サーバサイドレンダリングを用いたコンポーネントフレームワークの負荷分散

Workload Distribution for Component Framework using Server Side Rendering

近年，スマートデバイスの業務活用に対するニーズが高まっている。(株)日立ソリューションズ東日本では，スマートデバイス上で多くの計算量を必要とする可視化画面を表示するコンポーネントを開発するための Web UI コンポーネントフレームワーク(CUIC)を開発した。レンダリングをサーバで実行する CUIC は，サーバに負荷が集中するため負荷分散が必要になる。しかし，従来の負荷分散手法では，サーバ間で均等に負荷分散ができない問題があった。本研究では，CUIC の負荷分散を可能とするスケーリングシステムを考案した。評価の結果，効率的な負荷分散が可能になることを確認した。

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

1．はじめに

近年，スマートデバイスの普及に伴い，モバイルファースト[1)]が提唱されている。そうした中にあり，スマートデバイスの業務活用に対するニーズが高まっている。その一例として，PCでの操作を前提としたアプリケーションをスマートデバイスからも利用可能にするといったものがある(以下，モバイル対応)。こうした背景のもと，現在，企業で運用されている業務アプリのモバイル対応が求められている。

モバイル対応では，既存のWebアプリをスマートデバイス上で閲覧および操作するほか，ネイティブアプリであればマルチプラットフォーム対応のWebアプリ化してから利用するといった方法もある。どちらのWebアプリであっても，MVCアーキテクチャを適用し図1のような構成とすることが多い。本アーキテクチャでは，サーバは表示用のデータをXML[2)] やJSON[3)]などで送信する。その後，クライアントでレンダリングを実行する。

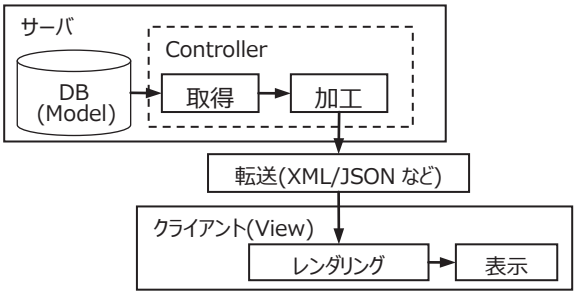

図 1 MVC アーキテクチャによる Web アプリの構成

業務アプリの多くは，数値や文字列で表現されたデータをグラフやチャートといった人間が理解しやすい形式に変換(以下，可視化)する機能を持つ。例として，作業日程をガントチャートで見せるものや，日ごとの在庫推移を棒グラフで可視化するといったものがある。可視化は大規模かつ複雑なデータの把握が容易になる反面，画面のレンダリングに多くの計算量を必要とする。そのため，性能の低いスマートデバイスではレンダリングに時間がかかり画面がスムーズに応答しないなどの問題が発生する。 このような課題を解決する手法の一つとして，スマートデバイス上で可視化画面を表示するWeb UIコンポーネントフレームワーク(以下，CUIC)がある[4)]。CUICの特徴として以下の二つがある。

(1) サーバサイドイメージレンダリングによるクライアントの負荷軽減

MVC アーキテクチャの Web アプリは，レンダラがクライアントに存在し，レンダリングをクライアントで行う場合が多い。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，実用的な応答性能を確保できない。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(図 2)。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だけとなり，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。

(2) WebSocket による通信の高速化

CUIC の画像転送では，クライアントの画面表示に必要な部分だけを送信し，スクロール操作などで未転送の領域が出現する都度，差分を転送する。画像の転送速度は UI 操作の応答時間に影響するため，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 がよく利用されている。しかし，HTTP は通信のたびにコネクションを確立するため，必要な応答時間が確保できない。そこで，CUIC は WebSocket[5]を利用して通信を行う(図 2) 。WebSocket は一度接続を確立すると，ユーザが任意に切断するまでコネクションを維持する。CUIC では初期接続時に確保したコネクションを利用して画像転送を行うことで画像表示の遅延を抑止している。

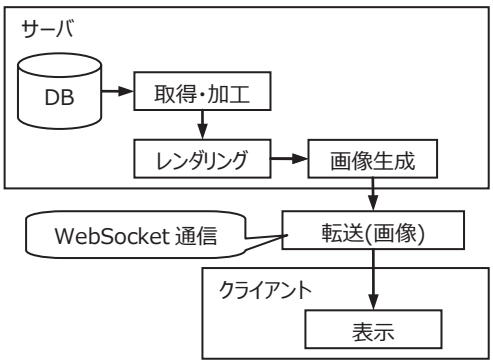

図 2 CUIC アーキテクチャによる Web システムの構成

CUICは, クライアントの負荷を軽減し既存のWebアプリのモバイル対応を容易にする。その一方で，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ため，サーバの負荷が高くなるといった問題がある。CUICの実用化に向けて，サーバの負荷対策が必要である。

