 会田 雅樹，“待ち行列理論の基礎と応用“， 川島 幸之助 監修，共立出版株式会社， 2014

[9] 古川 昌弘， 村田 大輔， 溝畑 潔， “e-Learning における負荷分散システムの運用”， THE SCIENCE AND ENGINEERING REVIEW OF DOSHISHA UNIVERSITY, vol. 54, no.2, 2013

[10]“Amazon Web Service (AWS) - Cloud Computing Services”, http://aws.amazon.com/, Accessed 2016/2

[11]“Apache JMeter”, http://jmeter.apache.org/, Accessed 2016/3

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.