2．負荷対策の課題

一般的な負荷対策の手法(以下，従来手法)として，スケールアウトによる負荷分散がある。本手法は，複数台の Web サーバとロードバランサ(以下，LB)で構成される。LB はクライアントからのリクエストを受け取ると，各サーバにリクエストを転送する。この際，リクエストの転送先はサーバ間で負荷が偏らないように決められる。

一方で，WebSocket は一度確立したコネクションを維持する。つまり，初期接続時に通信するサーバが決まると，以降の通信では別サーバを利用しない。そのため，ユーザの利用状況によっては，特定サーバだけ負荷が偏るといった状況が発生し得る(図 3)。ユーザ間の利用状況によらない負荷分散が CUIC の課題である。

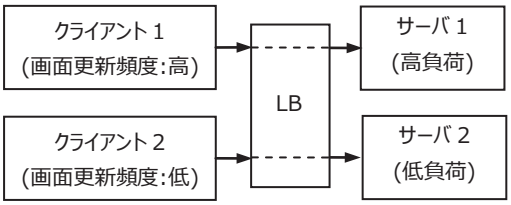

図 3 従来手法による CUIC のスケールアウトの例

3．スケーリングシステムの考案

本研究では，CUIC に適用可能なスケールアウト手法(以下，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概要を図 4 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 HTTP を用いる。CUIC サーバに残る処理は低負荷処理だけであるため，CUIC サーバ自体のスケールアウトが不要となる。これにより，クライアントと CUIC サーバ間の通信は従来どおり 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，画像取得のリクエストと，そのレスポンス(画像)とすることで特定のアプリに依存しない汎用的な通信になる。レンダラサーバでは，レンダリングの他に画像生成も実行する。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) WebSocket による高速な通信は従来どおり
(3) 特定のアプリに依存しない汎用的な設計

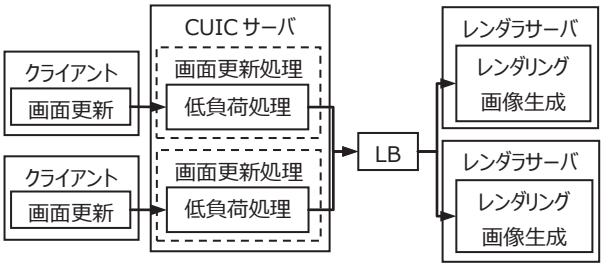

図 4 スケーリングシステムの構成

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を求められないため，これによる弊害は生じない。

4．スケーリングシステム評価

4.1 評価概要

スケーリングシステムの評価は，工程管理 Web アプリのモバイル対応したものを題材として実施した。本アプリケーションは，プロジェクトの工程管理を目的とした Web アプリである。メインの機能としてガントチャートの閲覧および編集ができる。一方で，ガントチャートは多くのオブジェクトをレンダリングする必要があるため，スマートデバイスでの閲覧が困難といった問題がある。モバイル対応では，ガントチャート部分に CUIC を適用してスマートデバイスでも閲覧可能なガントチャートアプリを試作した。

ガントチャートアプリの画面更新のフローを図 5 に示す。クライアントから画面更新のリクエストが届くと，始めに CUIC サーバで該当のデータが取得済みか確認する。初期表示時などでデータが存在しない場合は Web アプリからデータを取得する。ここで，取得したデータは CUIC サーバ内で保持する。再描画時は，このデータを利用するため，データ取得処理が発生しない。また，レンダラサーバで画像生成後，その画像は画像ストア用の DB に保存する。なお，本環境は Amazon Web Service（AWS）[6]を用いて構築した。

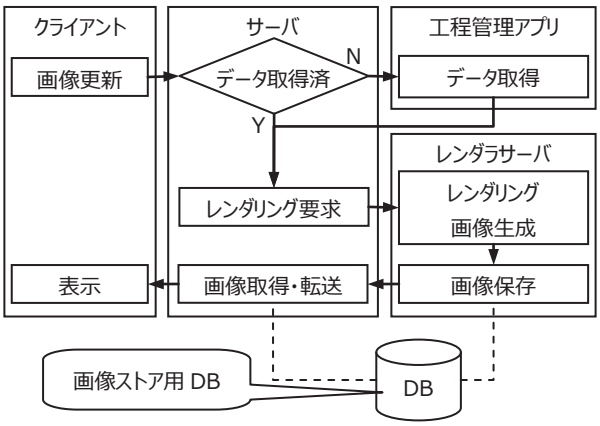

図 5 ガントチャートアプリの画面更新フロー

スケーリングシステムの評価として，「許容リクエスト数評価」と「従来手法との比較評価」の二つを実施した。以下，それぞれの評価について述べる。

4.2 許容リクエスト数評価

サーバに他の処理による負荷がかかっていない状態で計測したリクエスト 1 件分の処理時間を標準処理時間とする。処理時間にはばらつきがあることを考慮し，あるリクエストの処理が時間標準処理時間の 1.5 倍を越えた場合に遅延が発生したものとする。なお，ここではネットワーク遅延は含まない。連続で送信されるリクエストを単位時間内のまとまりとして見た際に，ある瞬間だけ突発的に遅延が発生することもあるが，こうしたケースは異常値として除外する。以上を考慮し，許容リクエスト数は，単位時間に任意数のリクエストを一定間隔で送信したとき，リクエストの 80%以上が遅延しないこととする。また，本評価では単位時間を 1 分とした。

本評価では，レンダラサーバを増加させることでガントチャートアプリの最大許容リクエスト数が増加することを確認する。使用したデータについて表 1 に示す。今回は，大規模プロジェクトと小規模プロジェクトを想定したデータをそれぞれ用意した。

表 1 評価データ概要

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	2325(ms)	410(ms)

大規模データの評価結果を図 6，小規模データの結果を図 7 に示す。図の横軸がレンダラサーバの台数，縦軸が最大許容リクエスト数を表す。両結果とも，レンダラサーバを増加させることで，最大許容リクエスト数が増加することが確認できた。しかし，大規模データでは 9 台目以降，小規模データでは 7 台目以降から増加しなくなった。この原因調査のため，追加で評価を行ったところ，画像ストア用 DB への IO がボトルネックとなっていることが判明した。そのため，必要に応じて DB の並列化対応などを行うことで，許容リクエスト数をさらに伸ばすことが可能である。

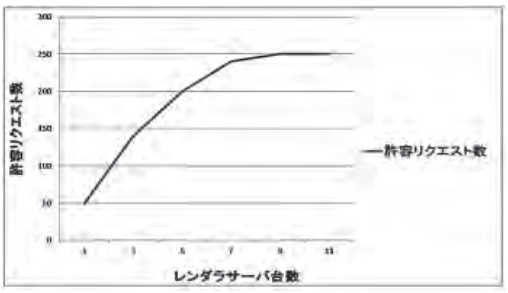

図 6 許容リクエスト数評価結果（大規模）

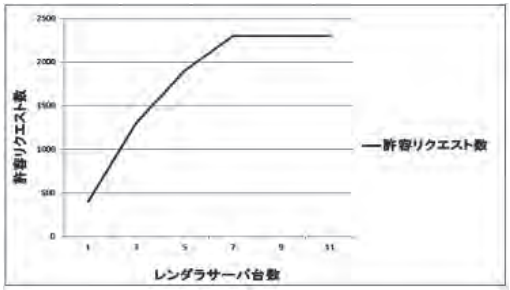

図 7 許容リクエスト数評価結果(小規模)

4.3 従来手法との比較評価

本評価では，ガントチャートアプリに対して，従来手法でスケールアウトを行った場合と，スケーリングシステムを利用した場合(以下，提案手法)で同時接続可能なユーザ数を比較する。全サーバの処理時間の中央値に対して，各サーバの平均処理時間が 1.5 倍以下であればそのサーバは遅延なし，全サーバで遅延がない場合を同時接続可能であるとする。各手法のサーバ構成について，図 8 および図 9 に示す。従来手法は，5 台の CUIC サーバを使って構築する。提案手法では，1 台の CUIC サーバと 4 台のレンダラサーバで構築する。ただし今回は，CUIC サーバにレンダラサーバも兼用させている。

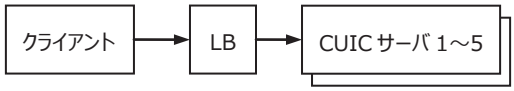

図 8 従来手法のシステム構成

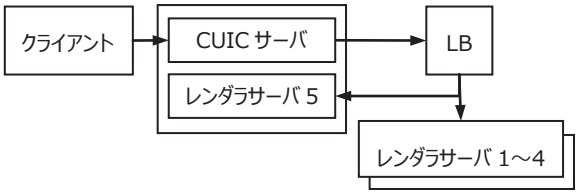

図 9 提案手法のシステム構成

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。両ユーザは以下に示す間隔で画面更新のリクエストを行うものとする。

高負荷ユーザの更新間隔：平均 6 秒/回

低負荷ユーザの更新間隔：平均 20 秒/回

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 WebSocket によるコネクションを確立する。このとき，サーバからは接続してきたユーザがどちらのどちらであるかは不明である。コネクション確立後，各クライアントから画面更新リクエストを送信し，遅延の有無を確認する。評価の結果，同時接続数 24(高負荷 12，低負荷 12)のときに従来手法で 1 台のサーバで遅延が発生した(表 2)。図 10 は上記条件の従来手法の各リクエスト処理時間の推移である。本図からもサーバ 5 だけで大幅な遅延が発生していることが分かる。

表 2 比較評価結果

サーバ	平均値(ms)	
	従来	提案
全サーバ中央値 (中央値×1.5)	1511 (2265)	1377 (2065)
1	1470	1344
2	1340	1399
3	1560	1729
4	1585	1540
5	**3110**	1447

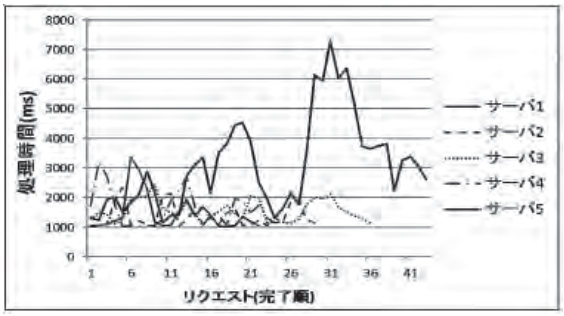

図 10 従来手法の評価結果

従来手法で，特定のサーバだけで遅延が発生した原因を調査するため，各サーバの接続ユーザ数と更新処理が実行された回数を調査した。結果を表 3 に示す。各サーバ間で接続ユーザの合計数は同程度であった。しかし，ユーザの内訳を見ると，大幅な遅延が見られたサーバ 5 に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遅延の原因であると言える。なお，提案手法についても同様に調査を行ったが，こちらは各レンダラサーバの更新処理実行回数に大きな偏りは見られなかった。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。

表 3 各サーバの接続人数と更新回数(従来手法)

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

5．おわりに

高機能な画面を持つ業務システムをスマートデバイスで実現可能とする Web UI コンポーネントフレームワーク CUIC を開発した。本研究では，CUIC の実用化に向けての課題であったサーバの負荷対策手法(スケーリン

グシステム)を考案した。LB の単純なラウンドロビンなどによるスケールアウトでは，ユーザの利用形態のばらつきにより，一部サーバだけに負荷が集中することがある。その結果，大部分のサーバでリソースに余裕があるにも関わらず，ユーザから見た性能が大幅に低下する場合がある。一方, 考案したスケーリングシステムは CUIC のシステム構成に合わせた最適なスケールアウトを行うことにより，サーバ間の負荷が均一となるため，従来手法と比較し効率的なサーバ運用が可能となる。

CUIC の事業適用に向けて，本稿でも触れたガントチャートアプリを試作した。外出先(現場)での進捗確認や実績入力をユースケースとし，SynViz S2 のガントチャートと同等の表現力を実現している。また，社内でのユーザビリティ評価をもとに機能改善を行った。今後は試行ユーザ環境での評価を実施し，より現場のニーズに合致したアプリケーションになるよう検討を進めていく。

また，Web ブラウザ上で高機能な画面を実現できる特長を活かし，(株)日立ソリューションズ東日本(HSE)の可視化技術の適用先拡大への活用を検討している。一例としてデータ統合・分析基盤である Pentaho[7]上で PSI Visualizer の可視化画面を実現するために CUIC を活用した試作を進め，今後評価していく。

こうした試作を通して，CUIC を用いた自社製品・ソリューションのモバイル対応を実現し, HSE の事業拡大とともに，顧客満足度の向上に貢献していく。

参考文献

1) “LukeW | Mobile First”, http://www.lukew.com/ff/entry.asp?933, Accessed 2016/9
2) “Extensible Markup Language (XML) 1.1”, http://www.w3.org/TR/xml11/, Accessed 2016/9.
3) “Introducing JSON”, http://www.json.org/, Accessed 2016/9.
4) 内海 宏律, 他, “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.
5) “RFC 6455 – The WebSocket Protocol”, http://tools.ietf.org/html/rfc6455 , Accessed 2016/9
6) “Amazon Web Service (AWS) – Cloud Computing Services”, http://aws.amazon.com/ , Accessed 2016/9
7) “Data Integration, Bussiness Analycs and BigData | Pentaho”, http://www.pentaho.com/, Accessed 2016/9

石倉 直弥 2009 年入社
研究開発部
情報ストレージ基盤技術の開発

naoya.ishikura.kk@hitachi-solutions.com

内海 宏律 2006 年入社
研究開発部
Web システムおよびソフトウェアコンポーネントに関する研究開発
hironori.utsumi.zc@hitachi-solutions.com

齋藤 邦夫 1992 年入社
研究開発部
自社パッケージ製品・ツールの研究・開発
kunio.saito.uh@hitachi-solutions.com

手塚 大 1994 年入社
研究開発部
研究戦略の立案，研究開発の推進

masaru.tezuka.fd@hitachi-solutions.com